МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

БАЗОВІ АЛГОРІТМИ ТА ОСНОВИ ПРОГРАМУВАННЯ. ТЕОРІЯ І ПРАКТИКА

О.Є. Тверитникова, В.А. Крилова, О.Г. Васильченков

Навчальний посібник

для студентів спеціальностей «Автоматизація та комп’ютерно-інтегровані технології», «Метрологія та вимірювальна техніка»
усіх форм навчання вищих навчальних закладів

Рекомендовано
редакційно–видавничою
радою університету,
протокол № 1 від 19.02.2020

Харків 2020

УДК 004.42 (075.8)
ББК 018 * 32.973я73
Т26

Рецензенти:

М.А. Мирошник, д-р техн. наук, проф. «Українська державна академія залізничного транспорту» м. Харків

Ю.Ф. Павленко, д-р техн. наук, професор
головний науковий співробітник ННЦ «Інститут метрології», м. Харків

Тверитникова О.Є.
Т 26 Тверитникова О.Є, Крилова В.А., Васильченков О.Г. Базові алгорітми та основи програмування. Теорія і практика : навч.-метод. посіб. Харків : НТУ «ХПІ», 2020. 264 с.

ISBN

У навчальному посібнику розглянуто основні поняття програмування та описані базові принципи розробки програмного забезпечення з використанням сучасних засобів програмування, наведено програмні реалізації практичних завдань, а також розглядаються питання алгоритмізації, типи алгоритмів і їх властивості.

Призначено для студентів денної та заочної форм навчання вищих навчальних закладів, які вивчають мову програмування С/С++.

Табл.12. Іл.56. Бібліогр. 21

УДК 004.42 (075.8)
ББК 018 * 32.973я73

ISBN 978-617-7859-53-5

ЗМІСТ

ПЕРЕДМОВА

Зростання продуктивності комп'ютерів, обсягів їх оперативної і зовнішньої пам'яті, пропускної здатності зовнішніх пристроїв і каналів зв'язку сильно змінив ситуацію в обчислювальній техніці і сферах її застосування. Зменшуються розміри комп'ютерів, споживання ними електроенергії, а швидкість обчислень зростає. Історично, основним завданням перших десятиліть появи і використання комп'ютерів являлася побудова апаратних комп'ютерних засобів. Це було обумовлено високою вартістю оброблення й зберігання даних. У 1980-ті роки успіхи мікроелектроніки привели до різкого збільшення продуктивності комп'ютера при значному зниженні вартості. Мова програмування *C*/*C*++ поступово стала універсальним інструментом та однією з найуживаніших мов програмування загального призначення.

Основним завданням наприкінці XX ст. – початку XXI ст. стало підвищення якості комп'ютерних програм, можливості яких цілком визначаються програмним забезпеченням. Знято практично всі апаратні обмеження на вирішення завдань. Решта обмежень припадають на частку програмного забезпечення. Проблеми використання програмного забезпечення в основному пов'язані з тим, що апаратна складність обчислювальної техніки і вимоги до нових програм випереджають розвиток розробки програмного забезпечення, а вимоги до експлуатації програмних продуктів не виконуються через низьку якість їх розробки. Вивчення програмування дозволяє зрозуміти процес розробки програмних продуктів, основні методики алгоритмізації завдань для подальшого їх подання у вигляді програмного забезпечення, отримати навички у використанні мов програмування. Комп'ютерні науки взагалі і програмна інженерія зокрема – дуже популярні і стрімко галузі знань, що розвиваються. Причиною є те, що людське суспільство XXI століття – інформаційне суспільство. У провідних країнах зайнятість населення в інформаційній сфері становить 60%, а в сфері матеріального виробництва – 40%. «Хто володіє інформацією – той володіє світом!».

Даний посібник містить основні поняття програмування, описує базові принципи розроблення програмного забезпечення, описує використання сучасних засобів для створення програм, а також надає початкові навички на мові програмування *C/C*++.

РОЗДІЛ 1.

СИСТЕМИ ЧИСЛЕННЯ

Біт – найменша одиниця вимірювання інформації. Біт (binary digit – двійкова цифра 0 або 1) – кількість інформації, що отримується в результаті однократного вибору з двох рівноймовірних подій.

Використання двійкової системи числення пояснюється тим, що для зберігання двійкової цифри необхідний елемент всього з двома стійкими станами, а також прості правила двійкової арифметики.

Інформація розміром в один біт міститься у відповіді на запитання, яке вимагає відповіді «так» чи «ні». У комп'ютерній техніці біт відповідає фізичному стану носія інформації: намагнічений – ненамагнічений. При цьому один стан прийнято позначати цифрою 0, а інший – цифрою 1.

Вибір одного з двох можливих варіантів дозволяє також розрізняти логічні «true» і «false». Послідовністю бітів можна закодувати текст, зображення, звук або яку-небудь іншу інформацію. Такий метод подання інформації називається двійковим кодуванням.

В інформатиці часто використовується величина, яка називається байтом (byte) і дорівнює 8 бітам. І якщо біт дозволяє вибрати один варіант з двох можливих, то байт, відповідно – один з 256 (28).

Бітом називають один двійковий розряд. Крайній зліва біт числа називають старшим розрядом (він має найбільшу вагу), крайній справа – молодшим (він має найменшу вагу). Багато типів ЕОМ і дискретних систем управління переробляють інформацію порціями (словами) по 16, 32 або 64 біта (1, 2 і 4 байта). Двійкове слово, яке складається з двох байт.

Як і для інших стандартних одиниць вимірювання для біта і байта існують похідні від них одиниці, утворені за допомогою приставок кіло (к), мега (М), гіга (G або Г), тера (Т), пета (Р або П) та інших. Але для бітів і байтів вони означають не степені 10, а ступені двійки (табл. 1.1).

Таблиця 1.1 – Похідні одиниці для біта і байта

Префікс	*Ступінь 2*	*Ступінь 10*
кіло (к)	2^{10}	$\approx 10^3$
мега (М)	2^{20}	$\approx 10^6$
гіга (G або Г)	2^{30}	$\approx 10^9$
тера (Т)	2^{40}	$\approx 10^{12}$;
пета (Р або П)	2^{50}	$\approx 10^{15}$

8 біт – 1 байт

1 кбайт – 1024 байт – 2^{10} байт

1 Мбайт – 1024 кбайт – 2^{10} кбайт – 2^{20} байт

1 Гбайт – 1024 Мбайт – 2^{10} Мбайт – 2^{20} кбайт – 2^{30} байт

1Тбайт – 2^{10} Гбайт – 2^{20} Мбайт – 2^{30} кбайт – 2^{40} байт

1 Пбайт – 2^{10} Тбайт – 2^{20} Гбайт – 2^{30} Мбайт – 2^{40} кбайт – 2^{50} байт

Система числення – символічний метод запису чисел, подання чисел за допомогою заданого набору спеціальних письмових знаків. Всі системи числення діляться на дві групи: *позиційні і непозиційні.*

У *непозиційних системах* числення значення цифри (вага, тобто внесок, який вона вносить у значення числа) не залежить від її позиції в записі числа. Наприклад, у римській системі числення в числі XXXII (тридцять два) вага цифри X у будь-якій позиції дорівнює десяти (10).

У *позиційних системах* числення значення цифри (вага) залежить від її положення в числі. Наприклад, у десятковій системі число 757 : перша цифра 7 – сім сотен, друга цифра 5 – п’ять десятків, третя цифра 7 – сім одиниць. Позиційні системи зручні тим, що вони дозволяють записувати будь-які числа за допомогою порівняно невеликого числа знаків. Ще більш важлива перевага позиційних систем – це простота і легкість виконання арифметичних операцій над числами, записаними в цих системах.

Позиція цифри в числі називається *розрядом*. Розряд числа зростає справа наліво, від молодших розрядів до старших. У десятковій системі цифра, що перебуває в крайній праворуч позиції (розряді), означає кількість одиниць, цифра, зміщена на одну позицію вліво, – кількість десятків, ще лівіше – сотень, потім тисяч і т.ін. Відповідно маємо розряд одиниць, розряд десятків і т.ін.

Кожна позиційна система характеризується певним алфавітом цифр і основою. Основа позиційної системи числення – кількість різних знаків і

символів, які використовуються для зображення цифр у даній системи числення. Значення будь-якого числа визначається не тільки розрядністю (номером позиції), але також «ваговим» значенням і алфавітом системи числення. Будь-яка позиційна система може бути подана поліномом :

$$d = a_n \cdot p^n + a_{n-1} \cdot p^{n-1} + ... + a_1 \cdot p^1 + a_0 \cdot p^0, \qquad (1.1)$$

де a – алфавіт системи числення, p – основа системи числення, n – вага розряду.

Наприклад. $789 = 7 \quad 10^2 + 8 \quad 10^1 + 9 \quad 10^0$.

Існують такі позиційні системи числення:

Десяткова система числення має алфавіт з десяти символів (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), основою системи є 10.

Двійкова система числення має алфавіт з двох символів (0, 1), основою системи є 2.

Вісімкова система числення має алфавіт з восьми символів (0, 1, 2, 3, 4, 5, 6, 7), основа системи дорівнює 8.

Шістнадцяткова система числення має алфавіт з шістнадцяти символів (0, 1, 2, 3 ... 8, 9, *A, B, C, D, E, F*), основа системи дорівнює 16.

Подання чисел у форматі з фіксованою і рухомою комою

Двійкові числа в обчислювальних пристроях розміщуються в комірках пам'яті і для кожного розряду числа призначається окрема комірка, що зберігає 1 біт інформації. Сукупність комірок, призначених для розміщення одного двійкового числа, називають розрядною сіткою. Кількість осередків n у розрядній сітці обмежене і залежить від конструктивних особливостей обчислювального пристрою. Розміщення розрядів числа в розрядній сітці проводиться різними способами. Спосіб розміщення визначається формою подання чисел в ЕОМ. Розрізняють дві форми подання двійкових чисел: з фіксованою комою і з рухомою комою.

Цілі числа в ЕОМ зберігаються в пам'яті в форматі з фіксованою комою. У тих ЕОМ, в роботі з якими користуються числами з фіксованою комою, застосовується звичайна форма запису чисел, тобто з постійною кількістю розрядів для цілої і дробової частини числа, отже, фіксація коми однакова для всіх чисел. Додавання і віднімання чисел з фіксованою комою проводяться за правилами звичайного двійкового додавання і

віднімання, оскільки результат операції не впливає на положення коми. Однак при виконанні множення і ділення необхідно здійснювати корекцію положення коми. Наявність додаткових обчислень при поданні дробових чисел у форматі з фіксованою точкою ускладнює розрахунки на ЕОМ. Недоліки формату з фіксованою комою – спостереження за положенням точки і порівняно невеликий діапазон зображених чисел – усуваються поданням чисел у форматі з рухомою комою.

Формат з рухомою комою використовується для розширення діапазону та зменшення відносної похибки подання чисел. У цьому форматі розряди числа розбиваються на два поля, що мають назви мантиса і порядок. Якщо позначити мантису буквою m, а порядок букв – n, то величина числа $A = \pm m \pm n$. Цей запис є еквівалентом форми запису десяткових чисел $A = m \cdot 10^n$, де m – множник, що містить всі цифри числа (мантиса), а n – ціле число (порядок).

Наприклад. $200 = 2 \cdot 10^2$, $36000000000 = 36 \cdot 10^9$.

Для виділення додатних і від'ємних чисел в ЕОМ використовується знаковий розряд, причому знак «+» позначається цифрою 0, а знак «–» – цифрою 1.

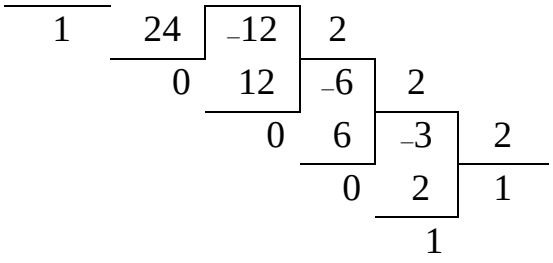
Перетворення з десяткової системи числення в двійкову, вісімкову, шістнадцяткову

Метод ділення. Для перетворення цілого числа з десяткової системи числення в будь-яку іншу позиційну систему необхідно розділити десяткове число на основу нової системи числення, потім отриману частку знову розділити на основу нової системи числення і так доти, поки в частці не залишиться число менш ніж основа нової системи числення.

Число в новій системі числення запишеться у вигляді залишків від ділення, починаючи з останньої частки. Тобто перший залишок дає молодшу цифру, а останній – старшу.

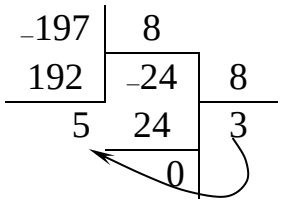
Приклад 1. Десяткове число 197_{10} перетворити в двійкову систему числення.

197	2			
196	98	2		
1	98	49	2	
	0	48	24	2



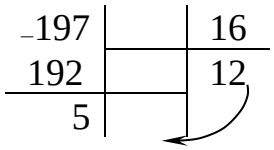
Отже, $197_{10} = 11000101_2$.

Приклад 2. Десяткове число 197_{10} перетворити в вісімкову систему числення.



Отже, $197_{10} = 305_8$.

Приклад 3. Десяткове число 197_{10} перетворити в шістнадцяткову систему числення. При перетворенні десяткового числа в шістнадцяткову систему треба враховувати, що алфавіт у шістнадцятковій системі числення, починаючи з 10 символу, має букви *A, B, C, D, E, F,* тому якщо в результаті ділення отримуємо числа більш ніж 9, їх треба переводити в символи шістнадцяткової системи.



Отже $197_{10} = C5_{16}$.

Метод віднімання. З десяткового числа віднімається найбільш можлива степінь двійки, у відповідний розряд двійкового числа записується одиниця, якщо різниця менше наступного степеня двійки, то далі записується нуль, а якщо більше – записується одиниця і знову проводиться віднімання, і так доти, поки вихідне число не зменшиться до нуля.

Приклад 4. Десяткове число 149_{10} перетворити в двійкове методом віднімання.

$$149_{10} = 10010101_2$$

$$\begin{array}{l} 149 \\ -\ \underline{128 = 2^7} \\ 21 \\ -\ \underline{16 = 2^4} \\ 5 \\ -\ \underline{4 = 2^2} \\ 1 \\ -\ \underline{1 = 2^0} \\ 0 \end{array}$$

Таким чином $149_{10} = 10010101_2$.

Приклад 5. Десяткове дробове число $685{,}5_{10}$ перетворити в двійкове методом віднімання.

$$685{,}5_{10} = 1010101101{,}1_2$$

$$\begin{array}{l} 685{,}5 \\ -\ \underline{512 = 2^9} \\ 173{,}5 \\ -\ \underline{128 = 2^7} \\ 45{,}5 \\ -\ \underline{32 = 2^5} \\ 13{,}5 \\ -\ \underline{8 = 2^3} \\ 5{,}5 \\ -\ \underline{2 = 2^4} \\ 3{,}5 \\ -\ \underline{2 = 2^1} \\ 1{,}5 \\ -\ \underline{1 = 2^0} \\ 0{,}5 \\ -\ \underline{0{,}5 = 2^{-1}} \\ 0 \end{array}$$

Таким чином $685{,}5_{10} = 1010101101{,}1_2$.

Метод множення. Даний метод застосовується для перетворення десяткових дробі, зокрема для чисел менших одиниці. При цьому число множиться на 2, якщо результат ≥1, то в старший розряд записується одиниця, якщо ні, то нуль. Множимо на 2 тільки дробову частину результату і повторюємо процедуру далі до отримання потрібного степеня точності або до обнулення результату.

Приклад 6. Десяткове число $0{,}321_{10}$ перевести в двійкове методом множення.

$0{,}321 \cdot 2 = 0{,}642$	– 0
$0{,}642 \cdot 2 = 1{,}284$	– 1
$0{,}284 \cdot 2 = 0{,}568$	– 0
$0{,}568 \cdot 2 = 1{,}136$	– 1
$0{,}136 \cdot 2 = 0{,}272$	– 0
$0{,}272 \cdot 2 = 0{,}544$	– 0

Отже $0{,}321_{10} = 0{,}010100_2$.

Приклад 7. Десяткове число $0{,}32812510_{10}$ перевести у вісімкове методом множення.

$0{,}328125 \cdot 8 = 2{,}625$	– 2
$0{,}625 \cdot 8 = 5{,}00$	– 5

Таким чином $0{,}32812510_{10} = 0{,}25_8$.

Приклад 8. Десяткове число $0{,}32812510_{10}$ перевести в шістнадцяткове методом множення.

$0{,}328125 \cdot 16 = 5{,}25$	– 5
$0{,}25 \cdot 16 = 4{,}00$	– 4

Таким чином $0{,}32812510_{10} = 0{,}54_{16}$.

Перетворення з двійкової, вісімкової, шістнадцяткової систем числення в десяткову

Для перетворення числа з системи числення з основою p в десяткову систему числення необхідно скористатися формулою 1.1 і кожній позиції числа присвоїти певну вагу. Потім значення ваги позиції множиться на коефіцієнт, що займає цю позицію. Результати операцій множення, виконаних для всіх позицій числа, підсумовуваються.

Приклад 9. Двійкове число 11001100_2 перевести в десяткову систему числення.

$$\overset{7}{1}\ \overset{6}{1}\ \overset{5}{0}\ \overset{4}{0}\ \overset{3}{1}\ \overset{2}{1}\ \overset{1}{0}\ \overset{0}{0}_2$$

$11001100_2 = 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 =$
$= 128 + 64 + 8 + 4 = 204_{10}$.

Приклад 10. Двійкове число $110111{,}11_2$ перевести в десяткову систему числення.

$$\overset{5}{1}\ \overset{4}{1}\ \overset{3}{0}\ \overset{2}{1}\ \overset{1}{1}\ \overset{0}{1},\ \overset{-1}{1}\ \ \overset{-2}{1}_2$$

$110111{,}11_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2}$
$= 32 + 16 + 4 + + 2 + 1 + 0.5 + 0{,}25 = 55{,}75_{10}$.

Приклад 11. Вісімкове число 537_8 перевести в десяткове.

$\overset{2}{5}\overset{1}{3}\overset{0}{7}_8 = 5 \cdot 8^2 + 3 \cdot 8^1 + 7 \cdot 8^0 = 320 + 24 + 7 = 351_{10}$.

Приклад 12. Вісімкове число $1172{,}25_{(8)}$ перевести в десяткове.

$\overset{3}{1}\overset{2}{1}\overset{1}{7}\overset{0}{2},\overset{-1}{2}\overset{-2}{5}_8 = 1 \cdot 8^3 + 1 \cdot 8^2 + 7 \cdot 8^1 + 2 \cdot 8^0 + 2 \cdot 8^{-1} + 5 \cdot 8^{-2} =$
$= 634{,}328125_{10}$.

Приклад 13. Шістнадцяткове число $3A2_{16}$ перевести в десяткове.

$\overset{2}{3}\overset{1}{B}\overset{0}{2}_{16} = 3 \cdot 16^2 + 11 \cdot 16^1 + 2 \cdot 16^0 = 768 + 160 + 2 = 946_{10}$.

Приклад 14. Шістнадцяткове число $27A{,}54_{16}$ перевести в десяткове.

$\overset{2}{2}\overset{1}{7}\overset{0}{A},\overset{-1}{5}\overset{-2}{4}_{16} = 2 \cdot 16^2 + 7 \cdot 16^1 + 10 \cdot 16^0 + 5 \cdot 16^{-1} + 4 \cdot 16^{-2} =$
$= 634{,}328125_{10}$.

Перетворення з двійкової системи числення в вісімкову і шістнадцяткову та навпаки

Для перетворення з двійкової системи числення у вісімкову, необхідно згрупувати (починаючи з молодшого розряду) по три біти (тріади), далі кожну групу записати однією вісімковою цифрою (табл. 1.2).

Таблиця 1.2 – Перетворення двійкових тріад у вісімкові цифри

Двійкові тріади	000	001	010	011	100	101	110	111
Вісімкові цифри	0	1	2	3	4	5	6	7

Приклад 15. Двійкове число 1111011011001_2 перевести в вісімкове:

$$\underbrace{001}_{1}\underbrace{111}_{7}\underbrace{011}_{3}\underbrace{011}_{3}\underbrace{001}_{1},$$

$001_2 = 1_8$; $011_2 = 3_8$; $011_2 = 3_8$; $111_2 = 7_8$; $001_2 = 1_8$.

Отже $1111011011001_2 = 17331_8$.

З цього прикладу видно, що старші розряди двійкового числа треба доповнювати нулями до 3-х розрядів (тріад) у двійковому коді.

Для перетворення з двійкової системи числення в шістнадцяткову, необхідно згрупувати (починаючи з молодшого розряду) по чотири біта (тетради), далі кожну групу записати однією шістнадцятковою цифрою (табл. 1.3).

Таблиця 1.3 – Перетворення двійкових тетрад у шістнадцяткові цифри

Двійкові тетради	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Шістнадцяткові цифри	0	1	2	3	4	5	6	7	8	9	*A*	*B*	*C*	*D*	*E*	*F*

Приклад 16. Двійкове число 111001011011001_2 перевести в шістнадцяткове.

$$\underbrace{0111}_{7}\underbrace{0010}_{2}\underbrace{1101}_{D}\underbrace{1001}_{9},$$

$1001_2 = 9_{16}$; $1101_2 = D_{16}$; $0010_2 = 2_{16}$; $0111_2 = 7_{16}$.

Отже $111001011011001_2 = 72D9_{16}$.

Для перетворення з вісімкової системи числення у двійкову, необхідно кожну цифру вихідного числа записати у вигляді еквівалентного трибітного двійкового числа (див. табл. 1.3).

Приклад 17. Вісімкове число 7265_8 перевести у двійкову систему числення.

$$\underbrace{7}_{111}\ \underbrace{2}_{010}\ \underbrace{6}_{110}\ \underbrace{5}_{101}$$

$5_8 = 101_2$; $6_8 = 110_2$; $2_8 = 010_2$; $7_8 = 111_2$.

Отже $7265_8 = 111010110101_2$.

Для перетворення з шістнадцяткової системи числення в двійкову, необхідно кожну цифру вихідного числа записати у вигляді еквівалентного чотирибітного двійкового числа (див. табл. 1.3).

Приклад 18. Шістнадцятирічне число $A2E5F_{(16)}$ перевести в двійкову систему числення.

$$\underbrace{A}_{1010}\ \underbrace{2}_{0010}\ \underbrace{E}_{1110}\ \underbrace{5}_{0101}\ \underbrace{F}_{1111}$$

$F_{16} = 1111_2$; $5_{16} = 0101_2$; $E_{16} = 1110_2$; $2_{16} = 0010_2$; $A_{16} = 1010_2$.

Таким чином $A2E5F_{16} = 10100010111001011111_2$.

Перетворення з вісімкової системи числення в шістнадцяткову і навпаки відбувається за допомогою двійкового коду. Для перетворення вісімкового числа в шістнадцяткову систему числення спочатку це число перетворюють у двійкову систему, потім, розбиваючи на тетради, починаючи з молодшого біта, перетворюють у шістнадцяткову за допомогою табл. 1.3. Для перетворення числа з шістнадцяткової системи у вісімкову це число перетворюють у двійкову систему, потім розбивають його на тріади, починаючи з молодшого біта, і замінюють тріади відповідними еквівалентами у вісімковой системі.

Приклад 19. Вісімкове число 2473_8 перевести в шістнадцяткову систему числення.

$$\underbrace{2}_{010}\ \underbrace{4}_{100}\ \underbrace{7}_{111}\ \underbrace{3}_{011}$$

$3_8 = 011_2$; $7_8 = 111_2$; $4_8 = 100_2$; $2_8 = 010_2$.

$$\underbrace{0101}_{5}\underbrace{0011}_{3}\underbrace{1011}_{B}$$

$1011_2 = B_{16}$; $0011_2 = 3_{16}$; $0101_2 = 5_{16}$.

Отже $2473_8 = 53B_{16}$.

Приклад 20. Шістнадцяткове число $CA5F_{16}$ перевести у вісімкову систему числення.

$$\underbrace{C}_{1100}\ \underbrace{A}_{1010}\ \underbrace{5}_{0101}\ \underbrace{F}_{1111}$$

$F_{16} = 1111_2$; $5_{16} = 0101_2$; $A_{16} = 1010_2$; $C_{16} = 1100_2$.

$$\underbrace{001}_{1}\underbrace{100}_{4}\underbrace{101}_{5}\underbrace{001}_{1}\underbrace{011}_{3}\underbrace{111}_{7}$$

$111_2 = 7_8$; $011_2 = 3_8$; $001_2 = 1_8$; $101_2 = 5_8$; $100_2 = 4_8$; $001_2 = 1_8$.

Таким чином $CA5F_{16} = 145137_8$.

1.2. Арифметичні дії в різних системах числення

Арифметичні операції в усіх позиційних системах числення виконуються за тими же відомими правилами, з якими працюємо в десятковій системі числення.

Арифметика в двійковій системі числення заснована на використанні таблиць додавання, віднімання та множення (рис. 1.1).

Таблиця ***додавання***

0	+	0	=	0
0	+	1	=	1
1	+	0	=	1
1	+	1	=	(1)0

перенесення одиниці до старшого розряду

Таблиця ***віднімання***

0	–	0	=	0
1	–	0	=	1
1	–	1	=	0
(1)0	–	1	=	1

позика одиниці зі старшого розряду

Таблиця ***множення***

0	·	0	=	0
0	·	1	=	0
1	·	0	=	0
1	·	1	=	1

Рисунок 1.1 – Таблиці додавання, віднімання та множення

Двійкове додавання виконується за тими же правилами, що і в десятковій системі числення, тобто порозрядно, але с тією лише різницею,

що перенесення одиниці в старший розряд проводиться після того, як сума досягне не десяти, а двох (10_2).

Приклад 1. Виконати додавання двійкових чисел $1101_2 + 1110_2$

$$\begin{array}{cccccc} & 1 & & & & \\ + & 1 & 1 & 0 & 1 & \\ & 1 & 1 & 1 & 0 & \\ \hline 1 & 1 & 0 & 1 & 1 & \end{array}$$

Отже, $1101_2 + 1110_2 = 11011_2$.

Зробимо перевірку в десятковій системі числення :

$\overset{3\,2\,1\,0}{1101}_2 = 1\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 = 13_{10}$;

$\overset{3\,2\,1\,0}{1110}_2 = 1\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 0\cdot2^0 = 14_{10}$;

$\overset{4\,3\,2\,1\,0}{11011}_2 = 1\cdot2^4 + 1\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 1\cdot2^0 = 27_{10}$.

Приклад 2. Виконати додавання двійкових чисел $10101{,}11_2 + 111{,}101_2$

$$\begin{array}{cccccccccc} & & 1 & 1 & 1 & 1 & & & & \\ + & 1 & 0 & 1 & 0 & 1 & , & 1 & 1 & \\ & & & 1 & 1 & 1 & , & 1 & 0 & 1 \\ \hline & 1 & 1 & 1 & 0 & 1 & , & 0 & 1 & 1 \end{array}$$

Отже, $10101{,}11_2 + 111{,}101_2 = 11101{,}011_2$.

Зробимо перевірку в десятковій системі числення :

$\overset{4\,3\,2\,1\,0\ -1-2}{10101{,}11}_2 = 1\cdot2^4 + 0\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 + 1\cdot2^{-1} + 1\cdot2^{-2} = 21 +$

$+ 0{,}75 = 21{,}75_{10}$.

$\overset{2\,1\,0\ -1-2-3}{111{,}101}_2 = 1\cdot2^2 + 1\cdot2^1 + 1\cdot2^0 + 1\cdot2^{-1} + 0\cdot2^{-2} + 1\cdot2^{-3} = 7 + 0{,}125 =$

$= 7{,}625_{10}$.

$\overset{4\ 3\,2\,1\,0\ -1-2-3}{11101{,}011}_2 = 1\cdot2^4 + 1\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 + 0\cdot2^{-1} + 1\cdot2^{-2} + 1\cdot$

$\cdot\, 2^{-3} = 29 + 0{,}625 = 29{,}325_{10}$.

Примітка. При додаванні кількох додатків необхідно стежити за одиницями перенесення в старші розряди, тому що ці одиниці можуть переходити не тільки в сусідні старші розряди, але і вище.

Приклад 3. Виконати додавання двійкових чисел $1111_2 + 1101_2 +$ $+ 10001_2 + 0111_2$.

Складаючи перший розряд отримують число 4, яке є трирозрядним двійковим числом 100. Отже, у цьому розряді буде 0, а перенесення одиниці роблять у 3-й вищий розряд. У 2-му розряді отримують 2, у цьому випадку перенесення роблять у сусідній вищий розряд. У 3-му розряді з урахуванням перенесення двох одиниць виходить число 5, яке дорівнює числу трирозрядному 101 у двійковій системі числення, тому одиницю в цьому розряді залишають, а 100 переносять через один розряд. У 4-му розряді отримують 2, отже, залишають 0, а одиницю переносять у сусідній вищий

розряд. У 5-му розряді отримують 3, яке дорівнює дворозрядному числу 11, одиницю залишають, а другу одиницю переносять у вищий розряд.

$$
\begin{array}{cccccc}
 & & +1 & 1 & 1 & 1 \\
 & & +1 & 1 & 0 & 1 \\
 & & +0 & 1 & 1 & 1 \\
 & 1 & 0 & 0 & 0 & 1 \\
\hline
1 & 1 & 0 & 1 & 0 & 0
\end{array}
$$

Отже, $1111_2 + 1101_2 + 10001_2 + 0111_2 = 110100_2$.

Зробимо перевірку в десятковій системі числення:

$1111_2 = 15_{10}$;

$1101_2 = 13_{10}$;

$10001_2 = 17_{10}$;

$0111_2 = 7_{10}$;

$15 + 13 + 17 + 7 = 52_{10}$.

$\overset{5\,4\,3\,2\,1\,0}{1\,1\,0\,1\,0\,0}_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 32 + 16 + 4 =$ $= 52_{10}$.

Додавання у шістнадцятковій системі числення виконується порозрядно, починаючи з молодших розрядів. Кожний символ

перетворюється в десяткову систему числення, потім виконується додавання, а результат обернено переводиться назад у шістнадцяткову систему.

Приклад 4. Виконати додавання двох чисел у шістнадцятковій системі числення $FB_{16} + C6_{16}$

$$\begin{array}{r} {\scriptstyle 1}\\ F\ B \\ +\ C\,6 \\ \hline 1C1 \end{array}$$

$B_{16} + 6_{16} = 11_{10}+6_{10} = 17_{10} = 16_{10}+1_{10} = 11_{16}$;

$F_{16} + C_{16} + 1_{16} = 15_{10}+12_{10}+1_{10} = 28_{10} = 16_{10}+12_{10} = 1C_{16}$;

перенесення з молодших розрядів

$FB_{16} + C6_{16} = 1C1_{16}$.

Зробимо перевірку в десятковій системі числення:

$FB_{16} = 15 \cdot 16^1 + 11 \cdot 16^0 = 251_{10}$;

$C6_{16} = 13 \cdot 16^1 + 6 \cdot 16^0 = 198_{10}$;

$251_{10} + 198_{10} = 449_{10}$;

$1C1_{16} = 1 \cdot 16^2 + 13 \cdot 16^1 + 1 \cdot 16^0 = 449_{10}$.

Приклад 5. Виконати додавання двох чисел у шістнадцятковій системі числення $FDB_{16} + 49F_{16} + C5A_{16}$

$B_{16} + F_{16} + A_{16} = 11_{10} + 15_{10} + 10_{10}= 36_{10} = 16_{10} + 16_{10} + 4_{10} = 24_{16}$;

$D_{16}+9_{16} + 5_{16} + 2_{16} = 14_{10} + 9_{10} + 5_{10} + 2_{10} = 29_{10}=16_{10} + 14_{10} = 1D_{16}$;

перенесення з молодших розрядів

$F_{16} + 4_{16} + C_{16} + 1_{16} = 15_{10} + 4_{10} + 13_{10} + 1_{10} = 32_{10} = 16_{10} + 16_{10} = 20_{16}$;

перенесення з молодших розрядів

$FDB_{16} + 49F_{16} + C5A_{16} = 20D4_{16}$.

Зробимо перевірку в десятковій системі числення:

$FDB_{16} = 15 \cdot 16^2 + 14 \cdot 16^1 + 12 \cdot 16^0 = 4059_{10}$;

$49F_{16} = 4 \cdot 16^2 + 9 \cdot 16^1 + 15 \cdot 16^0 = 1183_{10}$;

$C5A_{16} = 13 \cdot 16^2 + 5 \cdot 16^1 + 10 \cdot 16^0 = 12\ 3162_{10}$;

$4059_{10} + 1183_{10} + 3162_{10} = 8404_{10}$;

$20D4_{16} = 2 \cdot 16^3 + 0 \cdot 16^2 + 14 \cdot 16^1 + 4 \cdot 16^0 = 8404_{10}$.

При *відніманні двійкових чисел,* якщо віднімається 0 – 1, то в даному випадку займається 1 зі старшого розряду. Ця займана одиниця зі старшого розряду переходить у молодший як дві одиниці (тобто старший розряд подається двійкою більшого степеня) 2 – 1 = 1. Відповідь записуємо 1.

Приклад 6. Виконати віднімання двійкових чисел $11001_2 - 1101_2$

$$\begin{array}{r} -1\ 1\ 0\ 0\ 1 \\ 1\ 1\ 0\ 1 \\ \hline 1\ 1\ 0\ 0 \end{array}$$

Таким чином, $11101_2 - 1101_2 = 1100_2$.

Зробимо перевірку в десятковій системі числення:

$\overset{4\ 3\ 2\ 1\ 0}{1\,1\,0\,0\,1_2} = 25_{10}$;

$\overset{3\ 2\ 1\ 0}{1\,1\,0\,1_2} = 13_{10}$;

$25_{10} - 13_{10} = 12_{10}$;

$\overset{3\ 2\ 1\ 0}{1\,1\,0\,0_2} = 12_{10}$.

Приклад 7. Виконати віднімання двійкових чисел $11{,}01_2 - 1{,}1_2$

$$\begin{array}{r} -1\ 1\ ,\ 0\ 1 \\ 1\ ,\ 1 \\ \hline 1\ ,\ 1\ 1 \end{array}$$

Таким чином: $11{,}01_2 - 1{,}1_2 = 1{,}11_2$.

Зробимо перевірку в десятковій системі числення:

$\overset{1\ \ 0\ -1\ -2}{1\,1{,}0\,1_2} = 3{,}25_{10}$;

$\overset{0\ -1}{1{,}1_2} = 1{,}5_{10}$;

$3{,}25_{10} - 1{,}5_{10} = 1{,}75_{10}$;

$\overset{1\ -1\ -2}{1{,}1\,1_2} = 1{,}75_{10}$.

При *множенні в двійковій системі числення* двох n-розрядних чисел отримуємо 2^n – розрядний добуток. Множення виконується за допомогою операцій зсуву і додавання.

Приклад 8. Виконати множення двійкових чисел $111_2 \cdot 101_2$

$$\begin{array}{r} \cdot 1\ 1\ 1 \\ 1\ 0\ 1 \\ \hline \end{array}$$

			+1	1	1	
		+0	0	0		
	1	1	1			
1	0	0	0	1	1	

Отже, $111_2 \cdot 101_2 = 100011_2$.

Зробимо перевірку в десятковій системі числення:

$\overset{2\ 1\ 0}{111_2} = 7_{10}$;

$\overset{2\ 1\ 0}{101_2} = 5_{10}$;

$7_{10} \cdot 5_{10} = 35_{10}$;

$\overset{5\ 4\ 3\ 2\ 1\ 0}{100011_2} = 1\cdot2^5 + 0\cdot2^4 + 0\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 1\cdot2^0 = 32 + 2 + 1 = 35_{10}$.

Ділення двійкових чисел здійснюється за тими ж правилами, що й для десяткових. При цьому використовуються таблиці двійкового множення і віднімання.

Приклад 9. Виконати ділення двійкових чисел $101010_2 : 111_2$

– 101010	111	
111	110	
– 00111		
111		
0		

Отже, $101010_2 : 111_2 = 110_2$.

Зробимо перевірку в десятковій системі числення:

$101010_2 = 42_{10}$;

$111_2 = 7_{10}$;

$42_{10} : 7_{10} = 6_{10}$;

$110_2 = 6_{10}$.

Спочатку шукаємо в діленому число, починаючи від старшого розряду, яке було б більше ніж дільник. У даному примірнику це число 1010. Далі необхідно підібрати ділене цьому числу. Оскільки це цифра 0 або 1 та 1010 більш ніж 111, тому в частці пишемо першу 1. Множимо цю 1 на дільник, результат записуємо під ділене, дотримуючись розрядності. Виконуємо віднімання за правилами обчислення в двійковій системі числення. Зносимо наступну цифру діленого і отримане число порівнюємо

з дільником. У даному прикладі отримали число 111, яке дорівнює дільнику 111, тому в частці записуємо 1. Знову виконуємо віднімання і отримуємо 0. Але в діленому залишився останній розряд 0, тому в частці записуємо 0.

Отже відповідь 110.

Приклад 10. Виконати ділення двійкових чисел $110010_2 : 1010_2$

− 110010	1010
1010	101
− 001010	
1010	
0	

Таким чином, $110010_2 : 1010_2 = 101_2$.

Зробимо перевірку в десятковій системі числення:

$110010_2 = 50_{10}$;

$1010_2 = 10_{10}$;

$50_{10} : 10_{10} = 5_{10}$;

$101_2 = 5_{10}$.

Приклад 11. Виконати ділення двійкових чисел $11001_2 : 101000_2$

11001	101000
− 110010	0,101
101000	
−101000	
101000	
0	

Отже, $11001_2 : 101000_2 = 0{,}101_2$.

Зробимо перевірку в десятковій системі числення:

$11001_2 = 25_{10}$;

$101000_2 = 40_{10}$;

25	40
− 250	0,625
240	
−100	
80	

− 200
200
0

$25_{10} : 40_{10} = 0{,}625_{10}$.

Як видно з наведених прикладів, операція поділу може бути подана як операція порівняння, зсуву та підсумовування.

У вісімковій системі числення всі операції проводяться за тими ж правилами, за якими ці дії виконуються в десятковій системі числення. При виконанні операцій додавання і віднімання зручно використовувати вісімкову таблицю складання, при виконання операції множення використовуємо таблицю множення (додаток С).

Приклад 12. Додавання вісімкових чисел $741_8 + 252_8$

	7	4	1
	+ 2	5	2
1	2	1	3

Зробимо перевірку в десятковій системі числення:

$\overset{2}{7}\overset{1}{4}\overset{0}{1}_8 = 7\cdot 8^2 + 4\cdot 8^1 + 1\cdot 8^0 = 481_{10}$;

$\overset{2}{2}\overset{1}{5}\overset{0}{2}_8 = 2\cdot 8^2 + 5\cdot 8^1 + 2\cdot 8^0 = 170_{10}$;

$\overset{3}{1}\overset{2}{2}\overset{1}{1}\overset{0}{3}_8 = 1\cdot 8^3 + 2\cdot 8^2 + 1\cdot 8^1 + 3\cdot 8^0 = 651_{10}$;

$481_{10} + 179_{10} = 651_{10}$.

Приклад 13. Віднімання вісімкових чисел $346_8 - 154_8$

	− 3	4	6
	1	5	4
1	7	2	

Зробимо перевірку в десятковій системі числення:

$\overset{2}{3}\overset{1}{4}\overset{0}{6}_8 = 3\cdot 8^2 + 4\cdot 8^1 + 6\cdot 8^0 = 230_{10}$;

$\overset{2}{1}\overset{1}{5}\overset{0}{4}_8 = 1\cdot 8^2 + 5\cdot 8^1 + 4\cdot 8^0 = 108_{10}$;

$230_{10} - 154_{10} = 122_{10}$;

$\overset{2}{1}\overset{1}{7}\overset{0}{2}_8 = 1\cdot 8^2 + 7\cdot 8^1 + 2\cdot 8^0 = 122_{10}$.

Приклад 14. Виконати множення вісімкових чисел $31_8 \cdot 23_8$

$$\begin{array}{rrr} & \cdot 3 & 1 \\ & 2 & 3 \\ \hline 7 & 3 & 3 \end{array}$$

Зробимо перевірку в десятковій системі числення:

$\overset{1}{3}\overset{0}{1}_8 = 3 \cdot 8^1 + 1 \cdot 8^0 = 25_{10}$;

$\overset{1}{2}\overset{0}{3}_8 = 2 \cdot 8^1 + 3 \cdot 8^0 = 19_{10}$;

$25_{10} \cdot 19_{10} = 475_{10}$;

$\overset{2}{7}\overset{1}{3}\overset{0}{3}_8 = 7 \cdot 8^2 + 3 \cdot 8^1 + 3 \cdot 8^0 = 475_{10}$.

Приклад 15. Виконати множення вісімкових чисел $1170{,}64_8 \cdot 46{,}3_8$

$$\begin{array}{r} \cdot 1170{,}64 \\ \underline{\quad 46{,}3} \\ +\ 355\ 234 \\ +\ 7324\ 70 \\ \underline{47432\ 0} \\ 57334{,}134 \end{array}$$

Отже, $1170{,}64_8 \cdot 46{,}3_8 = 57334{,}134_8$.

Зробимо перевірку в десятковій системі числення:

$1170{,}64_8 = 8^3 \cdot 1 + 8^2 \cdot 1 + 8^1 \cdot 7 + 8^0 \cdot 0 + 8^{-1} \cdot 6 + 8^{-2} \cdot 4 = 632{,}8125_{10}$;

$46{,}3_8 = 8^1 \cdot 4 + 8^0 \cdot 6 + 8^{-1} \cdot 3 = 38{,}375_{10}$;

$632{,}8125_{10} \cdot 38{,}375_{10} = 24284{,}1796_{10}$;

$57334{,}134_8 = 8^4 \cdot 5 + 8^3 \cdot 7 + 8^2 \cdot 3 + 8^1 \cdot 3 + 8^0 \cdot 4 + 8^{-1} \cdot 1 + 8^{-2} \cdot 3 + + 8^3 \cdot 4 = 24284{,}1796_{10}$.

1.3. Прямий, додатковий та зворотний коди

В обчислювальній техніці з метою спрощення виконання арифметичних операцій застосовують спеціальні коди для подання чисел. Використання кодів дозволяє звести операцію віднімання чисел до арифметичного додавання кодів цих чисел. Застосовуються прямий, зворотний і додатковий коди чисел. Прямий код використовується для подання від’ємних чисел в запам'ятовувачі ЕОМ, а також при множенні і

діленні. Зворотний і додатковий коди використовуються для заміни операції віднімання операцією додавання, що спрощує пристрій арифметичного блоку ЕОМ. До кодів висуваються такі вимоги : – розряди числа в коді жорстко пов'язані з певною розрядною сіткою; – для запису коду знака в розрядній сітці відводиться фіксований та строго визначений розряд. Знаковим розрядом є крайній розряд у розрядній сітці.

Від'ємні десяткові числа при введенні в машину автоматично перетворюються на зворотний або додатковий двійковий код і в такому вигляді зберігаються, переміщуються і беруть участь в операціях. При виведенні чисел з машини відбувається зворотне перетворення в від'ємні десяткові числа. Від'ємні числа в прямому, зворотному і додатковому кодах мають різне зображення.

Прямий код двійкового числа являє собою код, отриманий прямим перетворенням числа з десяткової системи числення в двійкову та збігається з записом самого числа. Значення знакового розряду для додатних чисел дорівнює 0, а для від'ємних чисел – 1. Знаковий розряд відокремлюється точкою від розрядів двійкового коду числа. Додатні числа у всіх кодах зображуються однаково – двійковими кодами з цифрою 0 у знаковому розряді.

Приклад 1. Прямий код числа 6 і −6 (величина розрядної сітки $n = 4$):

$+6 \to 0.0110;$

$-6 \to 1.0110.$

знаковий розряд

Зворотний код. Зворотний код додатного числа збігається з прямим кодом. Зворотний код від'ємного числа отримується з прямого коду шляхом інверсії всіх його розрядів, окрім знакового. Для цього всі цифри числа замінюються на протилежні, а в знаковий розряд ставиться одиниця.

Приклад 2. Зворотний код числа –6 (величина розрядної сітки n = 4):

$+6 \to 0.0110$ – прямий код;

$-6 \to 1.0110$ – прямий код;

$-6 \to 1.1001$ – зворотний код.

Додатковий код. Додатковий код для додатного числа збігається з прямим кодом. Додатковий код від'ємного числа отримується зі зворотного коду шляхом додавання одиниці до його молодшого розряду.

Приклад 3. Додатковий код числа – 6 (величина розрядної сітки *n*=4):

$+6 \to 0.0110$ – прямий код;

$-6 \to 1.0110$ – прямий код;

$-6 \to 1.1001$ – зворотний код;

$+1.1001$

$\underline{\qquad 1}$

1.1010

$-6 \to 1.1010$ – додатковий код.

Особливості віднімання чисел у двійковій системі числення за допомогою додаткового коду

Віднімання в двійковій системі числення за допомогою додаткового коду замінюється операцією додавання прямого та додаткового кодів, або додаткового та додаткового кодів у випадку віднімання двох від'ємних чисел.

Наприклад, якщо за основу подання коду взято один байт, то для подання числа буде відведено 7 розрядів, а для запису коду знака – один розряд.

При додаванні чисел у прямому та додатковому кодах, якщо результат є додатним числом, виникає одиниця переносу у знаковому розряді, яка випадає за розрядну сітку та не впливає на результат. Перед тим як виконувати додавання двійкових чисел у прямому та додатковому кодах, треба зрівняти кількість розрядів у числах у двійковому коді до того, як перетворювати число з прямого коду в додатковий. Додавання відсутніх розрядів здійснюється шляхом написання нулів у старших розрядах.

Приклад 4. Виконати віднімання двох чисел у двійковій системі числення за допомогою додаткового коду $12_{10} - 7_{10}$.

Замінимо операцію віднімання операцією додавання:

$$12_{10} - 7_{10} = 12_{10} + (-7_{10}) = 5_{10}.$$

Перетворюємо числа з десяткової системи числення в двійкову:

$12_{10} = 1100_2$,

$7_{10} = 0111_2$.

Записуємо прямий код чисел:

$+12_{10} \to 0.1100_2$ – прямий код числа + 12_{10},

$-7_{10} \to 1.0111_2$ – прямий код числа – 7_{10}.

Отримуємо додатковий код числа -7_{10} у двійковій системі числення:

$-7_{10} \to 1.0111_2$ – прямий код;

1.1000 – зворотний код;

1.1001 – додатковий код.

Додаємо до прямого коду числа +12 додатковий код числа –7 за правилами додавання в двійковій системі числення:

$$\begin{array}{r} 0.1100 \\ + \quad 1.1001 \\ \hline 10.0101 \end{array}$$

одиниця випадає за розрядну сітку

$12_{10} - 7_{10} = 1100_2 - 1001_2 = 0101_2$.

Зробимо перевірку:

$0101_2 = 5_{10}$.

Приклад 5. Виконати віднімання двох чисел у двійковій системі числення за допомогою додаткового коду $122_{10} - 34_{10}$.

Замінимо операцію віднімання операцією додавання:

$122_{10} - 34_{10} = 122_{10} + (-34_{10}) = 88_{10}$.

Перетворюємо числа з десяткової системи числення в двійкову:

$122_{10} = 1111010_2$,

$34_{10} = 100010_2$.

Перед тим як перетворювати число –34 у додатковий код, треба підрахувати кількість розрядів у двійкових числах та зрівняти необхідну кількість розрядів, додавши нулі у старшому розряді.

$122_{10} = 1111010_2$ – 7 розрядів;

$34_{10} = 100010_2$ – 6 розрядів.

Існує різниця в один розряд, тому у двійковому коді числа 34 треба у старшому розряді додати 0:

$34_{10} = 0100010_2$.

Додаємо знаковий розряд та записуємо прямий код чисел:

$+122_{10} \to 0.1111010_2$ – прямий код числа $+122_{10}$;

$-34_{10} \rightarrow 1.0100010_2$ – прямий код числа -34_{10}.

Отримуємо додатковий код числа -34_{10} у двійковій системі числення:

$-34_{10} \rightarrow 1.0100010_2$ – прямий код;

1.1011101_2 – зворотній код;

1.1011110_2 – додатковий код.

Додаємо до прямого коду числа 122 додатковий код числа 34 у двійковій системі числення:

$$\begin{array}{r} 0.1111010 \\ +\ 1.1011110 \\ \hline 10.1011000 \end{array}$$

випадає за розрядну сітку.

$122_{10} - 34_2 = 1111010_2 - 100010_2 = 1011000_2$.

Зробимо перевірку:

$1011000_2 = 88_{10}$.

1.4. Основи алгебри логіки

Логіка – наука про закони і форми мислення. Математична логіка вивчає будь-які міркування за допомогою методів математики. Математична логіка входить до групи фундаментальних наук, які утворюють теоретичну основу інформатики.

Англійський математик Дж. Буль вперше застосував алгебричні методи для вирішення логічних задач.

Алгебра логіки – це розділ математичної логіки, значення всіх елементів (функцій і аргументів) якої визначені у двохелементному безлічі: 0 і 1. Алгебра логіки оперує логічними висловлюваннями. Істинне значення позначають одиницею (1) або символом *t* (*true* – істина), а хибне – нулем (0) або *f* (*false* – брехня). З двійковим кодом можна виконувати і логічні операції. Існують такі операції, які можна виконувати з інформацією : «И», «ИЛИ» і «НЕ» та ін. Використовуючи інформацію про істинність або хибність даних, які беруть участь у такій операції, вони визначають істинність або помилковість результату, керуючись логічними міркуваннями. Позначення логічних зв'язок (операцій) наведено в табл.4.1.

Таблиця істинності – це табличне зображення логічної операції, в якому перелічені всі можливі поєднання значень істинності вхідних сигналів (операндів) разом зі значенням істинності вихідного сигналу (результату операції) для кожного з цих поєднань (табл. 4.2, 4.3, 4.4, 4.5).

Таблиця 1.4 – Позначення логічних зв'язок (операцій)

Зв'язка	Назва	Логіка		програмування	
«И»	логічне множення (кон'юнкція)	Λ	*	&	*AND*
«ИЛИ»	логічне додавання (диз'юнкція)	V	+	|	*OR*
«НЕ»	інверсія (логічне заперечення)	¬	— (риса зверху)	!	*NOT*
«Исключающее ИЛИ»	додавання за модулем два	*a*^		⊕	*XOR*

Таблиця 1.5 – Таблиця істинності логічного множення

X	***Y***	***X & Y***
0	0	0
0	1	0
1	0	0
1	1	1

Таблиця 1.6 – Таблиця істинності логічного додавання

X	***Y***	***X | Y***
0	0	0
0	1	1
1	0	1
1	1	1

Таблиця 1.7 – Таблиця істинності логічного заперечення

X	$\overline{Y}$
0	1
1	0

Таблиця 1.8 – Таблиця істинності логічного додавання за модулем два

X	***Y***	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

Приклад 1. Обчислити логічний вираз: $m = \overline{\overline{x} \mid \overline{y \oplus z} \,\&\, x}$,

де $x = a[1]$, $y = a[3]$, $z = a[0]$, $a = 56_{10}$.

Перетворимо число *a* із десяткової системи числення в двійкову

$56_{10} = 111000_2$.

Задамо значення для *x*, *y* і *z*

$a_2 = \overset{5\,4\,3\,2\,1\,0}{1\,1\,1\,0\,0\,0}_2$,

$x = a[1] = 0$; $y = a[3] = 1$; $z = a[0] = 0$.

Записуємо вираз с заданими значеннями та за допомогою таблиць істинності розписуємо

$$m = \overline{\bar{0} \,|\, \overline{1 \oplus 0} \,\&\, 0} = \overline{1 \,|\, \bar{1} \,\&\, 0} = \overline{1 \,|\, 0 \,\&\, 0} = \overline{1 \,\&\, 0} = \bar{0} = 1.$$

Приклад 2. Обчислити логічний вираз $\mathrm{m} = x \oplus \overline{y \oplus z} \,|\, \overline{x \,\&\, y}$,

де x= a[6]; y = a[4]; $z = x \,\&\, y \,|\, a[2]$; $a_{10} = 583$.

Перетворимо число *a* із десяткової системи числення в двійкову:

$583_{10} = 1001000111_2$.

Задамо значення для *x*, *y* і *z*:

$a_2 = \overset{9\,8\,7\,6\,5\,4\,3\,2\,1\,0}{1\,0\,0\,1\,0\,0\,0\,1\,1\,1}_2$,

$x = a[6] = 1$; $y = a[4] = 0$; $z = 1 \,\&\, 0 \,|\, 1 = 0 \,|\, 1 = 1$.

Записуємо вираз із заданими значеннями та за допомогою таблиць істинності розписуємо

$$m = 1 \oplus \overline{0 \oplus 1} \,|\, \overline{1 \,\&\, 0} = 1 \oplus \bar{1} \,|\, \bar{0} = 1 \oplus 0 \,|\, 1 = 1 \,|\, 1 = 1.$$

Практичні завдання

І. ***Перетворення з однієї системи числення в іншу*** (табл. 1.9).

1. Перевести число $\boldsymbol{k_2}$ з двійкової системи числення у десяткову систему. Виконати перевірку, зробивши зворотне переведення.

2. Перевести число $\boldsymbol{l_8}$ з вісімкової системи числення у десяткову систему. Виконати перевірку, зробивши зворотне переведення.

3. Перевести число $\boldsymbol{m_{16}}$ з шістнадцяткової системи числення у десяткову систему. Виконати перевірку, зробивши зворотне переведення.

4. Перевести число $\boldsymbol{n_{10}}$ з десяткової системи числення у шістнадцяткову систему. Виконати перевірку, зробивши зворотне переведення.

5. Перевести число $\boldsymbol{p_{16}}$ з шістнадцяткової системи числення у вісімкову систему. Виконати перевірку, зробивши зворотне переведення.

ІІ. ***Арифметичні операції в двійковій системі числення*** (табл. 1.10)

1 Виконати додавання в двійковій системі числення. Перевірити результат у десятковій системі числення.

2 Виконати додавання в двійковій системі числення. Перевірити результат у десятковій системі числення.

3 Виконати множення в двійковій системі числення. Перевірити результат у десятковій системі числення.

4 Виконати віднімання в двійковій системі числення. Перевірити результат у десятковій системі числення.

Таблиця 1.9 – Практичні завдання

Число k_2	**Число l_8**	**Число m_{16}**	**Число n_{10}**	**Число p_{16}**
1101111	157	19*A*	948	*AB*9
1000111	452	*FD*45	279	45*F*
1110110	7765	392*E*	346	*ED*3
1001111	4443	*CA*3	244	569
1000110	445	57*B*2	864	4*B*8
1111101	2771	35*F*2	646	*AD*89
11011	3352	5*B*4	212	45*D*1
101111111	7235	78*C*	356	34*D*A
11100011	2356	541*E*	831	*C*567
1101101	6674	45*BB*	953	78*B*1

Таблиця 1.10 – Практичні завдання

1	**2**	**3**	**4**
110111 + 10011	11,11 + 111,011	1100 · 1010	10011 – 1011
1101011 + 1011	11,111 + 111,01	1011 · 1101	11010 – 0101
110111 + 11011	111,101 + 11,1	1010 · 1110	10001 – 0001
110101 + 11011	1111,1 + 11,101	1010 · 1101	10101 – 0111
110101 + 111011	101,101 + 110,1	10101 · 111	11111 – 1001
110111 + 10011	111,01 + 110,101	110 · 1110	10011 – 0111
110111 + 11111	111,101 + 111,1101	1011 · 1111	11101 – 1011
110101 + 1111	10,101 + 11,01	1111 · 110	11110 – 0011
11011 + 10011	111,01 + 110,11	100 · 1101	11100 – 1001
1101011 + 11011	111,11 + 111,101	10101 · 11	10111 – 1010

III. ***Арифметичні операції в різних системах числення*** (табл. 1.11)

1. Виконати додавання в вісімковій системі числення. Перевірити результат у десятковій системі числення.

2. Виконати віднімання в вісімковій системі числення. Перевірити результат у десятковій системі числення.

3. Виконати додавання в шістнадцятковій системі числення. Перевірити результат у десятковій системі числення.

4. Виконати ділення в двійковій системі числення. Перевірити результат у десятковій системі числення.

Таблиця 1.11 – Практичні завдання

1	**2**	**3**	**4**
343 + 424	5322 – 432	*A*2*DB* + 3*FEC*	10110 : 1011
422 + 423	7322 – 134	328*F* + *CB*26	11011 : 1010
532 + 643	6427 – 244	8*AF*7 + 9*ED*1	10011 : 1001
653 + 723	5322 – 375	3*CE*3 + 2*BCF*	11101 : 1111
462 + 734	1234 – 573	246*E* + *A*234	10101 : 1101
643 + 275	7532 – 421	*FE*71 + *A*46*B*	11110 : 0011
732 + 367	4352 – 212	29*AC* + 429*F*	10101 : 0101
324 + 532	6343 – 743	*D*3*B*6 + 1*F*8A	10001 : 1101
643 + 733	6437 – 432	36*F*9 + 8E*D*1	11010 : 1000
233 + 174	3332 – 643	435*A* + *DE*67	11101 : 0011

IV. Віднімання в двійковій системі числення (табл. 1.12).

1. Перевести число a_{10} в двійкову систему числення. Зробити перевірку.

2. Перевести число b_{10} в двійкову систему числення. Зробити перевірку.

3. Перевести число c_{10} в двійкову систему числення. Зробити перевірку.

4. Виконати віднімання в двійковій системі числення $a_2 - b_2$ за допомогою додаткового коду. Зробити перевірку в десятковій системі числення.

5. Виконати віднімання в двійковій системі числення $a_2 - c_2$ за допомогою додаткового коду. Зробити перевірку в десятковій системі числення.

Таблиця 1.12 – Практичні завдання

Число a_{10}	934	833	756	835	921	822	612	743	645	712
Число b_{10}	345	244	134	278	213	349	267	189	191	245
Число c_{10}	45	14	15	23	56	17	25	26	31	44

IV. ***Алгебра логіки*** (табл. 1.13).

1. Перевести число a_{10} в двійкову систему числення. Зробити перевірку.

2. Перевести число b_{10} в двійкову систему числення. Зробити перевірку.

3. Обчислити логічний вираз, $m = \overline{x | y \oplus z \,\&\, \bar{x}} \,|\, (z \oplus y) \,\&\, \bar{x}$,

де $x = a[5]; y = b[2]; z = x \oplus \bar{y} \,\&\, a[7]$.

Таблиця 1.13 – Варіанти завдання

Число a_{10}	456	566	498	541	589	590	601	678	578	615
Число b_{10}	675	233	451	2385	344	345	245	278	301	277

Контрольні запитання

1. Що таке система числення?

2. Яка система числення в обчислювальній техніці використовується як основна?

3. Які типи систем числення ви знаєте?

4. Чому система числення називається позиційною?

5. Які символи містить система з основою 8, 16?

6. Чим пояснити широке застосування двійкової системи числення?

7. Чому дорівнює вага молодшого розряду цілого числа?

8. Як пов'язана вага старшого розряду цілого числа з числом розрядів?

9. Чому перший залишок від ділення вихідного числа на основу нової системи є молодшим розрядом числа в новій системі?

10. Чому дорівнює вага старшого розряду дробу?

11. Яке найбільше десяткове число можна записати трьома цифрами: у двійковій системі; у вісімковій системі; у шістнадцятковій системі.

12. Яке найбільше натуральне число кодується 7 бітами?

13. Яким чином здійснюється переведення чисел, якщо основа нової системи числення дорівнює деякому степеню старої системи числення?

14. За яким правилом переводяться числа з десяткової системи числення?

15. За яким правилом переводяться числа в десяткову систему числення?

16. Арифметичні дії в двійковій системі числення.

17. Арифметичні дії в вісімковій системі числення.

18. Арифметичні дії в шістнадцятковій системі числення.

19. Додавання двійкових чисел з фіксованою комою.

20. Що таке прямий код?

21. Що таке зворотний код?

22. Що таке додатковий код?

23. Правила складання у додатковому коді.

24. Для чого використовуються операції «И», «ИЛИ»?

25. Як називаються операції «ИЛИ», «И», «НЕ»?

26. Як подається інформація про істинність або хибність у комп’ютері?

27. Таблиці істинності «И», «ИЛИ», «НЕ».

28. Таблиці істинності логічної операції «Исключающее ИЛИ».

.РОЗДІЛ 2.

ТЕХНОЛОГІЯ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ І АЛГОРИТМІЗАЦІЇ ЗАДАЧ

2.1. Основи технології розроблення програмного забезпечення

Технологія розробки програмного забезпечення це система інженерних принципів для створення економних програмних продуктів, які надійно і ефективно працюють в реальних умовах.

Технологія програмування – це система методів, способів і прийомів розробки і налагодження програм.

Відповідно до звичайного значення слова «технологія» під технологією програмування (programming technology) розуміється сукупність виробничих процесів, що приводить до створення необхідного програмного забезпечення, а також опис цієї сукупності процесів. Іншими словами, технологія програмування розуміється в широкому сенсі як технологія розробки програмного забезпечення, включаючи в неї всі процеси, починаючи з моменту зародження ідеї деякого програмного продукту.

Сучасна індустріальна технологія проектування програм включає в себе комплекс заходів, керівних документів та автоматизованих засобів, призначених для системного аналізу, розробки, налагодження, документування, управління роботою фахівців.

Для зменшення вартості виготовлення програмного продукту і підвищення продуктивності праці програмістів використовуються методи, які регламентують високу професійну культуру написання програм незалежно від мови, від системи, типу ЕОМ і розв'язуваної задачі. Такі методи отримали загальну назву – технології програмування.

Гарна технологія дає можливість отримати високий економічний ефект при її використанні, істотне зростання продуктивності праці програмістів, підвищує якість програмного продукту.

Технологія розробки програмного забезпечення включає в себе кілька основних етапів:

1. Постановка завдання.

2. Розроблення технічного рішення і розроблення системної специфікації програмного продукту.

3. Розроблення алгоритму розв'язання задачі.

4. Реалізація алгоритму з використанням мови програмування.

5. Тестування програмного продукту.

6. Налагодження програмного продукту.

7. Здача програмного продукту замовнику.

8. Супровід продукту.

Перераховані етапи є обов'язковими при розробці програмного продукту, що має комерційне застосування. Залежно від особливостей продукту, що розробляється деякі етапи можуть бути формальними, і, отже, не вимагати багато часу для реалізації. Крім того, на кожному етапі готується пакет документації, який використовується на подальших етапах.

Етап постановки завдання виконується в безпосередній взаємодії з замовником програмного продукту. Результатом даного етапу є технічне завдання на програмний продукт. Таке технічне завдання узгоджується і затверджується замовником і виконавцем проекту і включає в себе детальні вимоги до програмного продукту. Вихідні вимоги надаються замовником. Але замовник, як правило, не є фахівцем у галузі розробки програмного забезпечення, тому перелік вимог до програми є неповним і потребує уточнень з боку виконавця. Власне цим і визначається необхідність в цьому етапі. Можна розглянути наступний приклад.

Наприклад: є якийсь медичний діагностичний комплекс, який дозволяє отримати зображення внутрішніх органів людини, і лікар може визначити по зображенню наявність запалення в цих органах. Запалення зазвичай характеризується підвищеною температурою тканини. На зображенні температура тканин внутрішніх органів представлена кольором, ділянки з підвищеною температурою відображаються відтінками червоного кольору. Завдання, поставлене замовником, може звучати так: знайти загальну площу ділянок червоного кольору на зображенні, якщо ця площа перевищує деяке значення, то повинен бути зроблений висновок, що запалення присутнє. Така постановка завдання не містить низки важливих для розробки параметрів. Так необхідно визначити, в якому вигляді буде задана вхідна інформація: як відеосигнал,

як графічний файл певного формату, як дані, одержувані з комп'ютерної мережі і т.д. Також необхідно уточнити розмір зображення. Важливим питанням, яке має бути узгоджене, є визначення, який колір вважається червоним. Кожен колір в графічному комп'ютерному зображенні має власне числове значення і необхідно точно визначити перелік числових значень відтінків кольору, які будуть розпізнаватися як червоні. Також технічне завдання повинне визначати вид подання вихідної інформації. У представленому прикладі вихідна інформація може бути виведена на екран монітора в деякій графічній формі, записана в файл на зовнішньому носії, надрукована на принтері і т.д.

Технічне рішення визначає архітектуру програмного продукту – основні його частини і взаємодія цих частин. Крім того технічне рішення визначає набір методів, які будуть використовуватися для реалізації програмного продукту.

Наприклад, розробляється деякий програмний продукт, основне завдання якого видалення шумів на відеозображенні. Існує велика кількість методів цифрової обробки сигналів, які дозволяють вирішити таке завдання. Ці методи характеризуються швидкістю обробки зображення, якістю обробки і т.д. У технічному рішенні повинні бути визначені додаткові умови, які дозволяють вибрати оптимальний для даного випадку метод.

Основними документами на даному етапі є

- технічне рішення, яке описує архітектуру програмного продукту, аналіз можливих рішень і вибір одного з них, використання додаткового програмного забезпечення і т.д.,
- системна і функціональна специфікації продукту, яка включає в себе перелік всіх характеристик і особливостей програмного продукту, відповідності продукту стандартам і т.д..

2.2. Розроблення алгоритму розв'язання задачі

Етап розробки технічного рішення та розробки системної специфікації програмного продукту визначає, які функції повинен виконувати програмний продукт. Етап розробки алгоритмів розв'язання задачі визначає, як будуть реалізовані зазначені функції.

На цьому етапі завдання розбивається на складові частини. Визначаються функції кожної з частин та як ці функції будуть реалізовані. Важливим питанням, яке має бути узгоджений на цьому етапі, є питання взаємодії складових частин програмного продукту.

Можна розглянути такий приклад. Є велика корпоративна обчислювальна мережа, яка об'єднує велику кількість комп'ютерів, причому комп'ютери можуть бути віддалені один від одного на значні відстані (рис. 2.1). Така мережа включає в себе робочі місця співробітників, сервера зберігання даних і т.д. Частина комп'ютерів та іншого обладнання встановлена в головному офісі компанії, інша частина встановлена на виробничій ділянці, в сервісному центрі. Компанія також може мати одне або декілька віддалених регіональних представництв комп'ютери яких включені в загальну мережу.

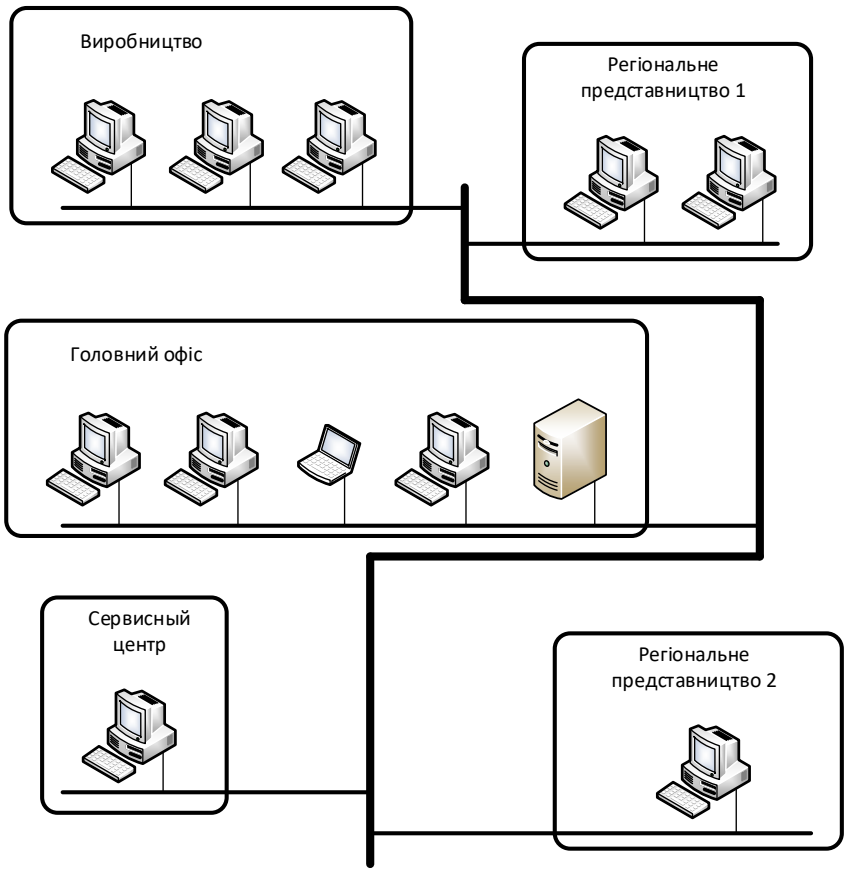


Рисунок 2.1 – Комп'ютерна корпоративна обчислювальна мережа

Для управління такою мережею створюється програмний комплекс, який дозволяє здійснювати моніторинг мережевих пристроїв, додавати і

видаляти мережеві пристрої в структуру мережі, управляти користувачами в мережі і т.д. Такий програмний продукт встановлюється на одному з комп'ютерів в головному офісі компанії, і системний адміністратор може віддалено виконувати описаний набір завдань управління обчислювальною мережею. Архітектура такого програмного продукту представлена на рис. 2.2.

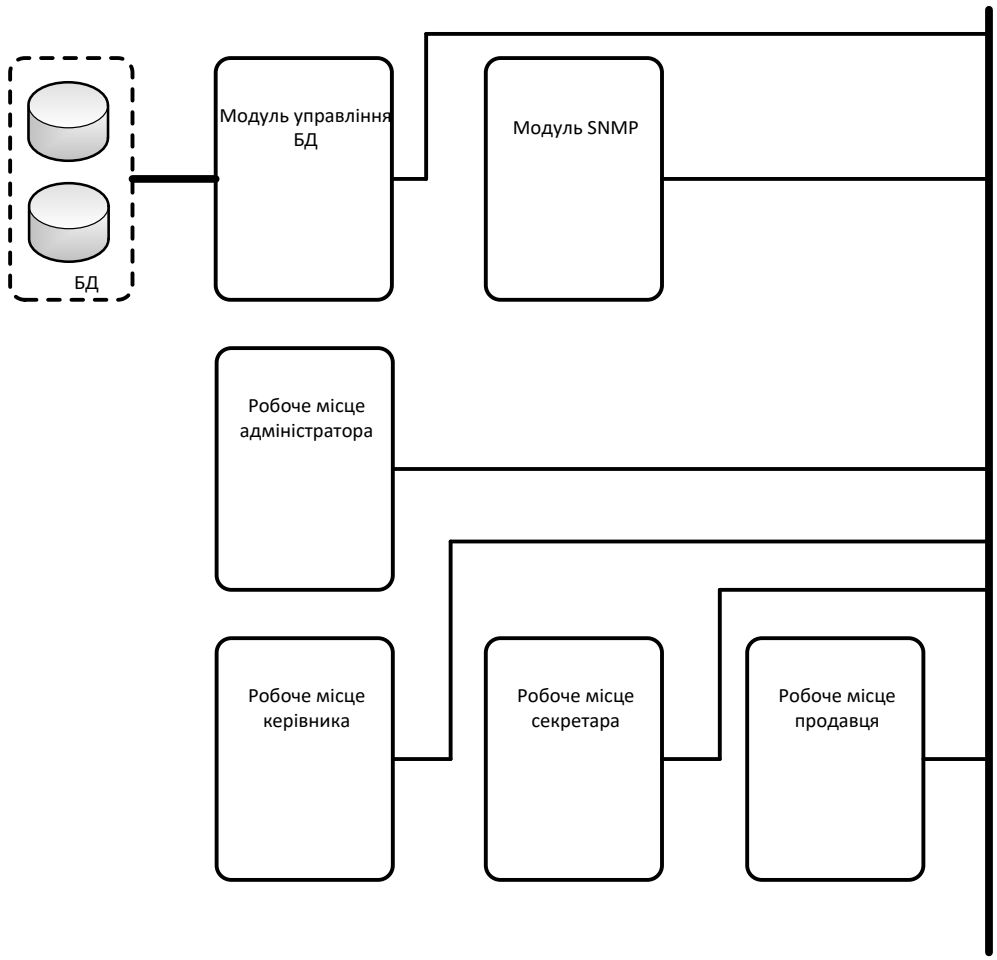


Рисунок 2.2 – Архітектура програмного комплексу

Програмний продукт повинен мати доступ до бази даних, де зберігається інформації про мережеві пристрої, про структуру обчислювальної мережі, про користувачів і т.д. До складу продукту входить програмний модуль, який забезпечує роботу з такою базою. Також є програмний модуль, який здійснює мережеву взаємодію з компонентами обчислювальної мережі. Для такої взаємодії можна використовувати стандартний протокол *SNMP* (*Simple Network Management Protocol*). Цей

протокол містить набір простих команд, за допомогою яких можна отримувати інформацію про мережеві пристрої, а також керувати ними. Крім того, програмний продукт містить програмний модуль, який реалізує інтерфейс користувача, тобто деякий набір функцій, за допомогою яких системний адміністратор може дистанційно керувати мережевими пристроями. Такий інтерфейс може мати графічну оболонку, меню, діалогові вікна тощо Всі перераховані модулі об'єднані програмним менеджером, який здійснює управління спільним функціонуванням програмного продукту.

На етапі розробки алгоритму роботи для кожного з перерахованих програмних модулів повинен бути підготовлений документ з описом даного модуля. Такий документ іноді називається дизайн–специфікацією. Документ детально описує набір функцій модуля, використання тих чи інших протоколів для реалізації модуля, засоби реалізації даного модуля, алгоритми реалізації модуля і т.д. У розглянутому вище прикладі повинна бути описана специфіка використання протоколу *SNMP*. Слід зазначити, що для великого програмного продукту різні його частина можуть бути реалізовані з використанням різних програмних технологій. Так в прикладі модуль *SNMP* може бути реалізований на мові C/C++, модуль взаємодії з базою даних може бути реалізований з використанням спеціальної мови запитів *SQL*, а модуль інтерфейсу користувача може бути реалізований на базі *Web* технологій (мови *PHP, Perl, Python* і т.д.). Обґрунтування вибору мови реалізації програмного модуля також має міститися в документі.

Реалізація алгоритму з використанням мови програмування є найпростішим етапом розробки програмного продукту, за умови дотримання і ретельного виконання вимог до описаних вище етапів. На цьому етапі пишеться програмний код на основі документації, розробленої на попередніх етапах. Код пишеться з використанням різних програмних засобів розробника.

Крім того, при розробці комерційного програмного продукту рекомендується дотримуватися правил оформлення програмного коду. Такі правила зазвичай розробляються для використання в проектах компанії розробника програмного забезпечення і є стандартом компанії. Правила визначають вимоги до іменування програмних об'єктів, оформлення операторів в програмах і т.д. Використання таких правил суттєво спрощує супровід програмних продуктів в подальшому.

2.3. Тестування й налагодження програмного продукту

Тестуванням програмного продукту є процес дослідження програмного забезпечення з метою отримання інформації про якість продукту. Тестування програмних продуктів є окремим напрямком в розробці програмного забезпечення зі своїми цілями і методологією. Метою тестування є пошук недоліків і помилок в програмному продукті. Для цього складається набір тестів для перевірки програми. Основною проблемою виконання тестування є можливість описати і виконати всі тести, з іншого боку виконання всіх тестів вимагає досить значного часу. Наприклад, в програмі необхідно ввести деяке з клавіатури прізвище як рядок довжиною до 30 літер. Для повного тестування такої функції необхідно здійснити введення всіх можливих поєднань всіх букв деякого алфавіту довжиною від 1 до 30 символів і при цьому перевіряти, що програма коректно відпрацьовує введені рядки. Зрозуміло, що число рядків, що вводяться буде дуже велике і виконати таке тестування за прийнятний час неможливо. Тому для перевірки програми виконують обмежений набір тестів, а які тести повинні увійти в цей набір визначається спеціальними правилами.

Види тестування мають власну класифікацію. Так по об'єкту тестування виділяють наступні види.

Функціональне тестування проводиться для перевірки наявності у програмного продукту особливостей, які затверджені в системній специфікації. Так, наприклад, програма повинна будувати графік функції, введеної користувачем. Функціональний тест повинен містити набір з декількох функцій для перевірки і результати у вигляді графіків, які будуть виводитися на екран. При збігу прикладів, що вводяться і одержуваних результатів робиться висновок, що дана функціональність працює коректно. До вибору перевірочних функцій також слід поставитися уважно. Функціональність буде вважатися непрацюючої, якщо, наприклад, при введенні функції $f(x) = x^2$ на екран буде виведена синусоїда. Крім того, необхідно включити в тестові приклади функції з обмеженою областю визначення – $1/x$ або $ln(x)$. Функція $1/x$ не може мати в якості аргументу значення 0, а функція $ln(x)$ може бути обчислена тільки для $x>0$. При виконанні розрахунків для таких функцій в програмі можуть виникати ситуації аварійного завершення – наприклад, ділення на 0. Такі ситуації

повинні оброблятися в програмі спеціальним чином. Якщо в результаті введення деякої тестової комбінації програма аварійно завершилася, то вважається, що тест не пройдений, а функціональність не працює.

Тестування навантаження дозволяє перевірити, як програмний продукт поводиться при інтенсивному і тривалому його використанні. Тестування *продуктивності* програмного продукту визначає, чи не виникають помилки при активній його роботі. Крім того, деякі програмні продукти повинні працювати тривалий час і загальний час роботи не повинно позначатися на роботі функціональностей. Така перевірка виконується тестуванням стабільності продукту. Для прикладу можна розглянути програму роботи деякого *Web* сайту. Програмне забезпечення сайту має обробляти запити декількох користувачів одночасно і віддавати коректну інформацію по кожному запиту. При виконанні тесту продуктивності, імітується звернення до сайту великого числа користувачів. Тест буде вважатися успішним, якщо сайт коректно відповідає на всі запити деякого заздалегідь визначеного числа користувачів, програма не завершує свою роботу аварійно і час відповіді користувачеві не перевищує деякого значення. Крім того програмне забезпечення сайту має працювати тривалий час без виникнення помилок. Якщо операції роботи з пам'яттю використовуються невірно, то при роботі програми може виникати т.зв. ефект «витоку пам'яті». Виникає він, коли пам'ять ЕОМ виділяється для якихось дій, і не звільняється після їх завершення. Оскільки кількість пам'яті кінцеве, то після тривалої роботи програми при виконанні операції виділення пам'яті може бути недостатньо і програма аварійно буде завершена. Така проблема може бути виявлена тестуванням програмного продукту на стабільність.

Тестування може проводитися в ручному або автоматичному режимах. В автоматичному режимі людина розробляє тестові сценарії, виконує деякі підготовчі операції і запускає тести на виконання. Далі тести виконуються без участі людини. Для виконання автоматичних тестів використовується спеціальне тестове програмне забезпечення. Використання автоматичного тестування дозволяє істотно збільшити кількість виконуваних тестів і більш повно оцінити якість програмного продукту.

По виду вихідних даних тестування буває позитивним або негативним. При позитивному тестуванні в якості вихідних даних

використовуються тільки коректні дані. Наприклад, при позитивному функціональному тестуванні вікна введення прізвища тестовий набір містить тільки рядки, що складаються з символів букв, причому перший символ повинен бути з великої літери. При негативному тестуванні тієї ж функціональності тестовий набір повинен включати строки, що містять символи цифр. При цьому програма повинна відповідним чином реагувати на такий введення.

За результатами тестування складається список знайдених помилок. Такий список є частиною програмної документації проекту. Кожна помилка в цьому списку отримує свій пріоритет за ступенем важливості. Найважливіші помилки отримують статус критичних, найменш важливі отримують статус незначних. Після цього приймається рішення про необхідність виправлення помилок. Критичні і важливі помилки повинні бути виправлені в будь–якому випадку. Деякі незначні помилки при дефіциті ресурсів на розробку можуть залишитися в продукті.

Етап тестування може бути повторений один або кілька разів, в залежності від кількості та важливості виявлених помилок. Такий процес називається регресійним тестуванням.

Для спрощення управління процесом тестування використовуються спеціалізовані програмні продукти – системи відслідковування помилок (bug tracking system). Такі системи дозволяють враховувати і контролювати помилки, а також стежити за процесом їх усунення.

На етапі налагодження програмного продукту виконується локалізація помилок програмного продукту і їх виправлення. Вихідним документом для цього етапу є список заявлених на етапі тестування помилок. Під локалізацією помилки розуміється пошук в програмному коді місця, де виникає та чи інша помилка.

Основними методами налагодження є:

- написання додаткового програмного коду для виведення користувачу інформації про стан програми в той чи інший момент її роботи, в подальшому, після усунення помилки, цей код видаляється;
- використання спеціальних програм – відладчиків (*debugger, bug* – помилка, *debug* – усунення помилки), така програма дозволяє спостерігати за виконанням досліджуваної програми, зупиняти і перезапускати її, проганяти в уповільненому темпі, змінювати значення в пам'яті і т.д.

Особливістю даного етапу є те, що кількість помилок в програмі заздалегідь невідомо, тому заздалегідь невідома тривалість налагодження. Кращим засобом для скорочення налагодження є оптимальні методи проектування програм, належне ведення програмної документації на всіх попередніх етапах, дотримання правил оформлення програмного коду.

Один з авторів мови *С* Брайан Керниган так писав про процес налагодження:

«*Налагодження складне і може займати непередбачувано багато часу, тому мета в тому, щоб мінімізувати цей час. Технічні прийоми, які допоможуть зменшити час налагодження, включають хороший дизайн, хороший стиль, перевірку граничних умов, перевірку правильності вихідних тверджень і розумності коду, захисне програмування, добре розроблені інтерфейси, обмежене використання глобальних змінних, автоматичні засоби контролю і перевірки. Грам профілактики вартує тони лікування* «.

Для виконання етапу здачі програмного продукту розробляється спеціальний документ – «Програма приймання». Даний документ містить опис умов, в яких слід приймати програмний продукт, перелік основних операцій, які треба виконати, щоб перевірити працездатність продукту і т.д. Документ затверджується замовником і виконавцем. Якщо програма відповідає перерахованим в документі функцій, то вважається, що програмний продукт прийнятий і це є підставою для взаєморозрахунків між сторонами, якщо продукт є комерційним.

Супроводом програмного продукту називається процес збору інформації про його якість в експлуатації, усунення виявлених в ньому помилок, його доопрацювання і модифікації, а також сповіщення користувачів про внесені до нього зміни.

Цей етап виконується, коли програмний продукт вже випущений в експлуатацію. Якщо продукт є досить складним, то всеосяжне його тестування виконати не можливо. Отже, продукт навіть в процесі його використання замовником може містити помилки. Розробник резервує частину свого робочого часу для виправлення помилок. Помилки виправляються і встановлюються в програму у вигляді оновлень.

Доброю ілюстрацією супроводу продукту є операційна система *Windows*. Кінцевим користувачам через мережу Інтернет розсилаються пакети оновлень у міру їх створення. Пакети можуть автоматично встановлюватися в системі. Коли кількість оновлень стає великою, то

випускається спеціальний сервісний пакет (*Service pack*), що включає всі зроблені модифікації і оновлення. Наприклад, для операційної системи *Windows XP* було випущено 3 сервісних пакета.

2.4. Поняття алгоритму та алгоритмізація задач

Алгоритмізація обчислювальної задачі є одним з найважливіших етапів розробки програмного продукту. На цьому етапі повинні бути розроблені алгоритми рішення основних частин програми. Основоположним є поняття алгоритму.

Алгоритм – точний набір інструкцій, що описують порядок дій виконавця для досягнення результату рішення задачі за кінцевий час.

Насправді точного єдиного визначення поняття «алгоритм» немає. Так, один із класиків програмування Д. Кнут дав таке поняття алгоритму:

«*Алгоритм – це кінцевий набір правил, який визначає послідовність операцій для вирішення конкретного множини завдань і має володіти п'ятьма важливими рисами: кінцевість, визначеність, введення, висновок, ефективність*».

Видатний радянський математик А.Колмогоров визначив алгоритм як «це будь–яка система обчислень, виконуваних по строго певним правилам, яка після будь–якого числа кроків свідомо призводить до вирішення поставленого завдання». Є й інші визначення цього поняття.

Алгоритми оточують людину всюди – від виконання професійних обов'язків до поведінки в побуті. Прикладом алгоритму може служити інструкція з експлуатації якогось домашнього приладу – електричного чайника, наприклад. В інструкції є розділ «Порядок застосування», в якому детально описані операції, за допомогою яких можна в чайнику з води отримати окріп. Алгоритмом є і інструкція, що описує дії механіка–водія необхідні для початку руху на танку. Такий документ містить близько 50 операцій. Ці операції включають перевірку параметрів систем танка, впливу на органи управління і, навіть, подачу звукового сигналу безпосередньо перед початком руху. Виконання деякої послідовності операцій може привести до того, що рух почати неможливо. наприклад, при невідповідності деяких параметрів двигуна номінальним, при яких можна почати рух. Якщо почати рух в таких режимах, то дорога техніка може бути пошкоджена. Подача звукового сигналу потрібна з міркувань

безпеки. Огляд у механіка-водія танка обмежений, і він може не помітити людей, що стоять поблизу танка. Подача звукового сигналу дозволить попередити їх.

Перегляд фільму в кінотеатрі теж можна представити у вигляді алгоритму:

1. Завітати до кінотеатру.
2. Вибрати відповідний фільм і сеанс.
3. Якщо вибір неможливий, то покинути кінотеатр, кінець алгоритму.
4. Підійти до каси.
5. Вибрати місце в залі.
6. Дізнатися ціну квитка.
7. Перевірити наявність грошей, якщо грошей недостатньо, то покинути кінотеатр, кінець алгоритму.
8. Купити квиток.
9. Очікувати початку сеансу.
10. Пройти в зал до місця, зазначеного у квитку.
11. Якщо місце вільно, то зайняти, інакше попросити звільнити ваше місце, після чого зайняти його (інакше, якщо ваше місце займає боксер, то знайти собі інше місце і зайняти).
12. Дивитися фільм;
13. Після закінчення сеансу покинути кінотеатр, кінець алгоритму.

Нижче наведено приклад обчислювального алгоритму. Необхідно скласти алгоритм, який дозволяє визначити, яке з трьох чисел має максимальне значення. Яким чином можна вирішити таке завдання? Очевидно, що визначити максимального значення з трьох чисел можна порівнюючи їх. Зробити це можна кількома способами.

На рис. 2.3 представлена схема, в якій попарно порівнюються перше і друге значення, і друге і третє. За результатами цих порівнянь визначаються два числа – максимальна з першої пари і максимальне з другої пари.

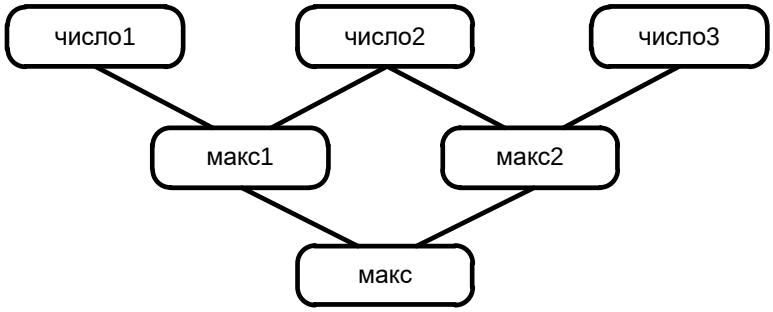


Рисунок 2.3 – Приклад обчислювального алгоритму

На наступному кроці порівнюється пара отриманих максимальних значень і результатом.

На рис. 2.4 представлена ще одна схема вирішення поставленого завдання.

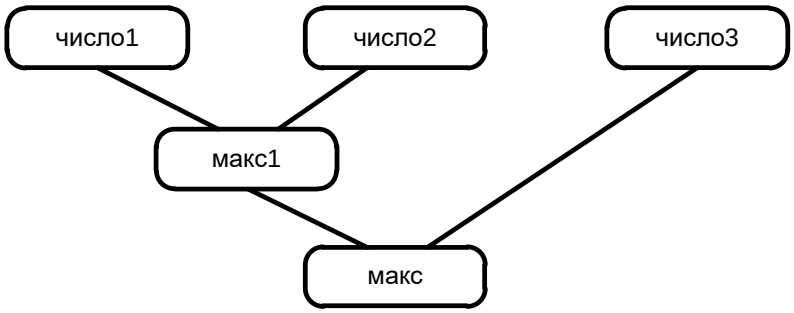


Рисунок 2.4 – Алгоритм розв'язання задачі

У такій схемі порівнюються перші два числа, і визначається максимальне значення. На наступному кроці отримане значення порівнюється з третім числом. Результат такого порівняння і буде максимальне значення з усіх трьох чисел.

Другий варіант є кращим, тому що в першому випадку треба виконати три операції порівняння, а в другому випадку тільки дві.

Наступна послідовність інструкцій представляє рішення задачі за другою схемою.

1. ВВЕДЕННЯ *x*, *y*, *z*
2. НЕХАЙ *u* = *x*
3. ЯКЩО *y*>*u*, ТО *u* = *y*
4. ЯКЩО *z*> *u*, ТО *u* = *z*
5. РЕЗУЛЬТАТ *u*

6. КІНЕЦЬ АЛГОРИТМА

Цей набір інструкцій є алгоритмом. Алгоритм містить шість операцій. Операції виконуються послідовно відповідно до порядку їх номерів.

В інструкції 1 визначені значення, які необхідно порівняти. Значення є вхідними параметрами алгоритму, тому виконується операція введення цих значень. Значення задані символьними іменами, що дозволяє використовувати даний алгоритм для будь–яких наборів, що складаються з трьох числових значень. Перше значення при цьому відповідає символьному імені *x*, друге число відповідає символьному імені *y*, а третє число відповідає символьному імені *z*.

В інструкції 2 використовується ще одне символьне ім'я – *u*. Цьому символьному імені встановлюється значення першого з порівнюваних чисел.

В інструкції 3 значення символьного імені *u* порівнюється зі значення другого числа вхідного набору. Оскільки *u* має значення *x*, то виконується порівняння перших двох значень вхідного набору. В інструкції зазначено, що якщо значення *y* більше, ніж значення *u*, то *u* приймає значення *y*. В іншому випадку значення *u* не зміниться і буде рівним значенню *x*. Таким чином, після виконання інструкції символьне ім'я *u* буде містити максимальне значення для перших двох параметрів вхідного набору.

В інструкції 4 виконується порівняння поточного значення *u* і третього числа, пов'язаного з символьним ім'ям *z*. Ця інструкція виконується також, як інструкція 3 і в результаті символьне ім'я *u* буде має значення, рівне максимальному значенню для всіх значень вхідного набору.

Інструкція 5 вказує, де можна отримати результат виконання алгоритму. В даному прикладі результатом є значення символьного імені *u*.

Інструкція 6 є завершальною інструкцією алгоритму.

До алгоритмів пред'являється ряд загальних вимог.

Алгоритм повинен бути дискретним. Це означає, що процес вирішення завдання повинен бути представлений у вигляді послідовного виконання деяких простих кроків (операцій, інструкцій). Виконання

кожного кроку виконується за кінцевий відрізок часу, і кожен виконаний крок наближає отримання результату.

Алгоритм повинен бути однозначним (певним, детермінованим). Це означає, що в кожен момент часу виконання алгоритму має бути однозначно визначено яка операція буде виконана наступної. Крім того, це означає, що алгоритм повинен видавати один і той же результат при одних і тих самих вихідних даних.

Алгоритм повинен мати властивість зрозумілості. Ця вимога полягає в тому, що операції, що входять до нього повинні бути зрозумілі виконавцю. При вирішенні завдань на ЕОМ алгоритм повинен містити операції, які збігаються з системою команд даного обчислювального пристрою.

Алгоритм повинен бути кінцевим (завершуваним) – алгоритм при коректно заданих даних повинен завершитися і видати результат за кінцеве число кроків.

Алгоритм повинен бути результативним, тобто виконання алгоритму має завершуватися певними результатами.

Алгоритм повинен бути масовим, що означає можливість використання алгоритму для різних наборів вихідних даних.

Крім того, слід розглядати таку проблему як ефективність алгоритму. У визначенні алгоритму вказується, що рішення задачі повинно бути досягнуто за кінцеве число кроків. На практиці алгоритм, який містить мільярд кроків, навряд чи може бути застосовний. Для вирішення однієї і тієї ж задачі може існувати безліч шляхів. При реалізації алгоритму потрібні додаткові умови, які є деякими обмеженнями і дозволяють вибрати оптимальний алгоритм. Такими умовами можуть бути максимально допустимий час виконання алгоритму, його розмір і т.д. У зв'язку з цим введено поняття складності алгоритму, яке широко використовують в інформатиці. Власне, коли говорять про складності алгоритму, то проводиться оцінка його ефективності.

Розглянутий алгоритм задовольняє перерахованим вище вимогам. Цей алгоритм дискретний, тому що містить набір послідовних операцій. Порядок виконання операцій однозначно визначено. Алгоритм є кінцевим і результативним (див. інструкції 5 і 6). Також алгоритм є масовим, тому що дозволяє обробляти різні вхідні набори даних, що складаються з 3 чисел.

Крім того, був обраний більш ефективний метод вирішення обчислювальної задачі.

У наступному прикладі необхідно обчислити значення полінома у при відомих значеннях x, a_3, a_2, a_1, і a_0

$$y = a_3x_3 + a_2x_2 + a_1x + a_0$$

Обчислення не представляє труднощів і може бути реалізовано таким алгоритмом

1. ВВЕДЕННЯ x, a_3, a_2, a_1, і a_0
2. НЕХАЙ $y = a_3*a_3 + a_2*x_2 + a_1*x + a_0$
3. РЕЗУЛЬТАТ y
4. КІНЕЦЬ АЛГОРИТМУ

З іншого боку відомо, що довільний поліном можна розкласти по схемі Горнера наступним чином

$$y = a_3x^3 + a_2x^2 + a_1x + a_0 = ((a_3x + a_2)\,x + a_1)\,x + a_0$$

Алгоритм, що дозволяє обчислити результат, буде таким

1. ВВЕДЕННЯ x, a_3, a_2, a_1, і a_0
2. НЕХАЙ $y = a_3$
3. НЕХАЙ $y = y*x + a_2$
4. НЕХАЙ $y = y*x + a_1$
5. НЕХАЙ $y = y*x + a_0$
6. РЕЗУЛЬТАТ y
7. КІНЕЦЬ АЛГОРИТМУ

Що дає використання схеми Горнера в такому алгоритмі?

Операції 3, 4 і 5 є однаковими, змінюється лише коефіцієнт ai. Використовуючи такий факт можна розробити алгоритм, що дозволяє обчислити значення полінома довільного ступеня.

$$y = a_n*x_n + a^{n-1}*x^{n-1} + \ldots + a_1*x + a_0 =$$
$$= ((\ldots (a_n\,x + a_{n-1})*x + a_{n-2})*x + \ldots + a_1)*x + a_0$$

1. ВВЕДЕННЯ n, x, a_i, $i = 0, \ldots n$

2. НЕХАЙ $i = n$
3. НЕХАЙ $y = a_n$
4. НЕХАЙ $i = i – 1$
5. ЯКЩО i <0, ТО ПЕРЕХІД п.8
6. НЕХАЙ $y = y*x + a_i$
7. ПЕРЕХІД п.4
8. РЕЗУЛЬТАТ y
9. КІНЕЦЬ АЛГОРИТМу

У такій реалізації алгоритму організовано циклічне виконання інструкцій. В інструкції 1 вводяться всі необхідні значення. В інструкції 2 задається символьне ім'я і, яке служить для циклічних операцій.

Початкове значення і дорівнює ступеню полінома. В ході виконання алгоритму значення і зменшується в інструкції 4. Алгоритм буде закінчений, коли це значення стане менше 0.

Перевірка значення виконується в інструкції 5. Якщо умова виконується, то значення полінома обчислено, у разі переходу до інструкції 8. Якщо умова не виконується, то наступною буде виконуватися інструкція 6. Таким чином, в алгоритмі реалізовано розгалуження – тобто послідовність виконання інструкцій змінена за допомогою інструкцій переходів.

В інструкції 7 виконується т.зв. безумовний перехід, коли без будь–яких умов вказується номер інструкції, яка буде виконуватися наступною. Це буде інструкція 4.

При виконанні алгоритму перший раз будуть послідовно виконуватися інструкції з 1 по 7. Потім буде виконаний перехід на інструкцію 4. Якщо ступінь полінома n>1, то циклічно будуть виконуватися інструкції з 4 по 7. Кількість повторень буде дорівнює ступеню полінома .

2.5. Способи подання алгоритмів

В процесі розробки програмного продукту складання і опис алгоритмів займає одне з ключових місць. Представлення розробленого алгоритму повинно мати якусь уніфіковану форму. Зазвичай уніфікація визначається стандартами компанії, що розробляє програмне забезпечення.

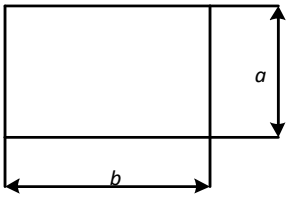
У свою чергу стандарти компанії ґрунтуються на деяких державних стандартах, а ті в свою чергу відповідають прийнятим міжнародним стандартам. Одним із способів загальноприйнятого уявлення алгоритмів є псевдокод. Вищенаведені алгоритми записані з використанням псевдокоду. Крім псевдокоду використовується опис алгоритмів у вигляді схем алгоритмів.

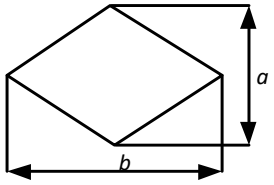
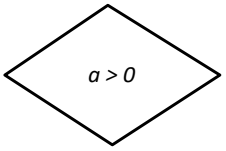
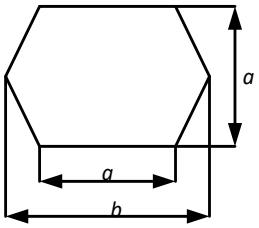
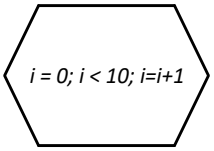
Для подання алгоритмів використовується стандарт, прийнятий ще в СРСР і дійсний в Україні – ГОСТ 19.701–90. Він відповідає міжнародному стандарту *ISO* 5897–85. Цей стандарт є частиною стандарту, який називається Єдина Система Програмної Документації (ЕСПД). Згідно з цим стандартом алгоритм представляється у вигляді схеми (блок–схеми). Окремі інструкції алгоритму подаються у вигляді умовних графічних позначень (УДО) або блоків. Порядок виконання алгоритму визначається лініями, що з'єднують блоки інструкцій.

Слід дотримуватися форму блоку, його розміри, а також конфігурацію вхідних і вихідних з'єднувальних ліній. Розміри блоків визначаються деяким базовим розміром а. Цей розмір визначається з ряду 5 мм, 10 мм, 15 мм, 20 мм і т.д. числами, кратними 5 мм. Решта розміри блоку пов'язані з а співвідношеннями, зазначеними на малюнках. Ще один базовий розмір визначається співвідношенням $b = 1{,}5a$.

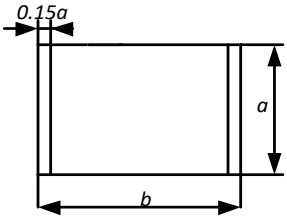
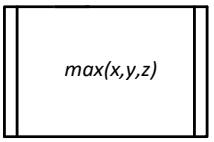
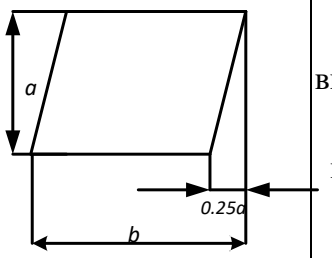
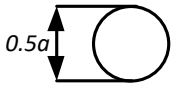
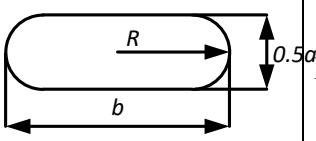
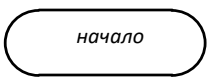
Основні інструкції алгоритмів описуються наступними блоками, представленими в табл. 2.6.

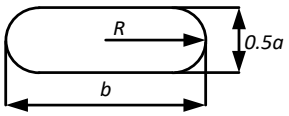
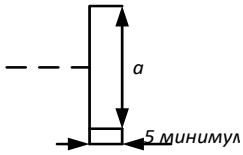
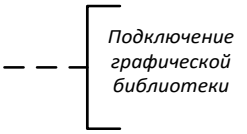
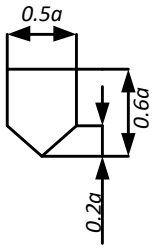
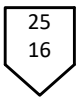
Таблиця 2.6 – Умовні графічні представлення основних інструкцій алгоритмів

Назва	Умовне графічне позначення	опис	приклад
Процес		Основний блок, містить виконання операції або групи операцій, вхід зверху, вихід внизу	$y = 2 * x^2$

Рішення, умовний блок		Вибір шляху виконання алгоритму в залежності від умови в блоці, вхід зверху, виходи з боків / внизу	
Модифікація		Рятувальна операція по автоматичної зміни змінної, використовується для організації циклічних процесів, вхід зверху, вихід до тіла циклу внизу, вхід з тіла циклу і вихід з циклу з боків	

Продовження таблиці 2.6

Зумовлений процес		Використання окремо описаних алгоритмів або програм, для виклику процедур / функцій, вхід зверху, вихід внизу	
Ввід/вивід		Використовується для введення даних в алгоритм і вивід результатів роботи алгоритму, вхід зверху, вихід внизу	
Внутрішньо–сторінковий з'єднувач		Використовується для розриву / з'єднання ліній в схемі алгоритму	
Початок алгоритму		Початок алгоритму, вихід внизу	

Кінець алгоритму		Початок алгоритму, вхід вгорі	
Коментар		Містить пояснювальний текст	
Міжсторін–ковий з'єднувач		Вказівка зв'язку між роз'єднаними частинами схем алгоритмів і програм, распо–викладених на різних аркушах	

Лінії зв'язку між блоками повинні бути паралельними кордонів листа і мають визначене спрямування. Блоки виконуються в послідовності зверху вниз і зліва направо. Якщо потрібно змінити напрямок виконання алгоритму лінія зв'язку повинна містити стрілку (рис. 2.5).

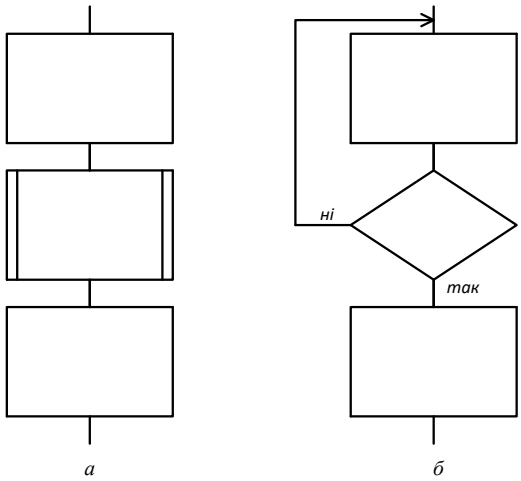


Рисунок 2.5 – Блок–схема алгоритму:
а – стандартна послідовність виконання блоків;
б – змінена послідовність

Для зручності опису блок–схеми кожен її блок слід пронумерувати. Зручно використовувати наскрізну нумерацію блоків. Номер блоку розташовують в розриві в лівій верхній частині рамки блоку (рис. 2.6). Початковий і кінцевий блоки не нумеруються.

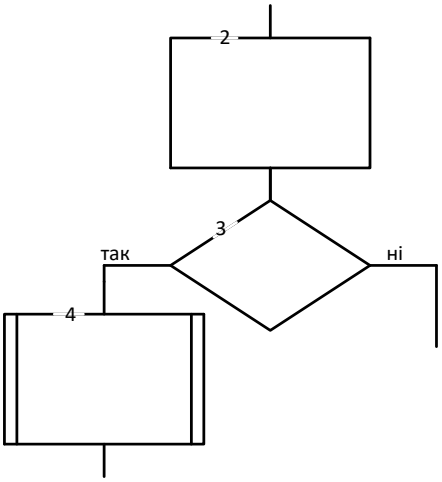


Рисунок 2.6 – Нумерація блоків алгоритму

Алгоритми бувають наступних типів: лінійні, що розгалужуються і циклічні.

Лінійний алгоритм – це алгоритм, в якому послідовність виконання блоків однакова при будь–якому наборі вхідних даних.

Прикладом лінійного алгоритму є наступний алгоритм обчислення довжини кола і площі круга *S* по відомому радіусу *R*.

Схема алгоритму представлена на рис. 2.7. Схема алгоритму складається з 6 блоків. Блоки виконуються послідовно відповідно до їх нумерації. У блоці 1 вводиться значення радіуса *R*. У блоці 2 обчислюється довжина кола *l* за вказаною формулою. У блоці 3 обчислюється площа кола *S*. У блоці 4 набуті значення виводяться на пристрій відображення, наприклад екран.

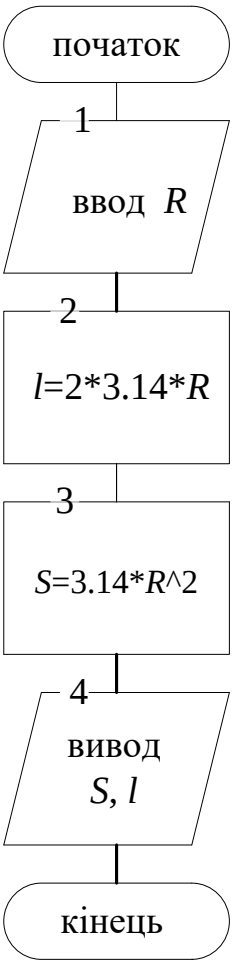


Рисунок 2.7 – Лінійний алгоритм

На практиці алгоритми лінійної структури зустрічається досить рідко. Найчастіше необхідно організувати процес, який в залежності від будь–яких умов проходить по тій чи іншій галузі алгоритму. Такий алгоритм називається той що розгалужується.

У блок–схемах розгалуження починається на виходах елемента «Рішення», за допомогою якого в алгоритмі виконується перевірка будь–якої умови. Кількість гілок тим більше, чим більше перевірених умов.

Як приклад можна розглянути алгоритм, який обчислює значення виразу $y = 1/x$, для значень, що вводяться користувачем значення x. Такий алгоритм схожий на наведений в прикладі вище, і створюється враження, що він може бути реалізований лінійно. Але, при обчислення значення виразу, треба спеціальним чином обробити ситуацію, коли $x=0$. Якщо допустити обчислення виразу при такому значенні x, то програма аварійно завершить роботу, і цього слід уникати. Таким чином, алгоритм повинен

не допускати обчислення виразу при x=0 і, крім цього, виводити користувачеві повідомлення, що виникла помилка. Схема такого алгоритму представлена на рис. 2.8.

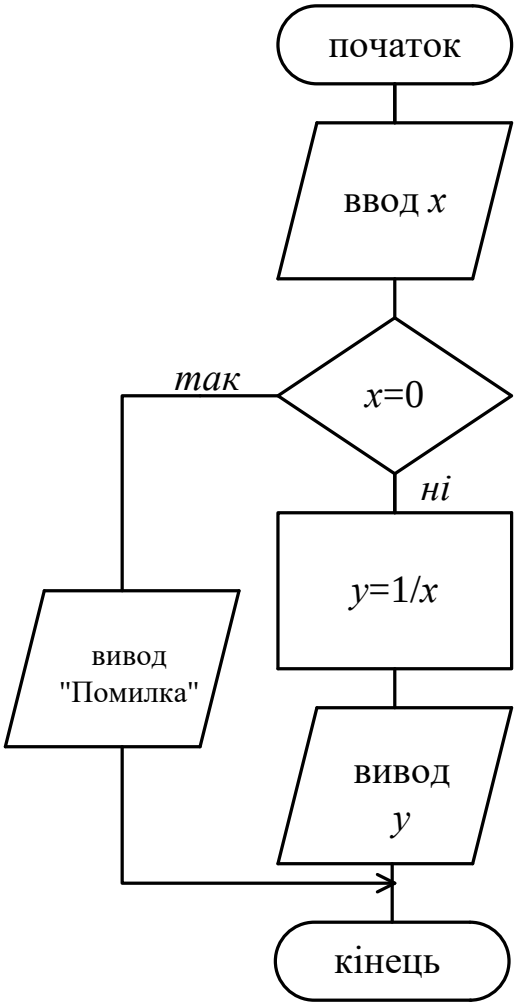


Рисунок 2.8 – Розгалужуються алгоритм

У блоці 1 виконується введення значення x. У блоці 2 введене значення перевіряється. Якщо умова є істинним, то алгоритм переходить до виконання блоку 3, в якому виводиться повідомлення про помилку. Якщо умови блоку 2 є помилковим, то алгоритм переходить до виконання блоку 4, в якому обчислюється значення виразу, заданого в умови. У блоці 5 обчислене значення виводиться користувачеві. Циклічні алгоритми будуть розглянуті пізніше.

2.6. Загальні відомості про реалізацію програмних продуктів

2.6.1. Типи програмних продуктів

Як було вище зазначено, після того як розроблені алгоритми вирішення тієї чи іншої програмної задачі, ці алгоритми повинні бути реалізовані. Реалізація полягає в створенні якогось програмного об'єкта, який в подальшому може використовувати кінцевий споживач. В результаті розробки програмний продукт може бути представлений різними способами. Найпоширенішими варіантами є такі варіанти програмного продукту:

1. Додаток (*application*, програма).

Додаток найчастіше оформляється у вигляді окремого файлу (виконуваного модуля) або групи файлів. Є самостійною програмної одиницею. Зазвичай додатки орієнтовані на кінцевих користувачів і дозволяють вирішувати конкретні прикладні задачі, наприклад, обробку текстів, зображень, роботу з файловою системою, передачу інформації по мережі і т.д. На різних програмних платформах додатки можуть бути представлені файлами різних типів. Так, в операційній системі *Windows* додатки зазвичай представлені двійковими виконуваними файлами, які мають спеціальну внутрішню структуру і розширення **.com, *.exe*. В *UNIX*–подібних операційних системах, наприклад в ОС Linux, додатки зберігаються також в довічних файлах, а ознакою того, що файл є виконуваним, є встановлений біт 'x' в правах доступу до цього файлу.

2. Бібліотека (*library*).

Бібліотека являє собою набір програмних модулів, кожен з яких може виконувати обмежений набір функцій. Крім того, бібліотека містить механізми доступу до збережених в ній програмним функцій. Функція з бібліотеки може бути викликана іншою програмою або іншою бібліотекою (рис. 2.9).

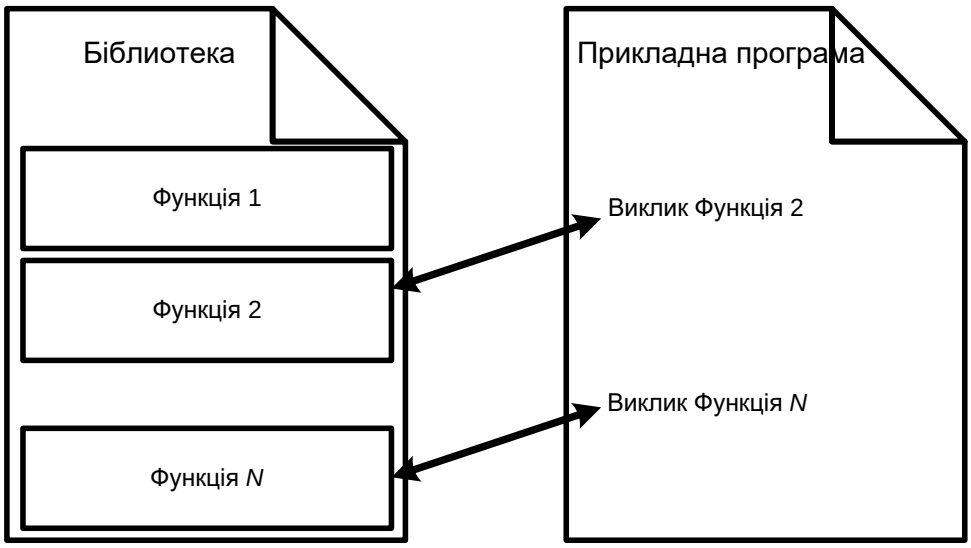


Рисунок 2.9 – Набір програмних модулів у бібліотеці

Такий програмний продукт орієнтований, зазвичай, на розробників програмного забезпечення. Так, якщо при розробці власного програмного продукту розробнику необхідно виконати ті чи інші дії з файлом певного типу. Розробник може написати власний код для реалізації таких дій, а може скористатися вже існуючими бібліотеками, які розроблені сторонніми програмістами. У деяких випадках такий підхід є економічно виправданим. Причому, тому що бібліотека є двійковим файлом, то вихідний програмний код є недоступним для кінцевого користувача, що дозволяє дотримуватися вимог законів про авторські права. Крім того, велика кількість стандартних алгоритмів реалізовано в бібліотеках з відкритим кодом.

Бібліотеки розрізняються за механізмом їх підключення до призначених для користувача програм. Статичні бібліотеки (*Static Library*) підключаються до додатка, де вони будуть використані, на етапі створення. При цьому встановлюються всі необхідні зв'язки між різними програмними модулями, жорстко визначається порядок виклику бібліотечних функцій тощо. В результаті статична бібліотека стає частиною програмного продукту. Динамічні бібліотеки (*Dynamic–Linked Library, DLL*) існують окремо від додатків, в яких вони використовуються. Додаток може підключати необхідні бібліотеки в процесі своєї роботи і відключати їх, коли це потрібно. В операційній системі Windows,

наприклад, в динамічних бібліотеках реалізована робота зі стандартними візуальними елементами управління – іконками, вікнами, кнопками, рядками введення і т.д.

Використання статичних і динамічних бібліотек має свої переваги і недоліки. Оскільки статична бібліотека міститься безпосередньо всередині програми, то цей додаток має більший розмір. Недоліком динамічних бібліотек є те, що для роботи програми необхідна наявність всіх бібліотек, які воно використовує. Якщо хоча б одна динамічна бібліотека відсутня, то програма не може бути запущена.

2.6.2. Вибір засобів реалізації програмного забезпечення

Ще одним важливим кроком реалізації програмного продукту є визначення, яким чином розроблений алгоритм буде перетворений в програмний код. Для вирішення цього завдання використовуються спеціальне програмне забезпечення – мови програмування.

Мова програмування– формальна знакова система, призначена для опису алгоритмів в формі, яка зручна для виконавця – в нашому випадку для комп'ютера. Мова програмування визначає набір правил, використовуваних при складанні комп'ютерної програми. Він дозволяє програмісту точно визначити то, на які події буде реагувати комп'ютер, як будуть зберігатися і передаватися дані, а також які саме дії слід виконувати над цими даними при різних обставинах.

З часу створення перших програмованих машин людство придумало вже більше двох з половиною тисяч мов програмування (повний список містить 8512 мов програмування). Щороку їх кількість збільшується. Деякими мовами вміє користуватися тільки невелике число їх власних розробників, інші використовуються дуже широко.

Творці мов по–різному тлумачать поняття мову програмування. Загальними термінами, які визнаються більшістю розробників, є наступні:

- Функція: мова програмування призначений для написання комп'ютерних програм, які застосовуються для передачі комп'ютеру інструкцій щодо виконання того чи іншого обчислювального процесу і організації управління окремими пристроями.
- Завдання: мова програмування відрізняється від природних мов тим, що призначений для передачі команд і даних від людини комп'ютера, в той час як природні мови використовуються лише для спілкування людей

між собою. В принципі, можна узагальнити визначення «мов програмування» – це спосіб передачі команд, наказів, чіткого керівництва до дії, тоді як людські мови служать також для обміну інформацією.

- Виконання: мова програмування може використовувати спеціальні конструкції для визначення і маніпулювання структурами даних і управління процесом обчислень.

Існують наступні типи мов програмування:

1. Машинна мова (код). Це набір команд конкретного мікропроцесора і іншого обчислювального пристрою, який інтерпретується на апаратному рівні. Команди мови представляють собою числа (коди), які обчислювач розпізнає як операції і виконує їх. Машинний код є апаратно–залежним мовою низького рівня, тобто кожен тип обчислювальної системи має власний набір команд. В даний час машинні коди не використовуються, тому що висока трудомісткість написання і налагодження програмного забезпечення.

2. Мова асемблера. Також є мовою програмування низького рівня. Мова отримала свою назву від слова assembler – складальник. Мова асемблера, в деяких випадках, для стислості, називають «асемблером» (а щось пов'язане з ним – «асемблерний»), але в загальному випадку це неправильно. Команди мови асемблера повністю відповідають командам процесора і фактично, є зручною символьною формою запису машинних кодів. Також, мова асемблера забезпечує скріплення частин програми і даних. Асемблер є апаратно–залежною мовою програмування. Зазвичай програми або частини програм пишуться на мові асемблера в випадках, коли розробнику критично важливо оптимізувати такі параметри, як швидкодія і розмір програми – програмне забезпечення для мікроконтролерів і процесорів з обмеженими ресурсами,

Останнім часом набули поширення так звані «проміжні» асемблери, які є платформо–незалежніми. Наприклад, асемблер, в який компілюються програми на платформі *.Net* (*p*–код), спеціалізовані асемблери, призначені для програмування відео процесорів і т.д.

3. Мови програмування високого рівня. Такі мови програмування на відміну від машинного коду і мови асемблера є машинно-незалежними. Це означає, що одна і та ж програма без змін може бути виконана теоретично на різних апаратних платформах (рис. 2.10).

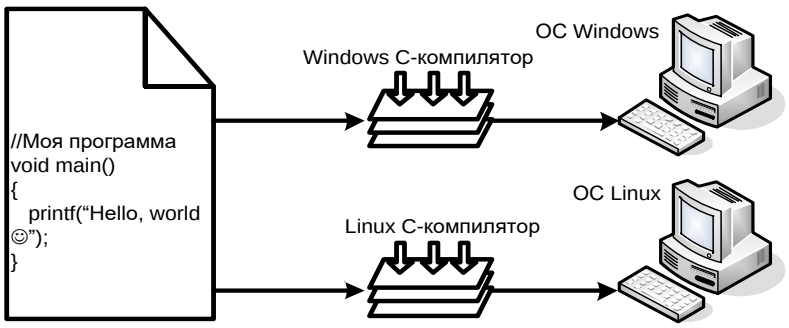


Рисунок 2.10 – Апаратні платформи

Конструкції в мовах високого рівня близькі по написанню до людської мови. Наприклад, один з операторів мови *Pascal* виглядає наступним чином:

if a <5 *then write ('Результат обчислень =', a);*

З такого запису, навіть людині незнайомому з програмуванням, зрозуміло, що оператор означає наступне:

«Якщо значення '*a*' менше 5, то вивести повідомлення 'Результат обчислень =' і значення '*a*'«.

На даний час ЯВУ використовуються дуже широко в різних областях програмування, причому для вирішення різних завдань використовуються різні мови. У досить великих програмних проектах може одночасно використовуватися кілька мов програмування. Серед сучасних мов програмування високого рівня широко використовуються такі як *Pascal, C, C ++, Java, Haskel, Perl, TCL* та ін.

Також зараз широко поширені системи візуального програмування. Такі системи зазвичай реалізовані на базі популярних мов програмування високого рівня і дозволяють частина коду програм генерувати автоматично – наприклад, код пов'язаний з реалізацією інтерфейсу користувача, робота з базами даних і т.д. Серед найбільш використовуваних систем візуального програмування – *Microsoft Visual Studio, Borland Delphi, Borland C ++ Builder*.

2.6.3. Технологія створення програмного продукту

Для отримання двійкових програмних файлів потрібне проведення певних перетворень вихідного коду. Процес перетворення вихідного коду програми в більш низькорівневий код або в машинний код називається трансляція програми. Трансляція програм виконується спеціальним програмним забезпеченням – транслятором. Транслятори діляться на два класи: компілятори (*compiler*) і інтерпретатори (*interpreter*).

Програма за допомогою компілятора перетвориться (компілюється) в набір інструкцій для даного типу процесора (машинний код) і далі записується в виконуваний файл, який може бути запущений для виконання як окрема програма. Іншими словами, компілятор переводить програму з мови високого рівня на низькорівневу мову, зрозумілу процесору (рис. 2.11).

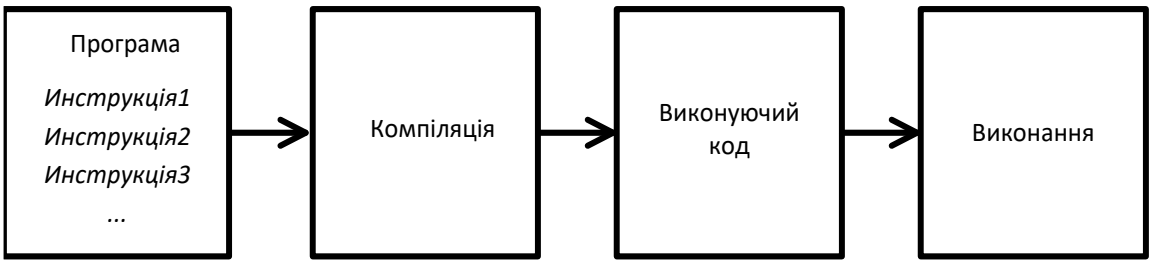


Рисунок 2.11 – Алгоритм роботи компілятору

Інтерпретатор безпосередньо виконує (інтерпретує) її текст без попереднього перекладу. При цьому програма залишається мовою оригіналу і не може бути запущена без інтерпретатора (рис. 2.12).

Рисунок 2.12 – Алгоритм роботи інтерпретатору

Коротко кажучи, компілятор переводить програму на машинну мову відразу і цілком, створюючи при цьому окрему програму, а інтерпретатор перекладає на машинний мову під час виконання програми.

Мови програмування поділяються на компільовані й інтерпретовані мови. До компільовані мов відносяться такі мови як *C, C ++, Pascal* і ін. До інтерпретується – *Java, C#* і ін.

Поділ на компільовані й інтерпретовані мови є дещо умовним. Так, для будь–якого традиційно компільованої мови, як, наприклад, Pascal, можна написати інтерпретатор. Крім того, більшість сучасних «чистих» інтерпретаторів не виконують конструкції мови безпосередньо, а компілюють їх в деяке високорівневе проміжне представлення.

Для будь–якої інтерпретованої мови можна створити компілятор – наприклад, мова *Lisp*, спочатку інтерпретована, може компілюватися без яких би то не було обмежень. Створюваний під час виконання програми код може так само динамічно компілюватися під час виконання.

Як правило, скомпільовані програми виконуються швидше і не вимагають для виконання додаткових програм, так як вже переведені на машинну мову. Разом з тим при кожній зміні тексту програми потрібно її перекомпіляція, що створює труднощі при розробці. Крім того, скомпільована програма може виконуватися тільки на тому ж типі комп'ютерів і, як правило, під тією ж операційною системою, на яку був розрахований компілятор. Щоб створити виконуваний файл для машини іншого типу, потрібна нова компіляція.

Інтерпретовані мови мають деякі специфічні додаткові можливості, крім того, програми на них можна запускати відразу ж після зміни, що полегшує розробку. Програма на інтерпретованій мові може бути часто запущена на різних типах машин і операційних систем без додаткових зусиль. Однак інтерпретовані програми виконуються помітно повільніше, ніж компільовані, крім того, вони не можуть виконуватися без додаткової програми–інтерпретатора.

Деякі мови, наприклад, *Java* і *C#*, знаходяться між компільованими і інтерпретованими. А саме, програма компілюється не в машинний мову, а в машинно–незалежний код низького рівня, байт-код. Для виконання байт-коду зазвичай використовується інтерпретація, хоча окремі його частини для прискорення роботи програми можуть бути трансльовані в машинний

код безпосередньо під час виконання програми за технологією компіляції «на льоту» (*Just–in–time compilation, JIT*).

Подібний підхід у певному сенсі дозволяє використовувати плюси як інтерпретаторів, так і компіляторів. Слід згадати також оригінальну мову Форт (*Forth*), який є одночасно інтерпретованим і компільованим. Процес створення програми на мові компільованого типу можна представити у вигляді такої схеми (рис. 2.13):

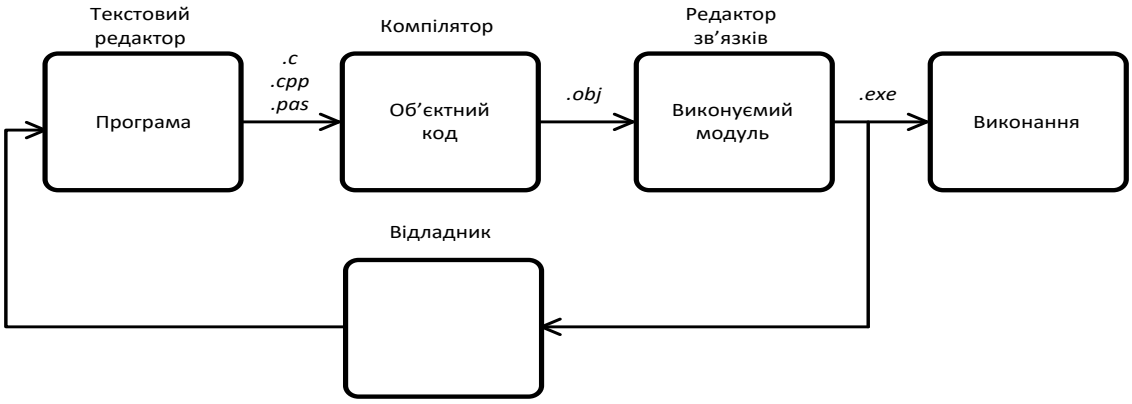


Рисунок 2.13 – Етапи створення виконуваного файлу програми

Вихідний код програми на мові високого рівня створюється в текстовому редакторі і записується в дисковий файл. Такий файл є звичайним текстовим і має розширення в залежності від мови програмування – **.c* (мова *C*), **.cpp* (мова *C ++*), **.pas* (мова *Pascal*). Після того, як текст програми створений, відбувається його компіляція. В результаті компіляції програмний продукт представлений у вигляді набору об'єктних файлів (модулів) з розширенням **.obj*. Об'єктний файл містить машинний код програми у вигляді машинних інструкцій, але не може бути виконаний, і потрібно його об'єднання з іншими об'єктними файлами, бібліотечними файлами, налаштування структур даних і т.д. Після виконання компіляції проводиться зв'язування об'єктних модулів. Ця операція виконується редактором зв'язків (*linker, link editor*).

Програмний продукт може містити помилки, які не були знайдені на етапі компіляції і зв'язування програми. Такі помилки називаються помилками часу виконання. Для виявлення таких помилок виконується процес налагодження програми. У процесі налагодження також може використовуватися спеціальне програмне забезпечення – відладник

(*debugger*). Усунення несправностей дозволяє локалізувати місце розташування помилки. Після знаходження помилок вносяться зміни у вихідний код за допомогою текстового редактора, і далі повторюється повний процес збирання програмного продукту.

Контрольні питання

1. Технологія створення програмного забезпечення. Етапи створення. Програми, необхідні для створення програмного забезпечення.

2. Оператори управління в мові програмування *C/C++*. Призначення. Види операторів управління.

3. Способи подання алгоритму. Призначення. Особливості застосування різних способів запису.

4. Алгоритмізація задач. Програмний принцип функціонування обчислювальних систем.

5. Оболонка *Microsoft Visual Studio*. Призначення. Зовнішній вигляд оболонки.

6. Створення проекту Програми з використанням оболонки.

7. Ітераційні циклічні алгоритми. Призначення, приклади використання.

8. Етапи створення програмного забезпечення. Призначення основних етапів.

9. Мови програмування. Призначення. Типи мов програмування.

10. Тестування, як етап розробки програмного забезпечення. Призначення, особливості тестування.

11. 52. Мова програмування *C/C++*. Основні поняття, лексеми

12. Машинні коди і мови асемблера. Призначення, використання. Гідності й недоліки.

13. Мови програмування високого рівня. Призначення. Гідності й недоліки. Галузь застосування.

14. Оформлення, протоколювання програмного забезпечення. Необхідність, здача програмного забезпечення замовнику.

15. Поняття алгоритму. Призначення. Вимоги до алгоритмів.

РОЗДІЛ 3.

ОСНОВИ МОВИ ПРОГРАМУВАННЯ *C/C ++*. ЛІНІЙНІ ПРОГРАМИ

3.1. Особливості мови *C++* і сфери використання

Мова *C* є мовою програмування загального призначення. Поява його тісно пов'язане зі створенням операційної системи *UNIX*, бо він розроблявся спеціально для цієї системи і на ньому написана велика частина системного програмного забезпечення *UNIX*. Однак мова *C* не орієнтована на роботу тільки в одній операційній системі або на будь–якої конкретної машині. Він являє собою універсальний, машинно–незалежний, легко переносну мову програмування, в рівній мірі підходить як для системного програмування, так і для вирішення завдань обчислювальної математики, технічних і комерційних додатків. Мова *C/C++* – це алгоритмічна мову «не дуже високого рівня». Наведена характеристика означає, що вона дає можливість працювати з такими типами об'єктів і дозволяє виконувати такі операції, які традиційно відносять до машинно–орієнтованих мов «низького рівня», подібним мові асемблера. Прикладом згаданих об'єктів можуть служити символи, числа (зі знаком і без), адреси оперативної пам'яті і покажчики, бітові ланцюги і т.д., над якими можна проробляти арифметичні, логічні і порозрядні операції. У той же час, *C* не забезпечує можливості роботи з рядками, множинами, масивами або списками як з єдиним цілим. У мові *C* є великий набір керуючих конструкцій для реалізації циклічних і розгалужених алгоритмів, засоби для блочного і модульного програмування, а також можливість гнучкого управління процесом виконання програми. Все це дозволяє охарактеризувати *C* як мову структурного програмування. Дії з обміну інформацією з периферійними пристроями виконуються за допомогою явно викликаються зовнішніх функцій, що входять до складу стандартних бібліотек. Поява *C* прийнято пов'язувати з ім'ям Мартіна Рітчі, який підготував в 1972 р. першу версію цієї мови в ході робіт над операційною системою *UNIX* для ЕОМ сімейства *PDP*.

Класичною книгою є Брайан Керниган, Денис Рітчі «Мова програмування *C*» (1978 р.). Однак історично його виникнення слід

пов'язувати з, багато в чому машинно-залежним, мовою *B*, створеним Кеном Томпсоном на основі мови *BCPL*. У 1980–х роках він був адаптований для використання в *IBM PC*, що призвело до різкого зростання його популярності. У 1983 р. Американському Національнрму Інституті Стандартизації (*ANSI*) сформував комітет для розробки стандартної специфікації *C*. Після закінчення цього довгого і складного процесу в 1989 році він був затверджений як «Мова програмування *C*» *ANSI X3*.159–1989. Цю версію мови прийнято називати *ANSI C*. У 1990 році стандарт *ANSI C* був прийнятий з невеликими змінами Міжнародною Організацією по Стандартизації (*ISO*) як *ISO/IEC* 9899: 1990.

У 1980–х роках Бьярне Страуструп в лабораторіях *Bell Labs* розпочали роботу по додаванню в *C* можливостей об'єктно–орієнтованого програмування. Страуструп почав працювати над «*C* з класами» в 1979 р. Мова *C*, будучи базовою мовою системи *UNIX*, на якій працювали комп'ютери «*Bell*» є швидким, багатофункціональним і переносним. Страуструп додав до нього можливість роботи з класами та об'єктами. В результаті, практичні задачі моделювання виявилися доступними для вирішення як з точки зору часу розробки (завдяки використанню класів) так і з точки зору часу обчислень (завдяки швидкодії *C*).

У 1983 р відбулося перейменування мови з *C* з класами в *C* ++ з міркувань маркетингу. У 1985 р. вийшло перше видання «Мови програмування *C*++», що забезпечує перший опис цієї мови, що було надзвичайно важливо через відсутність офіційного стандарту.

C/C++ широко використовується для розробки програмного забезпечення різного призначення:

1. Розроблення прикладного та системного програмного забезпечення для роботи в різних операційних системах.

2. Розроблення переносного програмного забезпечення, тобто програм, які можуть функціонувати під управлінням різних платформ.

3. Розроблення великих програмних проектів, в яких використовується одночасно декілька технологій програмування. Використання мови *C/C*++ дозволяє створити ефективну середу взаємодії між частинами проекту, написаними на різних мовах програмування.

4. Розроблення програмного забезпечення для *UNIX* систем. Компілятори мов входять до складу операційної системи.

5. Розроблення програмного забезпечення для мікроконтролерів. В даний час особливості мови С привели до того, що він став альтернативою мови асемблер для розробки програмного забезпечення для різних мікроконтролерів – *PIC* фірми *Microchip, AVR, ATMega* фірми *ATMEL,* TMS, *MISP* фірми Texas Instrument, *MCS*51 фірми *Intel*.

6. Розвитком мови *C/C++* є мова *C#*, яка використовується для створення машинно–незалежних додатків, які використовуються в мережі Інтернет, мобільних пристроях і т.д.

В даний час існує велика кількість засобів розробки програмного забезпечення на мовах *C* і *C++*. До складу операційних систем UNIX компілятор мови входить як частина операційної системи. Компіляція, зв'язування, налагодження програм виконується утилітами операційної системи – *make, gmake, dbg*. У графічній оболонці *KDE* є спеціальне середовище розробки *Kdevelop*.

Для операційних систем сімейства *Windows* в якості засобів розробки користуються оболонками *Microsoft Visual C++* фірми *Microsoft, C++ Builder* фірми *Inprise*. Для професійного програмування при створенні високооптимізованоо програмного забезпечення використовують такі компілятори мови Intel *C++ Compiler*.

Крім того, існує велика кількість програмних бібліотек, які можна підключати до програм на мовах *C* і *C++*. Такі бібліотеки містять вже готові програмні модулі і розробник може використовувати їх на свій розсуд. Наприклад, існують бібліотеки для роботи зі стандартними елементами управління Windows – вікнами, кнопками і т.д. Для оболонки *Microsoft Visual C ++* фірма Microsoft розробила бібліотеку з набором стандартних класів – *MF*C (*Microsoft Foundation Classes*). Серед часто використовуваних бібліотек слід згадати *STL* (*Standard Template Library*) – бібліотеку маніпуляції даними (сортування, пошук, робота зі структурами даних), до того ж з 1998 року бібліотека входить в стандарт мови *C++*, а також бібліотеку *Qt*, яка містить функції маніпуляції даними, процесами, елементами управління і т.д.

3.2. Основні поняття мови *C/C++*

Програма на мові програмування *C/C++* являє собою текстовий документ, оформлений відповідно до правил мови. Основною лексичною

одиницею мови є лексема. В *C/C++* використовуються наступні види лексем:

1. ***Ідентифікатор***. Ідентифікатор є послідовністю алфавітно–цифрових знаків для позначення якогось об'єкта в програмі. Символ підкреслення «_» розглядається компілятором як буква. Великі і малі літери латинського алфавіту вважаються різними. Довжина ідентифікатора формально може бути довільною, проте тільки перші символи ідентифікатора є значущими для компілятора. Кількість значущих символів залежить від компілятора. У програмах на мові *C/C++* ідентифікатори служать іменами змінних, символічних констант, функцій, типів даних і міток. Загальноприйнято для ідентифікаторів констант використовувати великі літери латинського алфавіту, малі літери використовують зазвичай для позначення змінних і функцій. Далі наведено приклади ідентифікаторів: *my_var1*, *number*, *Number* (є відмінним від number, але не рекомендується в програмі використовувати два або більше ідентифікаторів, які відрізняються тільки регістром символів), func_max, *SQUARE*.

2. ***Ключове слово***. Ключове слово – це зумовлений ідентифікатор, який має спеціальне значення, визначене синтаксисом мови. Імена ніяких об'єктів програми не повинні збігатися з ключовими словами. Далі наведено приклади ключових слів: *int, while, return*.

3. ***Константа.*** Ця лексема в програмі використовується для вказівки значень різних типів. Значення ніякої константи не може бути змінено в процесі роботи програми. Спеціальні директиви мови дають можливість привласнювати константам будь-якого типу символічні імена.

У мові *C/C++* використовують такі типи констант:

Цілі константи. Дозволяють задавати цілі числові значення різної довжини зі знаком (позитивні і негативні) і без знака (тільки позитивні), а також в різних системах числення. Цілі константи можуть бути представлені в трьох основних форматах: десятковому, вісімковому і шістнадцятковому. Цілі константи записуються цифрами, які можна використовувати в обраному форматі, перед записом знаковою негативною константи в десятковому форматі вказується знак ‘–‘. Крім того, після запису константи можуть зазначатися модифікаторами, що вказують на довжину і знаковість числа.

Для завдання константи в десятковому форматі використовуються десяткові цифри від “0” до “9”: 12, –984, 29659,0.

Для завдання константи в вісімковій системі числення використовуються десяткові цифри від “0” до “7” і префікс “0”: 0735, 07777.

Для завдання константи в шістнадцятковій системі числення використовуються десяткові цифри від “0” до “9”, букви “*A”*, “*B”*, “*C”*, “*D”*, “*E”*, “*F”*(або маленькі “*a”*, “*b”*, “*c”*, “*d”*, “*e”*, “*f”*) і префікс “0*x”*:

0*x*739, 0*xFFFF*.

Модифікатор “*L”* (“*l”*) вказує, що дана ціла константа має тип *long*. Це означає, що внутрішнє уявлення даної константи в разі необхідності буде розширено незначущими нулями в старших розрядах до заданого розміру. Без вказівки даного модифікатора константа є константою типу *int*. Наприклад: 327*L*, 0*xFFL* (така константа буде перетворена в 0*x*000000*FF*).

Модифікатор “*U”* (“*u”*) вказує, що дана ціла константа має беззнаковий тип. У шістнадцятирічному і вісімковому форматах константи з одиницями в старших розрядах внутрішнього уявлення числа можуть інтерпретуватися як великі позитивні числа, і як негативні числа, представлені в спеціальному форматі – додатковому коді. Даний модифікатор вказує, що константа позитивне число. Якщо модифікатор відсутній, то константа інтерпретується як знакове число. Наприклад: 0*xFFFFFFFFUL* – така константа означає ціле довге беззнаковое позитивне число 4294967295.

Дійсні константи. Служать для завдання чисел з плаваючою комою. Дозволяють задавати знакові речові значення різної точності в форматах з фіксованою і плаваючою крапкою.

Для завдання дійсної константи в форматі з фіксованою точкою вказуються ціла і дробова частини числа, розділені символом ‘.’:

Наприклад: 87.235, –0.373567, 4921., 3.1415926.

Після запису константи в форматі з фіксованою точкою може вказуватися модифікатор ‘*f*’, який вказує приналежність константи до типу *float* (одинарну точність). Точність в загальному випадку задає кількість значущих знаків в числі. За замовчуванням, без модифікатора ' *f* ', константи задаються з подвійною точністю, з зазначенням модифікатора – з одинарної точністю: 12.6*f*.

Формат завдання числа з плаваючою точкою в науковій формі більш складний. В основі цього формату лежить нормалізована форма подання речового значення *A* у вигляді:

$$A = m*10^n,$$

де *m* – мантиса числа, задана в діапазоні 0 <*m* <1,

n – порядок числа.

Матеріальна константа задається в вигляді *mEn* або *men*, мантиса і порядок задаються знакового значення: 1.575*E*1, 1575*e*–2, .1575*E*2, –.175*e*2.

Символьні константи. Символьна константа є буква, цифра, знак пунктуації або керуюча *Esc*–послідовність, укладені в одиночні лапки: ‘символ’. Символ являє собою будь-який друкований символ *ASCII*, крім одиночної лапки або зворотної косої риски ‘\’.

Керуючі *Esc*-послідовності складаються з зворотної косої риски, за якою слідує буква або комбінація цифр. Символи ‘'’ та ‘\’ також представляються у вигляді *Esc*-послідовностей.

Можливі такі *Esc*–послідовності і їх значення:

\ *a* – звуковий сигнал,

\ *t* – горизонтальна табуляція,

\ *b* – крок назад,

\ v – вертикальна табуляція,

\ f – нова сторінка,

\ '– одинарні лапки,

\ *n* – новий рядок,

\ «– лапки,

\ *r* – повернення каретки,

\\ – зворотний слеш (коса риска),

\ *ddd* – символ ASCII в вісімковому поданні,

\ *xddd* – символ ASCII в шістнадцятковому представленні.

У всіх інших випадках символ ‘\’ ігнорується компілятором. Послідовності виду '\ *ddd*' і '\ *xddd*' дозволяють уявити будь-який символ *ASCII*, включаючи і недруковані символи, в вісімковому і шістнадцятковому форматі відповідно. Незначущі нулі зліва в обох випадках можуть бути опущені. Значенням символьної константи є ціле число, яке відповідає внутрішньому машинному уявленню відповідного

символу в таблиці *ASCII*. Нижче наведені приклади правильного запису символьних констант: ‘*A*’, ‘*c*’, ‘?’, ‘7’, ‘\’ ‘,’ ‘\\’, ‘*n*’, ‘\033’, ‘\10’, ‘\ *x1F*’.

Строкові константи – це собою послідовності символів *ASCII*, укладені в подвійні лапки. Недруковані символи *ASCII*, а також символи ‘"’ і ‘\’ в складі рядків повинні представлятися відповідними *Esc*-послідовностями. Компілятор заносить символи рядка в пам'ять ЕОМ в порядку їх слідування, автоматично додаючи нуль-символ (*Esc*-послідовність ‘\0’) в кінець рядка. Формально допустимий є також порожній рядок “ ”, що складається у внутрішньому поданні з одного символу ‘\0’. Нульовий символ як ознака кінця рядка використовується, наприклад, для виконання функцій обробки рядків. Приклад запису символьного рядка: ‘Це рядок символів”.

Для формування довгої послідовності символів, що займає більше одного рядка на екрані терміналу, необхідно набрати символ ‘\’, і продовжити введення символів з нового рядка, наприклад:

*«Довга рядок може бути розбита *
на 2 частини»

Важливо зауважити, що символьна константа, наприклад ‘*x*’, ідентична рядку “x”, що містить один символ. Справа в тому, що символьна константа займає один байт пам'яті і відповідає своєму числовому коду у внутрішньому машинному поданні. Рядок ж з одного символу займає два байти пам'яті, перший з яких містить код цього символу і символ ‘\0’.

3.3. Структура *C/C++* проекту. Директиви компілятору

Програмний проект, який реалізується на мові C/C++ має на меті створення програми або бібліотеки. Програмний проект містить один або кілька програмних файлів, тобто файлів з вихідним кодом на мові програмування. Крім цього, проект містить опис зв'язків між програмними файлами. Основні типи файлів, які входять до складу проекту наступні:

* *.c* – файли з вихідним кодом на мові *C*;
* *.cpp* – файли з вихідним кодом на мові *C ++*;
* *.h* – заголовки (*headers*).

Часто проект, якщо він досить великий, можна розбити на функціонально відокремлені частини – програмні модулі. Такий підхід дозволяє більш ефективно організувати процес розробки, тому що всі частини можуть паралельно розроблятися кількома програмістами. Кожна така частина зазвичай представлена в проекті двома файлами з однаковими іменами: файл з розширенням *.*h* містить описову інтерфейсну частину, файл з розширенням *.*c* або *.*cpp* містить власне функціональну реалізацію. Наприклад, програмний модуль проекту, який використовується для організації мережевих функцій програми, може міститися в файлах *net_conn.cpp і net_conn.h*.

Що означає інтерфейсна частина такого модуля, яка описана в файлі *net_conn.h*? В технології програмування широко використовується таке поняття як інтерфейс. Програмний інтерфейс – це механізм, який дозволяє надавати ресурси одного програмного модуля іншим програмним модулям (рис. 3.1).

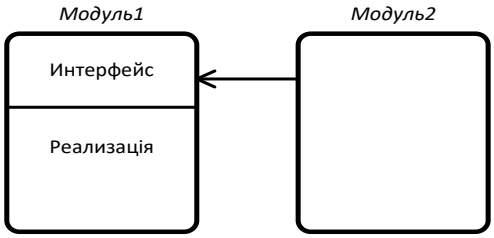


Рисунок 3.1 – Програмний інтерфейс

Модуль 2 використовує для своєї роботи певні частини з Модуль 1. При цьому немає прямого доступу до більшості функцій, реалізованих в модуль1 з Модуль 2. Модуль 2 запитує у інтерфейсної частини модуль1 необхідні дії. Якщо такі дії можуть бути виконані, то інтерфейс викликає необхідні функції і відправляє отриманий результат з Модуль 2.

Програмний інтерфейс має ряд переваг. Цей механізм дозволяє організувати ефективну взаємодію, як між окремими додатками, так і програмними модулями в одній програмі. При використанні інтерфейсної взаємодії зменшується кількість зв'язків між модулями. Ще однією перевагою є підвищення безпеки при виконанні програмного забезпечення (рис. 3.2).

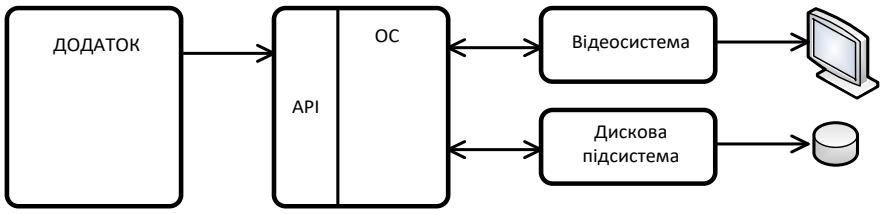


Рисунок 3.2 – Програмний інтерфейс з інтерфейсними взаємодіями

Операційна система керує апаратним забезпеченням обчислювальної системи. Якщо який-небудь додаток хоче записати інформацію на диск ЕОМ, воно використовує інтерфейс операційної системи, викликає необхідні для запису на диск функції, а вже операційна система визначає, як записувати зазначену інформацію, яку частину дискового простору для цього використовувати і т.д. Додаток може мати власні можливості для доступу до диска. Але при цьому виникає небезпека некоректної роботи з диском – наприклад, дані будуть записані в область дискового простору, де вже є записана інформація і вона буде загублена, або якщо кілька додатків одночасно будуть намагатися записати свої дані на диск і необхідно здійснювати коректне управління, щоб дані були розміщені правильно. Також виконується вивод інформації на монітор – через відповідний інтерфейс операційної системи. Операційна система *Windows* має такий програмний інтерфейс – *Application Program Interface* (*API*), функції якого використовують всі програми, що працюють під її управлінням.

Повернемося до згаданого вище файлу *net_conn.h*. У цьому файлі розміщується інтерфейсна частина модуля, тобто всі описи, необхідні для того, щоб цей модуль міг бути підключений до інших модулів проекту. Підключення здійснюється способом, який показаний на рис. 3.3

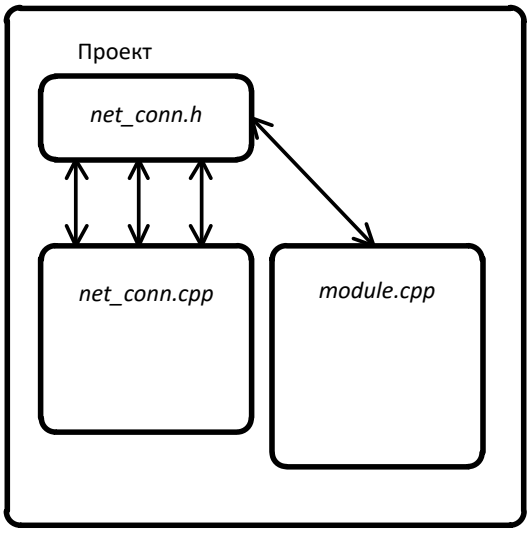


Рисунок 3.3 – Підключення файлу *net_conn.h*

Програмний модуль *net_conn* реалізований в файлі *net_conn.cpp* і його інтерфейсна частина описана в *net_conn.h*. Спеціальними директивами файл *net_conn.h* включається в файл *net_conn.cpp*. Якщо потрібно викликати функції модуля *net_conn* з будь–якого іншого модуля, наприклад, реалізованого в файлі *module.cpp*, в цьому випадку, використовуючи тіж директиви, розробник включає *net_conn.h* в файл *module.cpp*. Таким же чином можна підключити модуль *net_conn* і до інших модулів проекту.

Будь-яка програма на мові *C* являє собою сукупність функцій, які виконують основну роботу по реалізації деякого алгоритму. Кожна з цих функцій, в свою чергу, є незалежний набір описів і операторів, укладених між заголовком функції і її кінцем. Та функція, з якої починається виконання програми, називається головною функцією. Вона повинна мати визначене ім'я *main* ().

Програма може містити рядки, які дозволяють управляти процесом компіляції – директиви компілятора. Також використовується назва директива препроцесора. Така назву означає, що всі зазначені директиви обробляються перед основним процесом компіляції програми. Такі рядки можуть розташовуватися в будь-яких частинах програмного коду. Вони не є командами мови програмування. Основне призначення директив – в процесі компіляції коду програми задавати компілятору інструкції як компіляція повинна відбуватися.

Синтаксис включення директив в програму такий: # директива

Директива include служить для включення заголовків в інший файл. Може використовуватися як в заголовних файлах, так і в файлах реалізації. Зазвичай перелік включення файл розташовується на початку файлу.

Синтаксис директиви наступний:

#include <filename.h>– для включення стандартних заголовків файлів,

#include «filename.h»– для включення власних заголовків файлів,

Директива *include* використовується для підключення стандартних бібліотек до програми. Ось деякі заголовки стандартних бібліотек і їх призначення:

#include <stdio.h>– бібліотека для роботи з функціями стандартного вводу-виводу *C,*

#include <iostream>– бібліотека для роботи з функціями стандартного вводу-виводу *C ++,*

#include <math.h>– бібліотека для роботи з математичними функціями,

#include <string.h>– бібліотека для роботи із строковими даними,

#include <time.h>– бібліотека для роботи з функціями обробки часу.

Директива include використовується також для реалізації описаного вище механізму програмного інтерфейсу. Робиться це в такий спосіб (рис. 3.4).

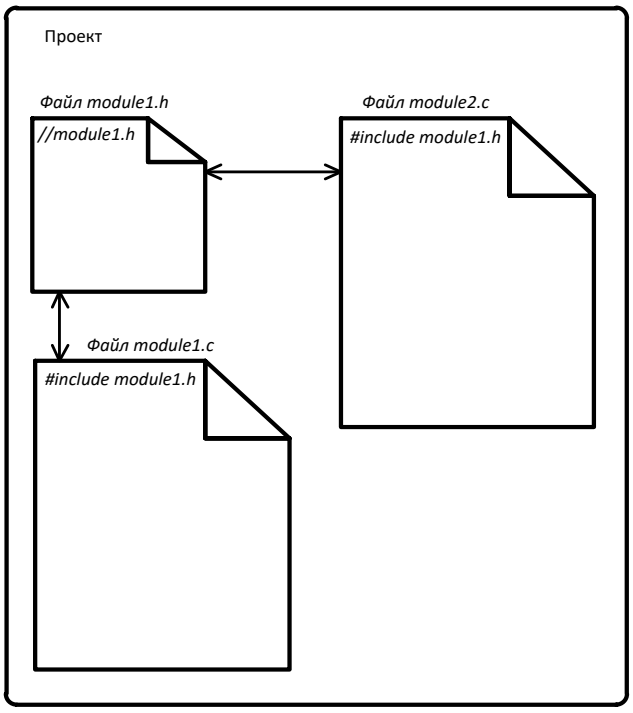


Рисунок 3.4 – Підключення директив

У файлі *module1.h* описані функції та інші елементи програми. У файлі module1.c виконана реалізація цих елементів. За допомогою директиви *#include «module1.h»* заголовки підключається до файлу реалізації. Якщо ми хочемо використовувати функції, реалізовані в *module1.c*, в якомусь іншому файлі, наприклад, *module2.c*, необхідно в нього включити заголовки module1.h, як файл інтерфейсу.

Це також виконується директивою *#include «module1.h»*.

Директива define дозволяє визначити символьний ідентифікатор, а також визначити для символьного ідентифікатора значення або вираз. У програмі можуть використовуватися такі варіанти директиви:

#define ідентифікатор – дозволяє задати символьний ідентифікатор.

#define ідентифікатор значення – дозволяє задати символьному ідентифікатором значення.

#define ідентифікатор (параметри) вираз – дозволяє замінити вираз символьним ідентифікатором.

Параметр ідентифікатор прийнято ставити великими літерами і цифрами: *MAX_VAL1, WIDTH, COUNT* і т.д. Наприклад, в програмі задана директива компілятора

#define PI 3.1415926

Це означає, що в програмі визначено символьний ідентифікатор *PI*, якому у відповідність встановлено дійсне значення 3.1415926. По суті, визначена матеріальна константа *PI* зі значенням 3.1415926. Тепер в програмі у всіх місцях, де є таке значення можна використовувати константу *PI*. При компіляції програми програма–компілятор автоматично замінить всі ідентифікатори *PI* на значення 3.1415926.

Як використовувати такі директиви в програмі?

Наприклад, є програма, яка виконує обробку зображення, ширина якого 800 пікселів. У різних частинах програми виконуються деякі повторювані дії з елементами зображення (пікселями). Потім з'явилася необхідність реалізувати таку ж обробку із зображенням, ширина якого становить 1024 пікселів. Для цього необхідно у всіх частинах програми, де значення 800 використовується як ширина зображення, замінити на значення 1024. Зручно ввести директиву

#define IMG_WIDTH 800

і використовувати в програмі саме цей ідентифікатор. При зміні програми необхідно замінити значення ширини зображення тільки в самій директиві

#define IMG_WIDTH 1024

Так, використання директиви *#define* дозволить значно зменшити обсяг роботи зі зміни програми і виключить можливість виникнення помилок.

Описані раніше ідентифікатори можна використовувати в подальших директивах. наприклад:

#define PI 3.1416926

*#define m_CIRCLE_LEN (d) PI * (d)*

В останній директиві виконано макровизначення (або макрос) для визначення довжини кола з діаметром *d*. У програмі таке макровизначення можна використовувати з параметром, наприклад: *m_CIRCLE_LEN* (5). При компіляції програма–компілятор автоматично замінить такий макрос на вираз 3.1416926*5. Причому зауважимо, що константа *P*I була задана в інший директиві. Прийнято ідентифікатори макровизначень починати з m_. Це дозволяє при читанні тексту програми відразу зрозуміти, що ховається під цим ідентифікатором: числове значення або вираз.

Макровизначення може містити кілька параметрів. Наприклад, макровизначень, яке обчислює квадрат гіпотенузи прямокутного трикутника, може бути реалізовано наступним чином:

*#define m_SHYPOTENUSE (x, y) (x * x) + (y * y)*

Ще один варіант директиви *#define* дозволяє задавати символьний ідентифікатор як такої:

#define LINUX

Таке визначення може, наприклад, бути використано для вказівки, що даний програмний код написаний для роботи в операційній системі *Linux*.

Розглянемо приклад, в якому використовується т.зв. умовна компіляція програми. Умовна компіляція дозволяє виконувати компіляцію не всієї програми, тільки тих рядків, які необхідні на даному етапі розробки програми. За допомогою директиви

#define DEBUG

визначимо ідентифікатор, який вказує, що дана компіляція виконується з налагоджувальними цілями. Це означає, що в результаті програма повинна містити код, який дозволяє виводити якусь додаткову інформацію про хід виконання програми. Така інформація дозволяє відстежувати процес виконання, знаходити помилки і т.д. При закінченні розробки і передачі програми користувачеві налагоджувальна інформація стає непотрібною і необхідно виконати «чистову» компіляцію, т.зв. реліз програми. Зазвичай без налагоджувальної інформації програми має менший розмір і швидше працює. Скасування опису виконується такий директивою:

#undef DEBUG

Використовується *DEBUG* ідентифікатор наступним чином. У програмі включаються такі рядки:

#ifdef DEBUG

оператори виведення налагоджувальної інформації

#endif

Це означать, що оператори між директивами *#ifdef DEBUG і #endif* будуть компілюватися, тільки якщо визначено ідентифікатор *DEBUG*.

Також є можливість виконати умовну компіляцію, коли заданий ідентифікатор не визначений. Наприклад, для нашого випадку необхідно включити в кінцевий код програми оператори, які не потрібні в отладочном варіанті. Тоді програма буде містити наступну конструкцію:

#ifndef DEBUG
оператори релізу програми
#endif

3.4. Описи в програмі змінних

Описи в мові *C/C++* це рядки програми, які не виконують дій відповідно до реалізованим алгоритмом. У цих рядках визначаються характеристики елементів програми, які можуть бути використані в подальшому.

Програма може містити такі види описів:

- опис функції;
- опис змінної;
- опис типу.

Будь-яка програма, призначена для реалізації на ЕОМ, є формалізований опис алгоритму вирішення тієї чи іншої задачі. Дії, що виконуються програмою відповідно до цього алгоритму, спрямовані на перетворення деяких об'єктів, що визначають поточний стан процесу рішення задачі. Такі внутрішні об'єкти програми прийнято називати даними. Ті елементи даних, які зберігають незмінні значення протягом усього часу роботи програми, прийнято називати константами. Інші ж об'єкти, які є предметом зміни в ході виконання алгоритму, називають змінними. З точки зору мови С, яка змінна ототожнюється з її ідентифікатором (ім'ям). З позиції ЕОМ, змінна розглядається як змінюється в часі вміст деякої області оперативної пам'яті – осередки ОЗУ. У процесі виконання програми можуть виконуватися інструкції читання даних із зазначеної осередку пам'яті – таким чином можна отримати значення змінної. Також можна змінити значення змінної – для цього необхідно записати в комірку пам'яті даної змінної записати нове значення. Осередок пам'яті має свою адресу (номер) – адреса змінної. Такий адресний механізм використання змінних висуває ряд вимог до них. Наприклад, спочатку треба встановити значення змінної, і потім використовувати цю змінну. Так, якщо в програмі визначена змінна, то спочатку має бути встановлено її значення, тобто виконана запис значення за адресою даної змінної, а тільки після цього можна використовувати

змінну в операторах, де необхідно отримати значення змінної, тобто виконувати інструкції читання з комірки ОЗУ. Це пов'язано з тим,

Опис змінної необхідно для коректного виділення комірки ОЗУ, в якій буде зберігатися значення змінної. Опис виконується наступним чином:

тип ідентифікатор;
тип ідентифікатор = ініціатор;
тип ідентифікатор1, ідентіфікатор2 = ініціатор, ідентіфікатор3;

Параметр «*тип*» є описувачем типу змінної і визначає наступні характеристики описуваної змінної:

1. Розмір комірки пам'яті, який необхідно виділити для зберігання змінної.
2. Перелік допустимих значень змінної.
3. Формат внутрішнього подання значення змінної.
4. Перелік операцій, які можна виконувати зі змінною.

Змінна може мати стандартний тип або призначений для користувача тип, який описаний безпосередньо в програмі.

Параметр «*ідентифікатор*» це власне ім'ям змінної. Параметр ініціатор дозволяє задати початкове значення змінної при її описі. Кілька змінних одного типу можна задавати в одному описі.

Опис константи виконується, так само як і змінної, але обов'язково потрібно ставити модифікатор *const*:

const тип ідентифікатор = ініціатор*;*

Такий опис вказує, що буде створена константа, значення якої в програмі не може бути змінено. Слід звернути увагу, що при такому описі обов'язково повинен мати місце ініціатор, тому що це єдиний спосіб поставити значення константи.

У мові використовуються наступні стандартні типи даних:

1. Символьний тип даних.
2. Логічний тип даних.
3. Цілі типи даних різної довжини зі знаком і без знака.
4. Речові типи даних різної довжини і точності.

Символьний, логічний і цілі типи даних відносяться до перелічуваних типів.

Змінні символьного типу описуються ключовим словом *char* і можуть приймати значення символів, відповідно до вказаної кодовою таблицею – *character set*. Як відомо, обчислювальна система може обробляти тільки числову інформацію. У кодової таблиці кожному символу, який може бути представлений, задано відповідне число (код).

Таким чином, змінна символьного типу містить числове значення, а програма, яка працює з такою змінною, інтерпретує це число як відповідний символ відповідно до кодової таблиці. Однією з поширених таблиць є таблиця *ASCII* (*American Standard Code for Information Interchange*).

Це кодовий набір, що складається з 256 знаків. У набір символів *ASCII* входять великі та малі літери латинського та російського алфавітів, а також керуючі символи, знаки пунктуації та цифри. Наприклад, символу ‘*A*’ (велика латинська ‘*A*’) відповідає числове значення 65 (0x40), символу ‘Б’ (велике російське ‘Б’) відповідає числове значення 129 (0x81), символ пробіл кодується числом 32 (0x20). Для зберігання змінної символьного типу виділяється 1 байт пам'яті. Приклади опису символьних змінних:

```
char symbol;
char ans = 'Y', letter;
```

Логічний тип даних є тільки в *C++* і використовується для обчислення логічних виразів, зі змінною такого типу виконуються операції алгебри логіки – *AND, OR, XOR,* NOT. Змінна логічного типу може приймати одне з двох значень – true або false. Для зберігання значення змінної виділяється 1 байт пам'яті. Числове значення, яке зберігається в осередку, інтерпретується наступним чином: якщо числове значення дорівнює 0, то логічна змінна має значення false, якщо будь-яке відмінне від 0 числове значення, то логічна змінна має значення *true*. Приклади опису логічних змінних:

```
bool result, flag = true;
```

Цілий тип даних призначено для представлення числових значень без дробової частини. Можуть бути різними і відрізняться розміром і наявністю або відсутністю знака. Розмір змінної цілого типу визначає діапазон можливих значень такої змінної. Розглянемо, наприклад, цілий тип даних, який займає в пам'яті 2 байти (16 біт). Кожен біт може приймати значення 0 або 1. Кількість всіх можливих комбінацій бітів дорівнюватиме 216 = 65636, тобто ціла змінна цілого типу розміром 2 байти може мати 65536 значень. Тобто за допомогою такої змінної можна, наприклад, оперувати цілими числами в діапазоні 0..65535. Для цілої змінної розміром 1 байт діапазон представлених значень буде 0..255 (28–1). Таким чином, розмір цілого типу впливає на діапазон значень змінної. Таке уявлення є беззнаковим.

В описаних вище випадках ми можемо працювати тільки з позитивними числами. Для уявлення і позитивних і негативних значень за допомогою змінної розміром 2 байти можна використовувати ті ж 65536 значень. Кількість позитивних і негативних значень зазвичай має бути однаковим. Для цього той же діапазон можливих 65536 значень розміщуємо так, щоб цей діапазон містився в рівній мірі в позитивної та негативної областях. Тоді діапазон значень цілого числа зі знаком розміром 2 байти буде –32768..32767.

Базовий цілий тип у мові *C ++ – in*t. Тип є знаковим. Особливістю всіх цілих типів є те, що їх розмір залежить від платформи, на якій виконується компіляція програми. Для 32–розрядних операційних систем, наприклад, для поточних версій операційної системи *Window*s, змінна типу *int* займає в пам'яті 4 байта.

Також використовуються цілі типи *short* і *long*. Для вказівки, що цілий тип є беззнаковим, використовується спеціальний модифікатор *unsigned*. У табл. 3.1 наведені характеристики стандартних цілих типів мови *C++*.

Таблиця 3.1 – Стандартні цілі типи мови *C++*

Тип	Розмір	Діапазон	Знаковий
int	4 байта (для 32–розрядних ОС)	–2147483648..2147483647 ($–2^{31}..2^{31}–1$)	+
short	2 байта	–32768 до 32767 ($–2^{15}..2^{15}–1$)	+

long	4 байта	–2147483648..2147483647 ($-2^{31}..2^{31}-1$)	+
unsigned int	4 байта (для 32–розрядних ОС)	0 до 4294967297 ($0..2^{32}-1$)	–
unsigned short	2 байта	0 до 65535 ($0..2^{15}-1$)	–
unsigned long	4 байта	0 до 4294967297 ($0..2^{32}-1$)	–

Приклади опису змінних цілих типів:

int i, j = 10, k;
unsigned long max = 0xFFFFFFFFUL;

Дійсне тип даних використовуються для подання позитивних і негативних числових значень з дробової частиною. Можуть бути різного розміру, тобто змінні дійсних типів можуть займати в ОЗУ різну кількість пам'яті. Розмір дійсної змінної впливає на діапазон представлених значень, а також на точність речового значення. Внутрішній формат уявлення дійсної змінної має вигляд (рис. 3.5), де *m* (0 <m <1) – мантиса числа, *n* – порядок числа, а саме значення обчислюється змінної як $mxx*10^n$.

Рисунок 3.5 – Формат представлення дійсної змінної

З рисунка видно, що відведена під зберігання зміної область пам'яті розділена на 2 частини. Деяка кількість біт відводиться на подання мантиси, що залишилися біти служать для подання порядку числа.

Якщо розглядати дві речові змінні різного розміру, то очевидно, що у змінної більшого розміру для мантиси і порядку виділяється більша кількість біт. Порядок числа, по суті, відповідає за діапазон значень дійсної змінної, тому що збільшення значень самого порядку веде до збільшення числа в цілому. Молодші розряди мантиси, по суті, відповідають за дробову частину значення. Очевидно, що збільшення бітів для

представлення мантиси веде до збільшення кількості розрядів для представлення дробової частини, що, в кінцевому підсумку, веде до збільшення точності представлення самого значення.

Особливістю дійсних значень є те, що немає уявлення для значення 0. Число може набувати значень дуже близькі до 0, але не нульове значення.

Речові змінні з одинарної точністю описуються типом *float*. Розмір змінної такого типу 4 байта, діапазон можливих значень

$$1{,}0*10^{-38} <| x | <1{,}0*10^{38}$$

Речові змінні з подвійною точністю описуються типом double. Розмір змінної такого типу 8 байта, діапазон можливих значень

$$1{,}0*10^{-308} <| x | <1{,}0*10^{308}$$

Приклади опису змінних дійсних типів:

float y, eps =.0003*f*;
double g = 9.8;

Будь-яка програма на мові *C/C++* – це сукупність функцій, які виконують основну роботу по реалізації деякого алгоритму. Кожна з них, у свою чергу, є незалежний набір описів і операторів, укладених між заголовком функції і її кінцем. У складі загальної програми будь–яка функція ідентифікується своїм власним унікальним ім'ям. Та функція, з якої починається виконання програми, називається головною функцією і повинна мати визначене ім'я *main* (). Всі інші функції, що входять в програму, запускаються в роботу шляхом їх прямого або опосередкованого (через інші функції) виклику з головної функції. Функції відіграють надзвичайно важливу роль при розробці програми. Використовуючи функції, вихідну задачу можна представити у вигляді послідовності простіших завдань, кожна з яких реалізує певну частину загального алгоритму. Такий підхід до розробки програмного продукту іноді називають методом декомпозиції або процедурною парадигмою програмування. Оперуючи функціями, що складаються з порівняно невеликого числа операторів, значно легше задовольнити вимогам

програмування і зменшити час на налагодження програмного забезпечення. Також, у вигляді функцій можуть бути реалізовані окремі часто використовувані алгоритми. Включення таких функцій до складу стандартних бібліотек дає принципову можливість не реалізовувати самостійно щоразу найбільш використовувані методи, алгоритми та операції.

Зазвичай функція має наступну структуру:

тип ім'я (параметри)
{
    *опис*1;
    *опис*2;
    ...
    *оператор*1;
    *оператор*2;
    ...
}

Як бачимо, функція повинна мати тип. При виклику функція може бути частиною вираження, і після завершення функція може повертати деяке значення. Тип функції вказує, якого типу буде повертається значень і може бути як стандартний, так і описаний додатково. Для функцій, які не повертають значень (процедур), є спеціальний тип – *void*. Виклик функції здійснюється за її імені і заданим в заголовку параметрам. У тілі функції оператори поділяються символом ‘;’ і виконуватися послідовно: оператор1, оператор2, і т.д. При виконанні спеціальних операторів передачі управління порядок виконання може бути змінений. Функція закінчує своє виконання після завершення свого останнього оператора або при виконанні оператора повернення, причому якщо у функції є тип, то потрібно обов'язково повернути результат.

Крім того існує ряд необов'язкових, але бажаних вимог до функцій. Так назва функції повинно відображати її зміст. Тобто якщо ви пишете функцію знаходження максимального елемента деякої послідовності, назвати її можна *max* (...), *max_element* (...), *MaxElement* (...) і т.д. і незручно якщо функція буде називатися просто a(). Крім того функція повинна виконувати одну зрозумілу задачу і не мати побічних ефектів.

Коментарі. Програма може містити рядки, які будуть ігноруватися при компіляції програми – коментарі. У мовах *C* і *C++* коментар оформляється наступним чином:

```
/ *
        ...
        довільний текст
        ...
* /
```

Або / * довільний текст * /

Символи "/*" відкривають текст коментаря, символи "*/" закривають текст коментаря. Весь текст, розташований між ними ігнорується при компіляції. Коментарі не може бути вкладеними.

В *C ++* є можливість використовувати однорядковий коментар:

```
// довільний текст
```

В даному випадку ігнорується текст від символів "//" до кінця рядка.

Коментарі можуть використовуватися для внесення пояснювальних написів в текст програми: призначення тих чи інших змінних, функцій, частин програми і т.д. Такий підхід є корисним при супроводі програмного забезпечення, тому що дозволяє швидше зрозуміти, що виконується в тій чи іншій частині програми, навіщо введена змінна і т.д.

Також коментарі використовуються в процесі налагодження програмного забезпечення. Коментуючи частина вже написаної програми, можна виключити її виконання, що допоможе визначити, наприклад, місце виникнення деякої помилки.

Приклад пояснюючих коментарів:

```
/ **********************************
* Лабораторна робота 1
* Автор:
* Іванов П.П.
********************************** /
void main ()
{
        ...
        int count;     // лічильник елементів
        ...
}
```

3.5. Оператори в програмі на мові *C/C++*

3.5.1. Оператор присвоювання

Оператор присвоювання є одним з основних операторів в багатьох мовах програмування. Зазвичай оператор присвоювання використовується, як засіб встановлювати значення змінної. Синтаксис оператора:

змінна = вираз;

Параметр змінна є ідентифікатор, описаний як змінна. Параметр вираз являє собою деякий обчислювальний вираз, значення яке буде визначено в ході виконання даного оператора. Такий вираз може бути досить складним, або, навпаки, простим – наприклад, константою або змінною. Обчислення в вираженні значення буде присвоєно зазначеної змінної. На рис. 3.6 представлена схема виконання оператора *c= a+b;*.

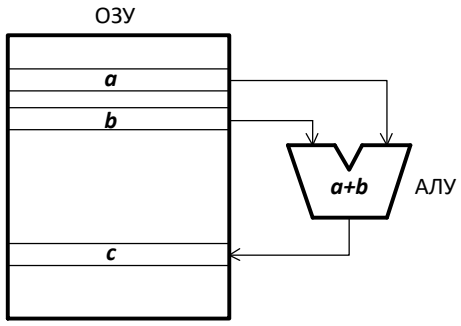


Рисунок 3.6 – Схема виконання оператора складання

Вираз, заданий в деякому операторі присвоювання, обчислюється, наприклад, з використанням значень змінних *a* і *b*. Значення цих змінних зберігаються в осередках ОЗУ. При виконанні оператора ці значення мікропроцесором і поміщаються в його спеціальний обчислювальний блок – арифметико-логічний пристрій (АЛП). В АЛП виконуються всі необхідні обчислювальні операції, і результат поміщається в відповідному полі пам'яті.

Вираз містить зазвичай набір операцій і операндів. Операнд це програмна конструкція, над якою виконуються обчислювальні дії. Операнд може бути константою, змінною чи функцією. Власне обчислювальний дію називається операцією.

Залежно від кількості операндів операції бувають:

1. Унарні операції – містять тільки один операнд: операнд операція або операція операнд.

2. Бінарні операції – містять два операнда: операнд1 операція операнд2.

3. Тернарние операції – містять 3 операнда.

Є такі види операцій:

1. Арифметичні операції.
2. Бітові операції.
3. Операції відношення.
4. Логічні операції
5. Операції присвоювання
6. Умовна операція.

3.5.2. Арифметичні, бітові і логічні операції

Арифметичні операції використовуються для перетворень з числовими значеннями. Існуючі арифметичні операції представлені в таблиці 3.2.

Результат операції ділення залежить типу операндів. Якщо хоча б один з операндів є речовим, то результат операції теж буде речовим, тобто буде містити дробову частину. Якщо ж обидва операнда цілі, то буде виконана операція ділення без залишку, тобто, наприклад, результат операції 14/5 дорівнюватиме 2.

Операція ділення по модулю дозволяє знайти залишок від ділення двох цілих операндів. Наприклад, результат операції ділення по модулю 14%5 буде 4.

Таблиця 3.2 – Арифметичні операції

Позначення	Тип	Назва	Результат	Приклад
+	бінарна	додавання	Сума двох операндів	$a + 5$
–	бінарна	віднімання	Різниця двох операндів	$b – a$

–	унарна	Зміна знака операнда	Значення операнда з протилежним знаком	*–x*
*	бінарна	множення	Твір двох операндів	9.8*f***mass*
/	бінарна	Розподіл або поділ без залишку	Приватне двох операндів	*PI/2*
%	бінарна	Розподіл по модулю	Залишок ділення операндів	7% 3
++	унарна	інкремент	Операнд збільшується на 1	*i* ++ або ++ *i*
–	унарна	декремент	Операнд зменшується на 1	*k* - - або – - - *k*

Операції інкремента і декремента змінюють значення операнда. Наприклад, якщо змінна *k* до операції мала значення 8, то після операції *k* — її значення стане дорівнювати – 7. Крім того, ці операції допускають префіксних і постфіксний записи. Від способу запису залежить порядок виконання операції. Розглянемо такі вирази:

а) *i* = 3;
5*(++*i*)
б) *i* = 3;
5*(*i*++)

Змінна *i* до виконання деякої операції приймає значення 3. Після виконання операції інкремента її значення стане дорівнює 4 в обох випадках. Але результат операції множення буде різним. При використанні префиксной записи (варіант а) спочатку буде виконано інкремент, а потім виконано множення. Тобто результат множення буде дорівнює 20. При використанні постфіксной записи (варіант б) спочатку буде виконана операція множення, а потім – інкремент. І результат операції множення буде дорівнює 15.

Бітові операції. Бітові операції виконуються над значеннями розрядів двійкового представлення цілого значення. Бітові операції представлені в таблиці 3.3.

Таблиця 3.3 – Бітові операції

Позначення	Тип	Назва	Результат	Приклад
&	бінарна	Бітове логічне “И”	Порозрядне логічне “И” операндів	*a & 5*
|	бінарна	Бітове логічне “ИЛИ”	Порозрядне логічне “ИЛИ” операндів	*x | 0xFF*
^	бінарна	Бітове логічне “НЕ-ИЛИ”	Порозрядне логічне “НЕ-ИЛИ” операндів	*a ^ 0x10*
~	унарна	Бітове логічне “НЕ”	Порозрядне логічне “НЕ” операнда	*~ MASK*
<<	бінарна	зрушення вліво	Зрушення першого операнда вліво	*cword << 4*
>>	бінарна	зрушення вправо	Зрушення першого операнда вправо	*cbyte >> 2*

Операція виконується над двійковими уявленнями операндів, причому операція проводиться безпосередньо над відповідними розрядами кожного операнда. Наприклад, уявімо операцію 5&3. Подання значень наступні: 510 = 01012, 310 = 00112.

0 1 0 1
&
0 0 1 1
———
0 0 0 1.

Так результат операції буде 00012 = 110.

Для бітових операцій логічних “И”, “ИЛИ” і “НЕ-ИЛИ” використовуються відповідні їм таблиці істинності (рис. 3.7).

X	*Y*	*X* & Y
0	0	0
0	1	0

X	*Y*	*X*| Y
0	0	0
0	1	1

X	*Y*	$X \oplus Y$
0	0	0
0	1	1

1	0	0
1	1	1

1	0	1
1	1	1

1	0	1
1	1	0

Рисунок 3.7 – Логічні операції: *а* – логічне “И”, б – логічне “ИЛИ”, *в* – логічне “НЕ-ИЛИ”

Також виконується унарна операція бітового логічного НЕ. Таблиця істинності задає інверсію кожного розряду двійкового представлення числа (рис. 3.8):

X	$\overleftarrow{Y}$
0	1
1	0

Рисунок 3.8 – Таблиця істинності логічного НЕ

Відповідно до такої таблиці розглянемо операцію ~ 0x07. Даному шістнадцятиричним поданням відповідає двійкове подання 00000111_2. При виконанні операції виконується інверсія кожного розряду, і результат операції в двійковому поданні буде дорівнює 11111000_2 або 0x*F*8 в шістнадцятковому.

Перераховані вище бітові операції зручно виконувати, використовуючи шістнадцяткове представлення значень. На практиці ці операції використовуються для зміни того чи іншого двійкового розряду числа. З таблиці істинності логічного “И” видно, що якщо виконати операцію для будь-якого значення біта з бітом, рівним 0, то результат такої операції буде 0. А якщо виконати операцію для біта з бітом, що дорівнює 1, то значення вихідного біта не зміниться.

Таким чином, можна встановлювати необхідні біти якогось значення в 0, використовуючи операцію логічного “И”.

В наступному прикладі в 0 встановлюються 3 молодших біта першого операнда:

01101110
& 11111000
_____
01101000

Виділена частина, як бачимо, не змінилася, а 3 молодших розряду встановилися в 0. Такий прийом називається маскуванням, а число 11111000 в даному випадку – маска.

Також для маскування можна використовувати операцію бітового логічного АБО. Значення розрядів маски, рівні 1 дозволяють встановити відповідні біти результату в 1, а нульові розряди маски не змінюють вихідні розряди:

```
  01101110
| 11111000
  ————————
  11111110
```

Відповідно за маскою старші 5 розрядів результату встановилися в 1, а молодші 3 розряду (виділені) не змінилися.

При виконанні маскування з використанням Виключающе Або одиничні розряди маски залишають початкове значення незмінним, а нульові розряди змінюють його на протилежне:

```
  01101110
^ 11111000
  ————————
  01101001
```

Відповідно за маскою старші 5 розрядів результату не змінилися, а молодші 3 розряди (виділені) інвертована. Операції зсуву мають наступний синтаксис:

операнд1 << операнд2
операнд1 >> операнд2

Результатом операції буде значення, рівне значенню операнд1, який зміщений на число розрядів, рівне операнд2. Наприклад, результат операція 5 << 2 буде виконана таким чином (рис. 3.9):

510 = 01012
0101002 = 2010

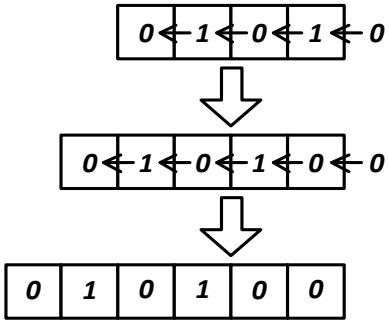


Рисунок 3.9 – Бітові операції зсуву 5 << 2

Можна помітити, що зсув на 1 розряд вліво еквівалентний множенню числа 2. Відповідно, зсув на 1 розряд вправо відповідає поділу числа на 2. Якщо число є непарним, що результатом зсуву буде ціле розподіл на 2 з втратою залишку (рис.3.10): 5 >> 1

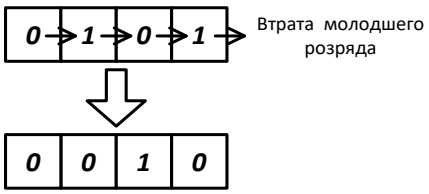


Рисунок 3.10 – Бітові операції зсуву 5 >> 1

Операції відношення. Операції відношення це група бінарних операцій, які дозволяють порівняти значення двох операндів, і результат порівняння є логічним значенням – *true* або *false*. Синтаксис операцій відношення:

*операнд*1 операція *операнд*2

Операції відношення представлені в таблиці 3.4.

Таблиця 3.4 – Операції відношення

Позначення	Назва	Результат
>	більше	*true*, Якщо операнд1 більше, ніж операнд2
<	менше	*true*, Якщо операнд1 менше, ніж операнд2

> =	Більше або дорівнює	*true*, Якщо операнд1 більше або дорівнює операнд2
<=	Менше або дорівнює	*true*, Якщо операнд1 менше або дорівнює операнд2
==	Так само	*true*, Якщо операнд1 дорівнює операнд2
! =	Не дорівнює	*true*, Якщо операнд1 НЕ дорівнює операнд2

Приклади операцій відношення і їх результати:

a == 5 результат *true*

a == 0 результат *false*

a! = 0 результат *true*

a<= 12 результат *true*

Логічні операції. Логічні операції зазвичай використовуються у виразах, що включають логічні змінні і константи, а також операції відношення. Результат логічної операції є значенням логічного типу: *true* або *false*. Логічні операції представлені в таблиці 3.5.

Таблиця 3.5 – Логічні операції

Позначення	Тип	Назва	Результат
&&	бінарна	логічне «И»	логічне значення *операнд*1 && *операнд*2 відповідно до таблиці істинності для логічного «И»
||	бінарна	логічне «ИЛИ»	логічне значення *операнд*1 || *операнд*2 відповідно до таблиці істинності для логічного «ИЛИ»
!	унарна	логічне «НЕ»	логічне значення *! операнд*1 відповідно до таблиці істинності для логічного «НЕ»

Розглянемо приклад: визначити, чи потрапляє значення змінної x в задані інтервали (рис. 3.11).

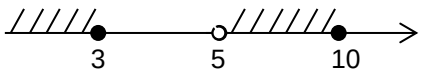


Рисунок 3.11 – Завдання 1

Для зазначених інтервалів треба визначити для змінної x істинність виразу $x \in (-\infty;3] \cup (5;10]$. Для вирішення цього завдання можна використовувати такий вираз (*x* <= 3) || (*x*> 5) && (*x* <= 10).

Результат такого виразу буде *true*, якщо значення змінної x належить заданими інтервалами, і *false* – якщо не належить. Зауважимо, що для об'єднання двох кордонів в один інтервал використовується операція '&&'. Кілька інтервалів об'єднуються з використанням операції '||'. Крім того, якщо інтервал відкритий, то використовується операція відношення зі строгаю нерівністю – в нашому прикладі *x*> 5. Якщо ж інтервал закритий, то використовується не строга нерівність – *x* <= 10. Операція логічного "НЕ" використовується для перевірки значення логічних змінних.

```
bool file_is_open = false;
...
// відкрити файл
...
// файл відкритий успішно
        file_is_open = true;
...
if (! file_is_open)
        // дії, якщо файл не був відкритий
```

Представлений фрагмент програми містить дії, які виконують деякі дії з файлом на диску. Для зручності роботи введена спеціальна змінна логічного типу *file_is_open*, яка має значення false, якщо файл не відкритий і, отже, не можна виконувати з цим файлом операції читання і запису. Якщо файл успішно відкритий, то змінна встановлюється в значення *true*.

Останній оператор перевіряє значення змінної і, якщо файл не був відкритий, виконує відповідні дії.

Операції присвоювання. У програмах часто використовуються оператори такого вигляду: *a* = *a* +7;

Суть такого оператора полягає в наступному: до поточного значення змінної a додається константа 7 і результат вираження записується в комірку пам'яті змінної *a*. У мові програмування *C/C++* є операції присвоювання для скороченого запису такого оператора: *a*+=7.

У таблиці 3.6 наведені доступні в мові операції присвоювання.

Таблиця 3.6 – Операції присвоювання

Позначення	Назва
+ =	Додавання з привласненням
– =	Віднімання з привласненням
* =	Множення з привласненням
/ =	Розподіл / ціле від ділення з привласненням
% =	Залишок від ділення з привласненням
& =	Бітове Логічне «И» з привласненням
| =	Бітове Логічне «ИЛИ» з привласненням
^ =	Бітове Логічне виключає «ИЛИ» з привласненням
>> =	Зрушення вправо з привласненням
<< =	Зрушення вліво з привласненням

Різниця полягає в тому, що при виконанні операції присвоювання змінюється значення першого операнда. Розглянемо наступні вирази:

а) 13.7*f* * (*y* + 3)

б) 13.7*f* * (*y* + = 3)

Значення всього виразу для варіантів а) і б) будуть однаковими. У варіанті а) значення змінної у не зміниться, в варіанті б) значення змінної у після обчислення всього виразу буде на 3 більше, ніж вихідне.

Умовна операція. Умовна операція є ще однією конструкцією мови, де використовуються логічні значення. Синтаксис оператора наступний:

*вираз*1 ? *вираз*2 : *вираз*3

Параметр вираження1 є вираженням логічного типу. При виконанні операції обчислюється значення параметра вираз 1. Якщо воно дорівнює *true*, то обчислюється вираз 2 і обчислене значення є результатом умовної операції в цілому. Якщо значення виразу 1 буде *false*, то результатом умовної операції буде значення параметра вираз 3. Наприклад, модуль значення деякої чисельної змінної val можна отримати, використовуючи наступну умовну операцію:

(Val> 0 *)? val : –val*

3.5.3. Основні математичні функції

Як зазначалося вище, в якості операнда в вираженні може використовуватися виклик функції. При обчисленні значення виразу, що містить функції в першу чергу виконується виклик цих функцій, обчислення їх значень і отримані значення використовуються як значення операндів в операціях. Виклик функції виконується зазначенням її імені переліком параметрів значень: констант, змінних, інших функцій:

імя_функциі (параметр1, параметр2, ...)

Основні математичні функції представлені в таблиці 3.7.

Таблиця 3.7 – Основні функції бібліотеки *math.h*

№	Ім'я функції	Значення, що повертається	Параметри	Тип результату
1	*abs (x)*	Абсолютне значення (модуль) аргументу	Ціле значення *x*	*int*
2	*acos (x)*	Функція *arcos (x)* в радіанах	Дійсне значення *x*	*double*
3	*asin (x)*	Функція *arcsin (x)* в радіанах	Дійсне значення *x*	*double*
4	*atan (x)*	Функція *arctg (x)* в радіанах	Дійсне значення *x*	*double*
5	*atan2 (x, y)*	Функція *arctg (x/y)* в радіанах,	Речові значення *x, y*	*double*
6	*cos (x)*	Функція *cos(x)*	Дійсне значення *x*	*double*
8	*exp (x)*	Функція e^x	Дійсне значення x	*double*
9	*fabs (x)*	Абсолютне значення (модуль) аргументу	Дійсне значення *x*	*double*
11	*labs (long)*	Абсолютне значення (модуль) аргументу	Довге ціле значення *x*	*long*
12	*log (x)*	Натуральний логарифм аргументу	Дійсне значення *x*	*double*
13	*log10 (x)*	Десятковий логарифм аргументу	Дійсне значення *x*	*double*

14	*pow (x, y)*	обчислення xy	Дійсне значення *x, y*	*double*
15	*sin (x)*	Функція *sin (x)*	Дійсне значення *x*	*double*
17	*tan (x)*	Функція *tg (x)*	Дійсне значення *x*	*double*
19	*sqrt (x)*	Квадратний корінь аргументу	Дійсне значення x	*double*

Для використання цих функцій необхідно підключити математичну бібліотеку директивою:

#include <math.h>

Приклади виразів з функціями:

12 + sin*(x)*

*amp*1 * cos *(w*1 * *t) – amp*2 * cos *(w*2 * *t)*

*sqrt(X * x + y * y)*

Часто зустрічаються випадки, коли параметром функції є інша функція. У цьому випадку використовується вкладення функцій:

*імя_функции*1 (імя_функціі2 (імя_функціі3 (параметр1) ...)

Наприклад для функції $\sin\sqrt{x}$ запис буде наступна: sin *(sqrt (x))*, а для функції $\sqrt{\sin(x)}$ запис буде *sqrt*(sin *(x)*).

Пріоритети виконання операцій у виразах. Синтаксис мови визначає порядок виконання операцій у виразах різних типів (далі наведено операції з спадання пріоритету):

1. Унарні операції: '!', '~', '–' (зміна знака), '++', '–'.
2. Арифметичні операції мультипликативной групи: '*', '/', '%'.
3. Арифметичні операції адитивної групи: '+', '–'.
4. Операції зсуву: '>>', '<<'.
5. Операції відносин: '<', '>', '<=', '> ='.
6. Операції відносин: ‘==’, ‘! =’.
7. Бітова операція «И» ‘&’.
8. Бітова операція виключає “ИЛИ”: «^ '.
9. Бітова операція “ИЛИ” ‘|’.
10. Логічна операція '&&'.
11. Логічна операція '||'.

12. Умовна операція.

13. Операції присвоювання '+ =', '– =', '* =', '/ =', '& =', '| =', '^ =', '<< =', '>> ='.

Звичайний порядок виконання операцій може бути змінений шляхом укладення деякої частини арифметичного виразу в круглих дужках, наприклад обчислення функції

$$y = \frac{\sqrt{a^2 - \sin 2a + 1}}{a - 12{,}8} + \cos\frac{\ln(a^2 + 3.14) - 1}{\sqrt{\sin a}}$$

буде реалізована наступним оператором присвоювання

$$y = sqrt\ (a * a - \sin\ (2 * a) + 1) / (a - 12.8) + \cos\ ((\log\ (a * a + 3.14) - 1) / sqrt\ (\sin\ (a)));$$

Дуже уважно слід ставитися до використання операцій з різним пріоритетом, коли застосовуються операції різних типів. Наприклад, необхідно виконати перевірку результату деякої бітової операції. Для цього можна використовувати вираз *a* & *b* == 1. Спочатку виконується бітова операція порозрядного логічного I, а потім перевіряється дорівнює чи результат цієї операції константі 1. Але відповідно до пріоритетом операцій вираз буде виконано наступним чином: значення змінної b порівнюється з константою 1 , потім результат операції відносини перетворюється в числове значення і виконується поразрядное логічне і зі значенням змінної a. Правильне вираження повинно мати вигляд (*a* & *b*) ==1.

Автоматичне перетворення типів і операції приведення

При роботі з виразами бажано, щоб типи всіх операндів у виразі відповідали один одному. При цьому забезпечується оптимальне виконання обчислювальних операцій, наприклад, коли всі змінні у виразі мають однакові типи. Але така умова, як правило, важко достижима. Крім цього бажано, щоб в операторі присвоєння ліва і права його частини були також однакових типів, і ця умова не завжди можлива.

Тип змінної задається при її описі. Тип вираження залежить від операндів і операцій, які входять до складу виразу. Забезпечити, щоб всі

змінні у виразі були однакових типів складно. Тому при обчисленні виразу компілятор виконує автоматичні перетворення (приведення) типів. Характер перетворення залежить від виду конкретної операції і типу операндів, існує загальний набір стандартних правил перетворення:

1. Якщо операція виконується з операндами двох різних типів, то операнди наводяться до «вищого» типу.

2. Якщо тип вираження і тип змінної в лівій частині оператора присвоювання різні, то після обчислення значення виразу це значення приводиться до типу змінної в лівій частині.

Послідовність імен типів, упорядкованих від «вищого» типу до «нижчого», виглядає наступним чином:

long double <– double <– float <– long <– int <– short <– char <– bool

Поряд з автоматичним перетворенням типів, виконуваних виконуючою системою, в мові С є можливість точно вказати тип даних, до якого необхідно привести деяку величину. Ця можливість реалізується в операції приведення типів наступним чином: перед даною величиною в круглих дужках записується ім'я необхідного типу. Нехай, наприклад, змінна res має тип int. Тоді значення арифметичного виразу

res = 2.7 + 1.5;

згідно із загальними правилами перетворення типів, так само 4. Однак застосувавши явно операцію приведення типу до обох операндам в правій частині

*res = (int)*2.7 + (*int*) 1.5;

отримаємо результат, рівний 3. Цей приклад говорить про необхідність з особливою обережністю ставитися до всіляких перетворень типів операндів.

3.5.4. Оператори введення/виведення

Оператори введення/виводу є також одними з основних операторів мови. Якщо розглядати будь-який додаток, то воно, як правило, має 3 основні частини:

1. Введення вихідних даних – здійснюється за допомогою пристроїв введення, з дискових файлів, каналів зв'язку, також можлива автоматична генерації даних.

2. Обробка отриманих вихідних даних.

3. Висновок даних – здійснюється пристроями відображення, через канали зв'язку, записуються в файл на диску і т.д.)

Розглянемо два типи операторів введення / виведення: «старого стилю», які використовуються в мові *C*, і можуть застосовуватися в *C* ++, і оператори потокового введення / виводу мови *C*++.

Оператори введення/виведення «старого стилю» (мова C/C++). Оператор введення дозволяє вводити значення змінних з клавіатури. Для використання оператора необхідне підключення бібліотеки введення / виведення <*stdio.h*>. Синтаксис оператора наступний:

scanf(«Формат», & ідентифікатор 1, & ідентифікатор 2, ...);

Першим параметром оператора є рядок «формат», яка визначає тип і уявлення даних, що вводяться. Після параметра “формат” вказуються список ідентифікаторів змінних, значення яких будуть встановлені в даному операторі. Слід зазначити, що в операторі scanf використовується не ім'я, а адреса змінної, значення якого буде встановлено. Це виконується операцією & ідентіфікатор1. Унарна операція ‘&’ дозволяє отримати адресу осередки пам'яті, де зберігається значення змінної.

Оператор виводу дозволяє виводити значення змінних на екран. Для використання оператора необхідне підключення бібліотеки введення / виведення <*stdio.h*>. Синтаксис оператора наступний:

printf(«Формат», ідентіфікатор1, ідентіфікатор2, ...);

Параметри оператора printf аналогічні оператору scanf. Різниця полягає в тому, що список ідентифікаторів містить саме ідентифікатори, а не їх адреси. Оператор виводить на екран значення змінних, зазначені в списку ідентифікаторів.

Параметр «формат» є рядком, в якій можна задати вид рядка введення / виведення, а також деякі параметри вводяться і виведених значень.

Наприклад, розглянемо оператори

char stud[] = «Іванов»;
unsigned int alg =5, *mathem* = 4;
printf («Студент %s отримав по програмуванню %u, а з математики %u», stud, alg, mathem);

У пункті «формат» заданий текст, який буде виведений на екран. Символами '%s' і '%u' задані місця, де в виведеному тексті будуть виведені значення змінних. При формуванні рядка виводу, якщо зустрічається рядок з префіксом '%', замість неї вставляється значення відповідної змінної зі списку. Таким чином, порядок змінних в списку повинен відповідати їх порядку в рядку «формат». Так, в нашому прикладі форматної рядку '%s' буде відповідати змінна '*stud*', першому '%u' – змінна '*alg*', а другого '%u' – змінна '*mathem*'. Інформація, що виводиться рядок буде мати вигляд:

Студент Іванов отримав по програмуванню 5*, а з математики* 4

Для завдання параметрів виведення змінної використовується рядок з префіксом '%'. Рядок має такий синтаксис:% Ширина.ТочністьТип

Параметри «ширина» і «точність» не є обов'язковими. Параметр «тип» потрібно вказувати завжди. Цей параметр задається символом і визначає, як буде інтерпретована змінна при виведенні її на екран – рядок, символ, числове значення відповідного типу. Тип змінної і тип, заданий в параметрі «формат» повинні відповідати один одному. Нижче в таблиці 3.8 представлені символи, які можна використовувати і відповідні їм типи.

Таблиця 3.8 – Опис типів для операторів *printf / scanf*

Символ	Тип	Опис
c	*char*	Виводить на екран / вводить з клавіатури значення символьної змінної
d	*int*	Виводить на екран / Вводить з клавіатури знакова десяткове ціле значення
i	*int*	Виводить на екран / Вводить з клавіатури знакова десяткове ціле значення

o	*int*	Виводить на екран / Вводить з клавіатури беззнаковое вісімкове ціле значення
u	*int*	Виводить на екран / Вводить з клавіатури беззнаковое десяткове ціле значення
x	*int*	Виводить на екран / Вводить з клавіатури беззнаковое шестнадцатеричное ціле значення, використовуються цифри «*abcdef*»
X	*int*	Виводить на екран / Вводить з клавіатури беззнаковое шестнадцатеричное ціле значення, використовуються цифри «*ABCDEF*»
e	*double*	Виводить на екран / Вводить з клавіатури дійсне значення змінної в експоненційному форматі men, де *m* – мантиса числа, *n* – його порядок, мантиса і порядок можуть мати знак
E	*double*	Те ж що і e, але роздільник мантиси і порядку відображається як *E*
f	*double*	Виводить на екран / Вводить з клавіатури дійсне значення змінної в форматі з фіксованою точкою *x.y*, де *x* – ціла частина числа, *y* – дрібна частина
p	*void **	Виводить на екран / Вводить з клавіатури покажчик на змінну (адреса змінної) в шістнадцятковому форматі
s	рядок символів	Виводить на екран / Вводить з клавіатури рядок символів

Параметр «ширина» дозволяє задати загальну кількість знаків на екрані для виведення значення змінної. Параметр «точність» визначає кількість знаків на екрані для подання дробової частини речового значення:

% 5*d* – буде виведено ціле десяткове значення, якщо значущих цифр значення менше 5, то зліва від числа будуть виведені додаткові пробіли, щоб загальне число знаків числа дорівнювало 5.

% 2*X* – буде виведено шістнадцяткове значення з цифрами *A, B, C, D, E, F*, для виведення буде відведено 2 символи на екрані.

% *e*
% 7.4*f*
% .2*f*

Розглянемо наступний приклад. Потрібно написати програму, яка обчислює довжину однієї зі сторін a довільного трикутника за двома відомими сторонами *b*, c та кути між ними. Для вирішення такого завдання зручно використовувати теорему косинусів: $a^2 = b^2 + c^2 - 2bc\cos\alpha$.

```
#include <math.h>
#include <stdio.h>
#define PI 3.1415926
#define DEG2RAD PI /180
void main ()
{
        int a, b, c;
        float alfa;
        printf ( «Введіть довжини2 сторін і кут між ними в градусах «);
        scanf ( «% u% u% f», & b, & c, & alfa);
        a = b * b + c * c – 2 * b * c * cos (alfa * DEG2RAD);
        printf( «Сторони трикутника дорівнюють% u,% u,% u \ n», (int) sqrt (a), b, c);
        printf( «Кут між сторонами дорівнює%.4f \ n «, alfa);
}
```

Для роботи програми були підключені бібліотека <*math.h*> для виконання математичних функцій cos і sqrt і бібліотека <*stdio.h*> для роботи з функціями введення / виводу. Приймемо, що сторони задаються цілими числами, а кут дійсним числом. Визначено макроси для числа *PI*, а також макрос для перекладу значення кута з градусів в радіани *DEG2RAD*.

Оператори потокового введення/виводу (мова *C*++). У мові програмування С ++ використовується інша модель, заснована на потоках введення / виведення – потоковий ввід / вивід. Пристрій введення / виводу є логічний канал, з яким можна обмінюватися даними, а також здійснювати настройку цього каналу за допомогою спеціальних керуючих послідовностей. Для роботи з потоковим вводом / виводом необхідне підключення бібліотеки <*iostream*>.

Для введення даних використовується оператор введення >>

cin>> переменная1 >> переменная2 >> ...;

При виконанні цього оператора змінним, зазначеним в ньому встановлюються значення, що вводяться з клавіатури.

Для виведення даних використовується оператор виведення <<

cout << *змінна*1 << переменная2 << вираження1 ...;

При виконанні цього оператора на екран виводяться значення змінних і виразів зазначених в ньому.

```
int x, y;
cin >> x >> y;
cout << «Для катетів довжиною» << x << «і» << y << «гіпотенуза» << sqrt (x * x + y * y) << '\ n';
```

При виконанні цього фрагмента програми користувач повинен ввести значення змінних x і y. Якщо, наприклад, x дорівнює 3, а y дорівнює 4, на екран буде виведений рядок

Для катетів довжиною 3 і 4 гіпотенуза 5

Символ '*n*' служить для вказівки, що після виконання оператора курсор буде переведений на наступний рядок.

Для управління каналом потокового введення / виводу використовуються спеціальні маніпулятори. Деякі маніпулятори наведені в таблиці 3.9. Для використання маніпуляторів необхідно підключити заголовний файл <*iomanip*>

Таблиця 3.9 – Маніпулятори в операторах *cin/cout*

Маніпулятор	Дія
setw (n)	Цілий параметр *n* вказує кількість символів на екрані для виведення значення змінної (вираження)

setprecision (p)	Визначає точність для виведення дійсних значень, параметр *p* задає кількість знаків після коми для виведеного на екран значення, якщо заданий маніпулятор *fixed*, або загальне число знаків значення
setbase (b)	Задає системи числення для виведених на екран значень, параметр b може приймати значення 8, 10, 16
left	Якщо при виведенні значення зарезервовано певну кількість символів модифікатором *setw*, А число значущих символів менше, то виведене значення вирівнюється по лівому краю зарезервованого простору
right	Якщо при виведенні значення зарезервовано певну кількість символів модифікатором *setw*, А число значущих символів менше, то виведене значення вирівнюється по правому краю зарезервованого простору
setfill (c)	параметр *c* дозволяє задати символ, який буде використовуватися для заповнення незначущих символів, по символом за замовчуванням є символ пропуску
hex	Теж що *setbase* (16)
dec	Теж що *setbase* (10)
oct	Теж що *setbase* (8)
showbase	При виведенні числового значення виводиться префікс, відповідний системі числення, в якій представлено значення
noshowbase	При виведенні числового значення не виводиться префікс, відповідний системі числення, в якій представлено значення
scientific	Дійсні значення будуть виводитися в форматі з плаваючою точкою
fixed	Дійсні значення будуть виводитися в форматі з фіксованою точкою

Приклади:

1. Значення змінних *a* і *d* будуть виведені в десятковій системі числення, значення змінної *b* буде виведено в шістнадцятковій системі числення без префікса, значення змінної c буде виведено в шістнадцятковій системі числення з префіксом, вивод буде таким «134 20 0x39 1783» (20 це шістнадцяткове подання числа 32, 39 шістнадцяткове представлення числа 57).

a = 134; b = 32; c = 57; d = 1783;

cout << *a* << '' << *setbase*(16) << noshowbase << b << '' << showbase << c << '' << dec << d;

2. Буде виведено число 15, для виведення числа зарезервовано 7 знакомісць, порожні знакомісця будуть заповнені символом 'x', висновок буде таким «xxxxx15 73xxxxx xxxx187».

cout << *setfill ('x')* << *setw*(7) << 15 << '' << left << 73 << '' << right << 187;

3. Для описаних дійсних змінних встановлюється точність 5 знаків у дробовій частині, далі виводяться значення без додаткових модифікаторів, в форматі з фіксованою точкою і в форматі з плаваючою точкою.

double a, b, c;
a = 3.1415926534;
b = 2010.0;
c = 1.12e–15;
cout << *setprecision*(5);
cout << *a* << *'\ T'* << *b* << *'\ T'* << *c* << *endl;*
cout << *fixed;*
cout << *a* << *'\ T'* << *b* << *'\ T'* << *c* << *endl;*
cout << *scientific;*
cout << *a* << *'\ T'* << *b* << *'\ T'* << *c* << *endl;*

Вивод буде таким
3.1416 2010 1.12e–015
3.14159 2010.00000 0.00000
3.14159e +000 2.01000e + 003 1.12000e–015

Вище розглядалися типи алгоритмів, і найпростішим типом алгоритму є лінійний алгоритм. У такому алгоритмі всі оператори виконуються послідовно – оператор1, оператор2 і т.д. Після виконання операторN алгоритм закінчує своє виконання. Мовою *C/C++* такий алгоритм реалізується в такий спосіб:

void main ()
{
*оператор*1;
*оператор*2;

```
  ...
  оператор N;
}
```

Прикладом використання лінійного алгоритму може бути задача обчислення значення деякого виразу y для введеного з клавіатури числа *x*:

$$y = |x-1| + \frac{x^2 - 2x + 1}{12}$$

Алгоритм рішення задачі представлений на рис. 3.12.

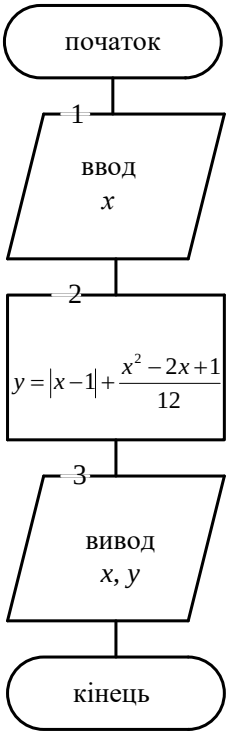


Рисунок 3.12 – Блок–схема алгоритму лінійної програми

Програма, яка реалізує даний алгоритм, наведена нижче.

```
#include <iostream>

void main ()
{
  double x, y;
  cout << «Vvedite chislo x –»;
```

```
  cin >> x;
  y = fabs(X–1) + (pow (x, 2) – 2 * x + 1) / 12;
  cout << «y =« << y;
}
```

Результатом роботи даної програми є:

Vvedite chislo x – 3.5

y= 0.23545

Лінійні алгоритми використовуються при вирішенні певного класу обчислювальних задач, але коло таких завдань досить вузький. Крім того, при використанні лінійних алгоритмів обсяг програм виходить досить великий. Перевагою таких алгоритмів є їх висока швидкодія.

Наприклад необхідно написати програму, яка знаходить і виводить на екран суму цифр тризначного цілого числа, що вводиться з клавіатури.

Для знаходження однакових цифр в тризначному цілому числі необхідно скористатися арифметичними операторами: % – знаходження залишку від ділення двох цілих чисел, / – цілочисельне ділення двох цілих чисел.

Наприклад: ціле тризначне число $n = 962$.

$a = n / 100$; $a = 9$ – перша цифра числа n.

$d = n\%100$; $d = 62$;

$b = d / 10$; $b = 6$ – друга цифра числа n.

$c = d\%10$; $c = 2$ – третя цифра числа n.

Блок–схема алгоритму програми представлена на малюнку 3.13.

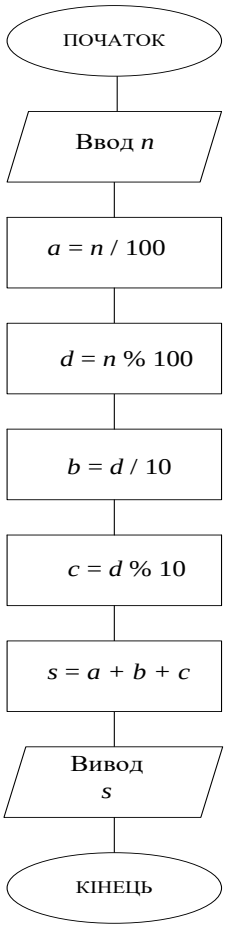


Рисунок 3.13 – Блок-схема алгоритму програми завдання 3.2

```
#include <iostream>
void main ()
{
        int n, a, b, c, d, s;
        cout << «Vvedite chislo \ n»; // Виведення тексту
        cin >> n; // Введення значення змінної n
        a = n/ 100; // Обчислення першої цифри числа n
        d = n% 100; // Обчислення залишку від ділення числа n на 100
        b = d/ 10; // Обчислення другої цифри числа n
        c = d% 10; // Обчислення третьої цифри числа n
        s= a + b + c; // сума цифр
        cout << “s =” << s << '\ n';
}
```

Результатом роботи даної програми є

Vvedite chislo – 235 *s* = 10

Практичні завдання

1. Обчислити значення функції $f(x) = 3{,}5\ x^3 + \cos(x)/\sin(2*x) + e^x$, де x=0,1. Округлити значення $f(x)$ до найближчого цілого числа. Знайти залишок від ділення цілої частини на 5 і вивести результат на екран.

2. Обчислити значення функції $f(x) = \cos(x + 1)*\mathrm{tg}(x)/(\ln(x) + 2{,}5)$, де x – будь-яке число з інтервалу [0 ... 1], що вводиться з клавіатури в діалоговому режимі. Визначити приналежність $f(x)$ відрізку [3 ... 9].

3. Ввести три цілих двозначних числа. Знайти суму перших цифр даних чисел. Вивести результат на екран.

4. Ввести два дійсних числа c, d. Знайти число x, відповідне цілому від ділення c і d. Визначити куб числа x і вивести результат на екран.

5. Ввести два цілих числа a і b. Знайти залишок від ділення a і b. Визначити приналежність залишку інтервалу [0 ... 4].

6. Визначити квадрат чотиризначного цілого числа, отриманого виписуванням в зворотному порядку.

7. Визначити куб тризначного числа, отриманого виписуванням в зворотному порядку цілої частини дійсного числа.

8. Обчислити цілу частину середнього арифметичного заданих чотирьох позитивних чисел.

9. Обчислити значення $f(x) = \sin(x) + \arccos(x)$, де x – будь-яке число з діапазону [0 ... 1]. Вивести на друк цілу частину значення $f(x)$.

10. Ввести будь дійсне число. Округлити його до найближчого цілого. Вивести результат на екран.

11. Знайти твір цифр заданого чотиризначного числа.

12. Обчислити значення $f(x) = \ln(x)/\sin(x)+e^x$, де x=0.7. Знайти залишок від ділення цілої частини $f(x)$ на $y(x)$, де $y(x)=\arcsin(x)+\arccos(x)+ + |x|$.

13. Обчислити довжину окружності, площа кола і об'єм кулі одного і того ж заданого радіуса.

14. Обчислити периметр і площу прямокутного трикутника за довжинами двох катетів.

15. Знайти добуток цифр заданого чотиризначного числа.

16. Визначити число, отримане виписуванням в зворотному порядку цифр заданого тризначного числа.

17. Визначити, дорівнює чи сума двох перших цифр заданого чотиризначного числа сумою двох його останніх цифр.

18. Визначити, чи рівний квадрат заданого тризначного числа кубу суми цифр цього числа.

19. Визначити, чи є серед перших трьох цифр заданого чотиризначного числа цифра 0.

20. Ввести три цілих тризначних числа. Знайти суму перших цифр даних чисел, а також твір останніх цифр. Результат на екран.

Контрольні питання

1. Описи в програмі на мові програмування *C/C++*. Види описів.

2. Вирази і операції в програмі на мові програмування *C/C++*. Типи операцій.

3. Поняття змінної в програмі. Опис змінних в *C/C++*. Поняття типу змінної.

4. Арифметичні операції. Призначення.

5. Додаток і бібліотеки як види програмного забезпечення. Призначення, основні відмінності. Типи бібліотек.

6. Типи змінних. Призначення. Цілі типи даних, що використовуються в мові програмування *C/C++*.

7. Оператор присвоювання. Призначення. Синтаксис. Призначення.

8. Типи змінних. Призначення. Речові типи даних, що використовуються в мові програмування *C/C++*.

9. Логічні операції. Призначення.

10. Константи в програмі. Використання. Запис констант різних типів.

11. Пріоритети операцій у виразі. Зміна пріоритетів операцій. приклади

12. Автоматичне перетворення типів у виразі. Операції приведення типів.

13. Оператори в програмі. Призначення, використання.

14. Оператори введення даних в мові програмування *C/C++*. Призначення, синтаксис.

15. Оператори виведення даних в мові програмування *C/C++*. Призначення, синтаксис.

РОЗДІЛ 4.

ОПЕРАТОРИ ПЕРЕДАЧІ УПРАВЛІННЯ. ЦИКЛИ

4.1. Умовний оператор *if*

Алгоритм з розгалуженням містить кілька шляхів, за якими може виконуватися програма. Вибір шляху виконується вже в ході виконання самої програми. Такі алгоритми є більш поширеними, тому що надають велику гнучкість при вирішенні завдання, дозволяють створювати більш компактні програми. Часто рішення задачі неможливо без реалізації розгалуження в принципі. Точка в алгоритмі, в якій відбувається такий вибір, називається точкою розгалуження. У точці розгалуження програма повинна містити той чи інший оператор передачі управління. Далі розглянемо оператори передачі управління мови *C/C ++*.

Умовний оператор *if*. Умовний оператор є найпростішим і ефективним засобом, що дозволяє програмувати алгоритми з розгалуженням. Умовний оператор реалізується наступними алгоритмами (рис. 4.1):

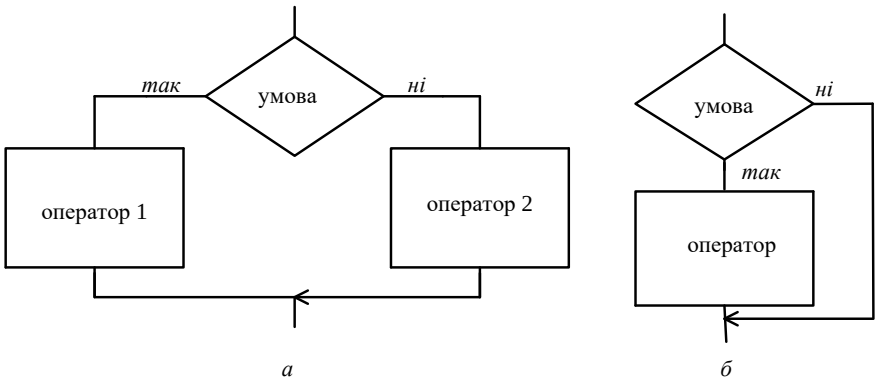


Рисунок 4.1 – Блок–схема умовного оператора *if*

В даному алгоритмі параметр *умова* являє собою вираз логічного типу. Якщо цей вислів приймає значення *true*, то виконується оператор1. Якщо ж вираз приймає значення *false*, то виконується оператор2 (рис.4.1,*а*) або оператор, який слід безпосередньо за умовним оператором (рис. 4.1, *б*).

Якщо вираз в умови має тип, відмінний від логічного, то значення виразу буде приведено до логічного типу.

Синтаксис умовного оператора в мові *C/C++* наступний. Алгоритмом, представленим на рис. 4.1, а відповідає оператор

```
if(умова)
        оператор1;
else
        оператор2;
```

Алгоритмом, представленим на рис. 4.2, б відповідає оператор

```
if(умова)
        оператор1;
```

Синтаксис мови вимагає, щоб в умовному операторі параметри оператор1 і оператор2 були представлені тільки одним оператором. Але часто потрібно, щоб було виконано кілька операторів. Для цього використовується складений оператор:

```
if(умова)
{
        оператор1;
        оператор2;
}
else
{
        оператор3;
        оператор4;
}
```

В даному випадку оператори, укладені у фігурні дужки розглядаються як один оператор, що задовольняє синтаксису мови програмування.

Як приклад використання умовного оператора розглянемо його застосування в алгоритмах маніпулювання з цілими числами. В основі таких алгоритмів лежить можливість подання будь-якого цілого числа у вигляді:

$$N = k * m + r$$

В даному співвідношенні k є ціла частина, отримана від ділення N на m ($k = N / m$), а r є залишком від такого поділу ($r = N\% m$).

Так, для визначення парності цілого числа необхідно перевірити значення залишку r при розподілі довільного цілого числа N на 2. Якщо значення r дорівнює 0, то число є парним. Алгоритм рішення задачі представлений на рис. 4.2.

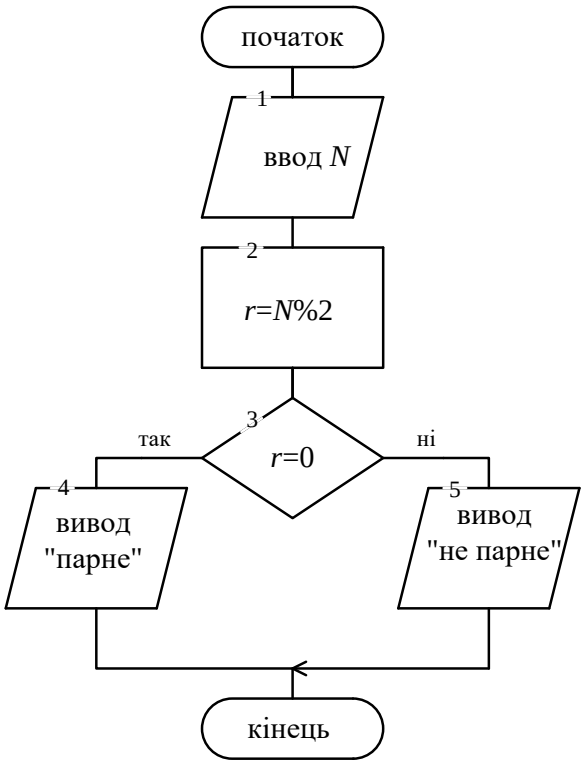


Рисунок 4.2 – Блок-схема алгоритму розв'язання задачі

Програма, яка реалізує алгоритм, представлена нижче:

```
#include <iostream>
void main()
{
        int N, r;
        cin >> N;
        r = N% 2;
        if(R == 0)
```

```
        {
            cout << «парне» << '\ n'
            cout << «ура!» << '\ n'
        }
else
        cout << «непарне» << '\ n'
}
```

Оператори

```
r = N%2;
if(r == 0)
{...
}
```

можна замінити наступним оператором

```
if(! (N% 2))
{...
}
```

При виконанні такого оператора буде обчислено значення виразу *N*%2. Цей вислів цілого типу і його результат теж буде цілим числом: 0, якщо *N* парне, або 1, якщо *N* непарне. Далі буде виконано приведення обчисленого значення до логічного типу: значення 0 буде приведено до логічного значення *false*, значення 1 буде приведено до *true*. Після чого буде виконана інверсія логічного значення – операція «!». Таким чином, якщо *N* є парним числом, то результат виразу! (*N*% 2) буде *true* і будуть виконані відповідні дії в програмі.

У наступному прикладі не обходимо вивести на єкран повідомлення, чи є серед цифр тризначного цілого числа, що вводиться з клавіатури, однакові. Для знаходження однакових цифр в тризначному цілому числі необхідно скористатися арифметичними операторами: % – знаходження залишку від ділення двох цілих чисел, / – цілочисельне ділення двох цілих чисел.

Наприклад: ціле тризначне число *n* = 962.

a = *n* /100; *a* =9 – перша цифра числа *n*.

d = *n*%100; *d* =62;

$b = d /10;$ $b = 6$ – друга цифра числа n.

$c = d\%10;$ $c = 2$ – третя цифра числа n.

Блок–схема алгоритму програми представлена на малюнку 4.3.

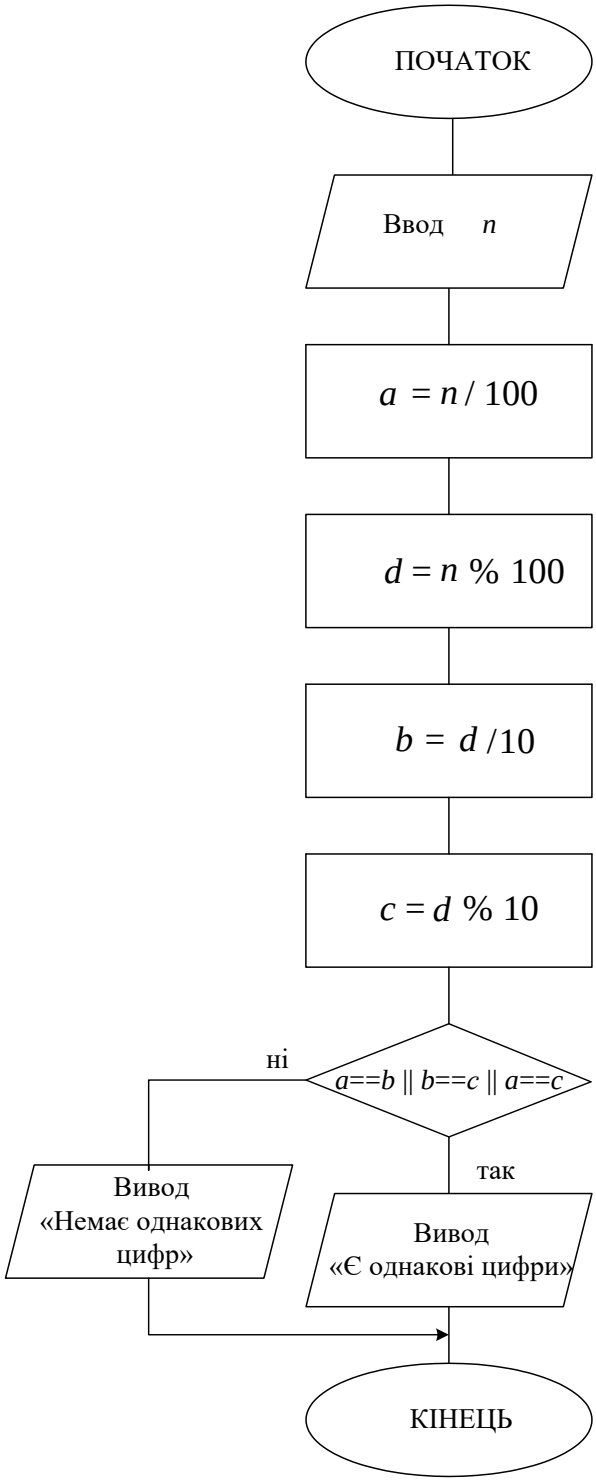


Рисунок 4.3 – Блок–схема алгоритму програми

```
#include <iostream>
```

```
using namespace std;

int main ()
{
  int n, a, b, c, d;
  bool k;
  cout << «Введіть ціле тризначне число \ n»;
   cin >> n;
  a = n / 100;
  d = n% 100;
  b = d /10;
  c = d% 10;
  if ((a == b) || (b == c) || (a == c))
        cout << «Є серед цифр числа» << n << «однакові?»;
 else
        cout << «Немає серед цифр числа» << n << «однакові?»;
 return 0;
}
```

Результати роботи програми.

1–й запуск

Введіть ціле тризначне число

235

*Немає серед цифр числа*235 однакових

2–й запуск

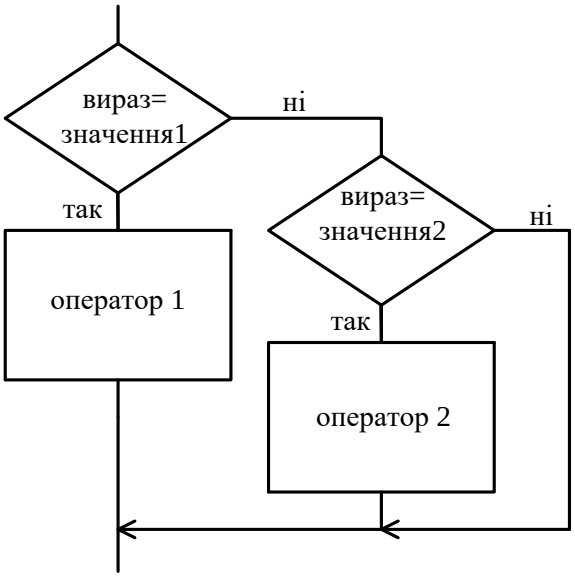
Введіть ціле тризначне число

565

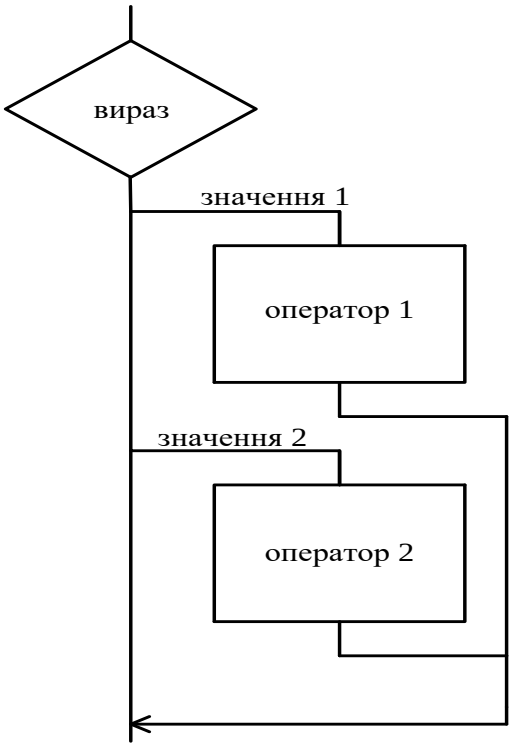
*Є серед цифр числа*565 однакові

4.2. Оператор-перемикач *switch*

Перемикач є спеціальним випадком умовного оператора і дозволяє здійснювати багатоваріантний вибір, замінюючи групу вкладених операторів *if-else*. Схема алгоритму, яка реалізується оператором наступна (рис. 4.4)



а



б

Рисунок 4.4 – Блок-схема алгоритму оператора *switch*

Параметр *вираз* представлений виразом перечислимого типу – цілого, символьного, логічного або призначеного для користувача. При виконанні оператора обчислюється цей вислів і отримане значення порівнюється з заданими в операторі значеннями: *знач* 1, *знач* 2, ..., *знач n*.

При збігу з одним зі значень виконуються оператори, які відповідають даним значенням.

У мові *C/C++* оператор–перемикач має такий синтаксис:

```
switch (вираз)
{
 case знач1:
 {
        оператор1;
        оператор2;
        break;
 }
 case знач2:
 {
        оператор3;
        оператор4;
        break;
 }
        ...
 default:
 {
        оператор5;
        оператор6;
        break;
 }
}
```

Секція *default* виконується, якщо значення виразу не дорівнює жодному із зазначених в операторі значень. Ця секція може бути відсутнім.

Як приклад розглянемо програму, яка реалізує найпростіший калькулятор.

```
#include <iostream>
void main ()
{
 double x, y, z;
```

```
 char op;
 cout << «Введіть операцію» << '\ n';
 cin >> x >> op >> y;
 switch (op)
 {
        case '+':
                z = x + y;
                break;
        case '– ':
                z = x – y;
                break;
        case '*':
                z = x * y;
                break;
        case '/':
                z = x / y;
                break;
        default:
                cout << «Не знаю такої операції» << '\ n';
 }
 cout << «Результат «<< setprecision (3) << z << '\ n';
}
```

Після запуску програми користувач повинен ввести необхідну операцію. Наприклад, «4 + 8». Після чого на екран буде виведений рядок з результатом операції. При введенні знака операції, відмінного від ‘+’, ‘–‘, ‘*’, ‘/’ буде виведено повідомлення «Не знаю такої операції».

Для того щоб виконувалися одні й ті ж оператори для декількох заданих значень, ці значення можна групувати. Розглянемо програму, яка дозволяє визначити який день тижня є робочим, а який – вихідним.

```
int weekday;
cin >> weekday;
switch (weekday)
{
        case 1:
```

```
        case 2:
        case 3:
        case 4:
        case 5: cout << «робочий день☹«<< '\ n';
                break;
        case 6:
        case 7: cout << «вихідний день☺«<< '\ n';
                break;
}
```

У даній програмі в змінної *weekday* міститься введений з клавіатури номер дня тижня. Дні тижня з номерами від 1 до 5 (понеділок, вівторок, середа, четвер, п'ятниця) ідентифікуються як робочі дні, 6 та 7 дні тижня (субота і неділя) є вихідними днями.

4.3. Оператори циклу

Цикл – різновид керуючої конструкції в мовах програмування, призначена для організації багаторазового виконання набору операторів. Послідовність операторів, призначена для багаторазового виконання, називається <*тілом циклу*>. Одноразове виконання тіла циклу називається <*ітерацією*>. Вираз визначає, чи буде в черговий раз виконуватися ітерація, або цикл завершиться, називається <*умовою виходу* «або» умовою закінчення циклу> (або <*умовою продовження*> в залежності від того, як інтерпретується його істинність – як ознака необхідності завершення або продовження циклу). Змінна, що зберігає поточний номер ітерації називається <*лічильником ітерацій*> циклу або просто <*лічильником циклу*>.

Існують кілька типів циклічний дій:

1. Цикл з передумовою.
2. Цикл з післяумовою.
3. Цикл з параметром (параметричний цикл, цикл з лічильником).

Цикл з передумовою. Оператор дозволяє циклічно виконувати певну послідовність операторів доти, поки істинно деякий умова, що перевіряється перед початком чергової ітерації циклу. Схема алгоритму для циклу з передумовою представлена на рис. 4.5.

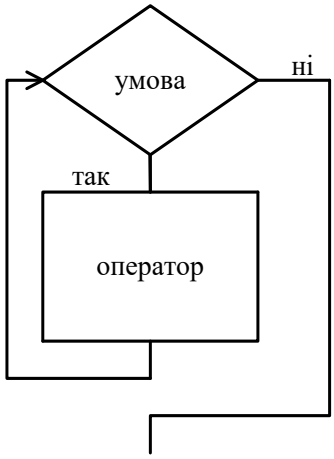


Рисунок 4.5 – Цикл з передумовою *while*

Параметр *умова* являє собою вираз логічного типу. Якщо тип цього виразу інший, то проводиться приведення його до логічного. Якщо значення виразу одно *true*, то проводиться виконання операторів в *тілі циклу*. Після виконання чергової ітерації проводиться повернення до обчислення значення виразу в умови, після чого приймається рішення про продовження виконання циклічних дій. Таким чином, в тілі циклу необхідно організувати ситуацію, коли виконуються дії, які змінюють виконання вираження в умови. Наприклад, можна змінювати всередині циклу лічильник і перевіряти його значення в умови. В іншому випадку можливий так званий «нескінченний цикл», що призведе до невірного виконання програми.

Необхідно відзначити, що умова перевіряється до виконання операторів тіла циклу, тому можлива ситуація, коли тіло циклу жодного разу не буде виконано.

В *C/C++* цикл з передумовою реалізований оператором *while*:

while(умова)
    оператор;

Синтаксис мови вимагає, щоб в тілі циклу був присутній тільки один оператор. При необхідності необхідно використовувати складовою оператор:

```
while (умова)
{
        оператор1;
        оператор2;
        ...;
}
```

Як приклад використання оператора *while* можна привести організацію програмного текстового меню в консольному додатку. При запуску такого додатка на екран виводиться перелік можливих дій, кожна дія пов'язане із заданою клавішею і користувач, натискаючи необхідну клавішу, запускає ту чи іншу на виконання.

```
void main()
{
  char choice = ‘0’;
  while(choice! = ‘4’)
  {
        cout << «Виберіть дію» << '\ n';
        cout << «1. Введення даних «<< '\ n';
        cout << «2. Обробка «<< '\ n';
        cout << «3. ... «<< '\ n';
        cout << «4. Вихід «<< '\ n';
        cin >> choice;
        switch(Choice)
        {
              case '1 ':
              {
              // дії з введення даних
                    ...
                    break;
              }
              case '2 ':
              {
              // дії по обробці даних
                    ...
                    break;
              }
              ...
              case '4 ':
```

```
                {
                cout << «Ви завершили роботу з програмою» << '\n ';
                        break;
                }
                default:
                {
                cout << «Невірно, повторіть вибір» << '\n ';
                        break;
                }
        }
}
```

В даному прикладі на екран виводиться текстове меню. Користувач повинен ввести номер пункту меню для вибору дії. Введений символ зберігається в змінної choice. Якщо вводиться один з перерахованих в меню символів (крім символу ‘4’), в операторі *switch* виконуються відповідні цим символом дії – введення даних, обробка даних і т.д. Після виконання цих дій меню знову виводиться на екран і очікується вибір наступного дії. Якщо введений символ, якого немає в меню, оператор switch виводить на екран повідомлення про помилку, і знову виводиться меню і очікується вибір користувача. Якщо користувач вводить символ ‘4’, то оператор *switch* виводить на екран повідомлення про завершення роботи, відбуватиметься повернення до перевірки умови циклу, умова перестає виконуватися і цикл припиняє свою роботу. Після чого додаток завершується.

Цикл з післяумовою. Оператор дозволяє циклічно виконувати певну послідовність операторів доти, поки істинно деякий умова, що перевіряється після чергової ітерації циклу. Схема алгоритму для циклу з умовою поста представлена на рис. 4.6.

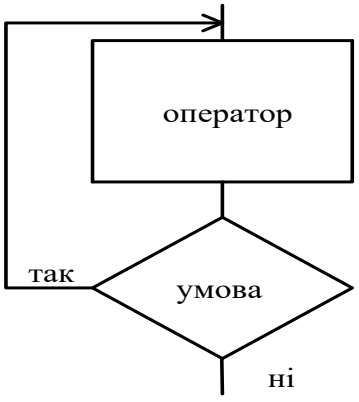


Рисунок 4.6 – Цикл з післяумовою *do ... while*

Параметр *умова* являє собою вираз логічного типу. Якщо тип цього виразу інший, то проводиться приведення його до логічного типу. Якщо значення виразу *true*, то відбувається повернення до виконання операторів в тілі циклу. Після виконання чергової ітерації перевіряється значення виразу в *умови*, після чого приймається рішення про продовження виконання циклічних дій. Необхідно відзначити, що умова перевіряється після виконання операторів тіла циклу, тому оператори тіла циклу виконуватися хоча б один раз. В *C/C++* цикл з передумовою реалізований оператором *do..while*:

```
do
{
  оператор1;
  оператор2;
  ...;
}
while(умова);
```

Розглянутий вище приклад можна реалізувати з використанням циклу з передумовою. В скороченому вигляді це буде виглядати так:

```
  char choice;
  do
  {
        cout << «Виберіть дію» << '\ n';
        cout << «1. Введення даних «<< '\ n';
        cout << «2. Обробка «<< '\ n';
        cout << «3. ... «<< '\ n';
        cout << «4. Вихід «<< '\ n';

        cin >> choice;

        switch(choice)
        {
        ...
        }
```

```
} while(choice! = '4');
```

Зауважимо, що такий варіант не вимагає ініціалізації змінної *choice*. Ще один приклад використання циклу з передумовою пов'язаний з можливістю перевірки введених значень на валідність. Під валідністю розуміють коректне значення деякого параметра, відповідність значення змінної деякого діапазону можливих значень. Так в вираженні $y=1/x$ значення змінної $x=0$ буде невалідним, тому що при обчисленні виразу виникне помилка «поділ на 0», що викличе аварійне завершення програми.

За допомогою оператора циклу з передумовою можна забезпечити таку перевірку. Розглянемо приклад, коли в програмі потрібно ввести значення змінної в заданому діапазоні.

Наприклад. Нехай в додатку моделюється політ літака. Необхідно оперувати зі змінною, яка зберігає поточну висоту польоту. З точки зору здорового глузду висота польоту не може бути негативною. З іншого боку літак не може бути використаний для польотів в космос, тобто піднятися вище 50 км. Таким чином, значення змінної, яка описує поточну висоту об'єкта повинно бути в діапазоні від 0 до 50000 м. Виконаємо перевірку діапазону з використанням оператора *do..while*.

```
int height; // висота
...
do
{
  cout << «Введіть поточну висоту» << '\ n';
  cin >> height;
}
while (height <0 || height> 50000);
```

У прикладі, якщо введена висота, яка не відповідає заданому діапазону, то Вас попросять про нові введення даних і очікується введення нової висоти. Якщо введене значення висоти валідність, то програма продовжує своє виконання.

Приклад з висотою узятий з реального життя. Під час перевірки льотних характеристик винищувача *F*–16 в Ізраїлі американські льотчики виявили періодичні зависання бортового комп'ютера і аварії літака. При випробуваннях на території США подібних випадків не фіксувалося.

Виявилося, що випробування відбувалися в районі Мертвого моря, яке лежить нижче рівня світового океану. Бортовий комп'ютер при перетині нульової висоти видавав помилку і перезавантажувався.

Використання ітераційних циклів. Оператори циклів з післяумовою часто використовуються для реалізації ітераційних алгоритмів. Ітераційні алгоритми засновані, як і всі циклічні алгоритми, на повторюваних діях, але заздалегідь невідомо, яка кількість разів цикл буде виконано. Ітераційні дії використовуються, наприклад, в аналого–цифрових перетворювачів (АЦП) послідовного наближення.

Аналогово–цифровий перетворювач це пристрій, який дозволяє перетворити якийсь безперервний – аналоговий – фізичний сигнал в цифровий код. Такі пристрої широко застосовуються в системах автоматики. На рис. 4.7 представлена схема, яка містить АЦП

Рисунок 4.7 – Схема ітераційного циклу з АЦП

Інформація про параметри деякого об'єкту визначається датчиком. Це може бути положення об'єкта, його температура і т.д. На виході датчика є електричний сигнал, наприклад, сигнал напруги. Причому, зміни значення електричного сигналу, строго відповідають змінам параметра об'єкта. Далі електричний сигнал надходить на вхід АЦП, який формує на виході цифрове значення для вхідного електричного сигналу. Цифрове значення може бути збережено в комірці пам'яті ЕОМ і використано в якості даних для роботи програми.

Для АЦП однією з найважливіших характеристик є дозвіл. Ця характеристика визначає мінімальну зміну вхідного сигналу, яке буде перетворено. Дозвіл пов'язано з розрядністю аналого-цифрового перетворювача. Для 4-х розрядного АЦП весь діапазон вхідної напруги, 0..100 У наприклад, може бути перетворений в 24=16 значень вихідного цифрового коду. На рис. 4.8 приведена шкала такого АЦП. Тобто якщо вхідна напруга буде 100В, то на виході АЦП буде цифровий код 15. Якщо вхідна напруга дорівнює 50В, тобто середина діапазону, значення

вихідного коду лежить між 7 і 8, приймається кількість повних одиниць – код 7, відповідно при вхідній напрузі 25 В – код 3, і т.д.

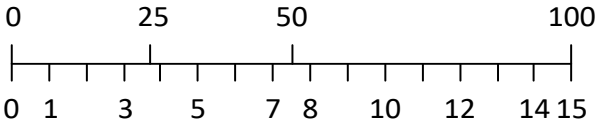


Рисунок 4.8 – Шкала АЦП

Дозвіл такого АЦП дорівнюватиме 100В / (16–1) = 6,67 В. Це означає, що для всіх значення вхідної напруги в діапазоні 0..6,67 У вихідний цифровий код буде 0, в діапазоні 6,67..13 , 34 В – код 1 і т.д. Така точність вимірювання може бути неприйнятною.

У АЦП послідовного наближення більш точне значення може бути отримано за рахунок декількох вимірювань. На першому кроці визначається значення з дозволом 6,67 В. Наприклад, отриманий код 4. Значить, вхідна напруга лежить в діапазоні 26,68..33,35 В (4х6,67..5х6,67).

На другому кроці отриманий діапазон ділитися на шістнадцять поділок і визначається код відповідний вхідній напрузі, але більш вузькому діапазоні, що дозволяє збільшити точність вимірюваної напруги. Дозвіл АЦП на другому кроці буде визначатися як 6,67 В/(16–1) = 0,44 В. Наприклад, на другому кроці отримано код 11. Значить вхідна напруга лежить в діапазоні 31,52..31,96 В (26,68 В + 0,44 Вх11..26,68 В + 0,44 Вх12). Якщо отримана точність влаштовує, то вимірювання припиняються. Якщо не влаштовує. Прикладом итерационного алгоритму може служити алгоритм обчислення суми елементів сходиться ряду. Послідовність $U_i(x)$ є такою, що сходиться, якщо $U_i(x) > U_{i+1}(x)$. При збільшенні і значення елемента $U_i(x) \to 0$. Необхідно знайти суму послідовності $S = \sum_{i=1}^{\infty} \frac{1}{i!}$.

Кількість доданків в такій послідовності невідомо. Зрозуміло, що кожне нове доданок менше попереднього. І на певному етапі нове доданок стає настільки мало, що не впливає на значення шуканої суми. Для обчислення такої суми можна використовувати циклічні дії. Умовою виходу з циклу має бути досягнення деякого мінімального значення чергового доданка.

На кожній ітерації необхідно обчислювати нове значення доданка. Значення k-го доданка одно $U_k = \frac{1}{k!}$, Значення $(k+1)$-го доданка одно $U_{k+1} = \frac{1}{(k+1)!}$. З іншого боку відомо, що $(k+1)! = k!(k+1)$ значить $U_{k+1} = U_k \frac{1}{k+1}$. Схема алгоритму представлена на рис. 4.9.

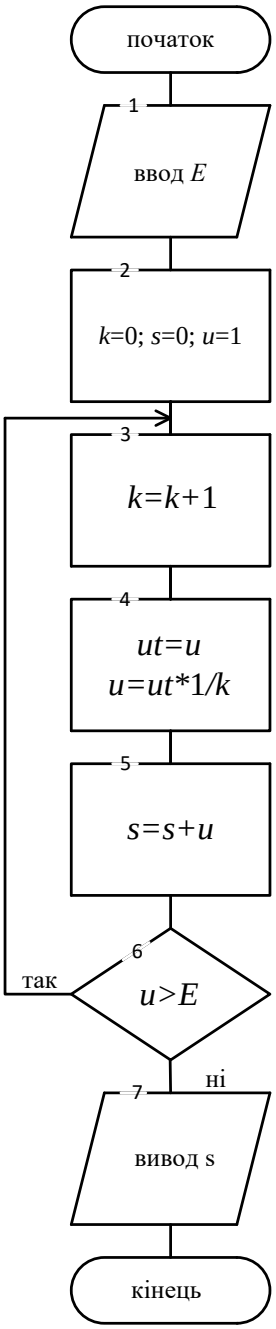


Рисунок 4.9 – Алгоритм обчислення суми елементів ряду, який сходиться

Далі слід програма, яка реалізує запропонований алгоритм.

```
#include <iostream.h>
void main()
{
int k = 0;
float s = 0, u = 1, ut;
double eps = .000001;

do
{
        k ++;
        ut = u;
        u= ut * 1/k;
        s + = u;
}
while (u> = eps);
cout << «Сума =« << s << '\ n';
}
```

У наступному прикладі необхідно скласти програму обчислення виразу y = 2 * *x* !.

Факторіал числа обчислюється як множення попереднього числа на наступне (збільшене на одиницю). Для цього виразу необхідно скористатися формулою $x! = 1 \cdot 2 \cdot 3 \ \ldots \cdot (x-1) \cdot x$. Таким чином, для обчислення факторіала числа *x* в програмі необхідно організувати цикл з початковими установками, рівними одиниці, а також умовою виходу з циклу, коли початкові установки досягнуто значення *x*. Тіло циклу – твір факторіала і збільшення на одиницю початкових установок.

Блок-схема алгоритму програми представлена на рис. 4.10.

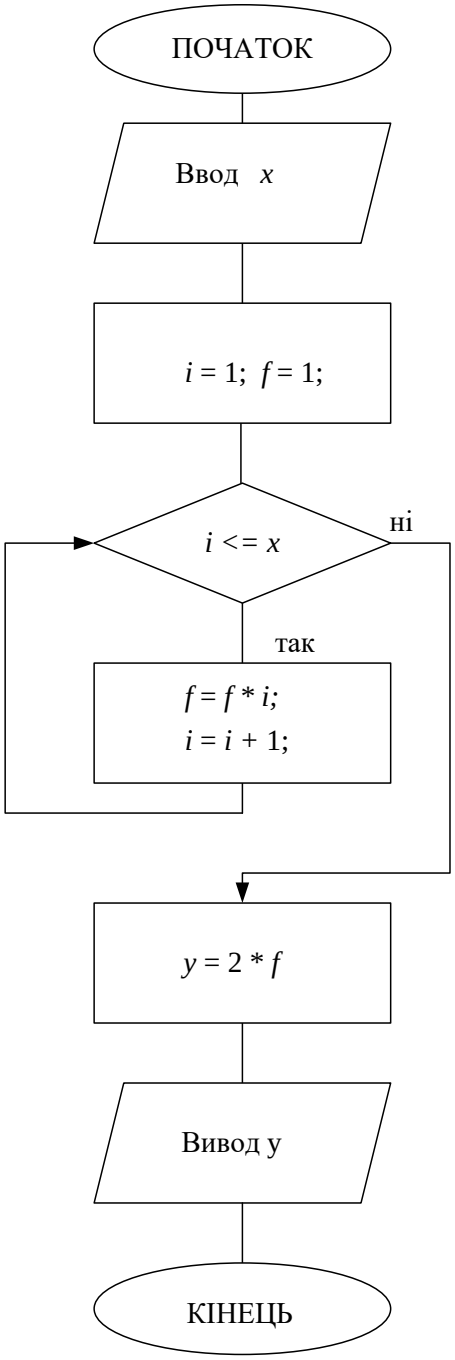


Рисунок 4.10 – Блок-схема алгоритму програми

```
# include <iostream>

int main()
{
    int x, i, f, y;
    cout << «Введіть ціле число x \ n»;
```

```
    cin >> x; // введення значення змінної x
    i =1;
    f =1;
    while (i <= x) // умова виконання циклу
    {
    f = f * i; // твір попереднього значення на наступне
    i = i +1; // збільшення на одиницю попереднього значення
    }
    y =2 * f;
    cout << «y =« << y << '\ n'; // вивід значення y на екран
    return 0;
}
```

Для налагодження програми використовуємо клавішу *F*7, переконуємося у відсутності помилок і запускаємо програму на виконання за допомогою комбінації клавіш *Ctrl+F*5. Нижче наведені результати роботи програми.

Введіть ціле число x = 5,

y = 240.

Цикл з параметром. Оператор дозволяє циклічно виконувати певну послідовність операторів. Умови виконання та закінчення циклу визначаються його описом. Схема алгоритму для параметричного циклу уявлення на рис. 4.11.

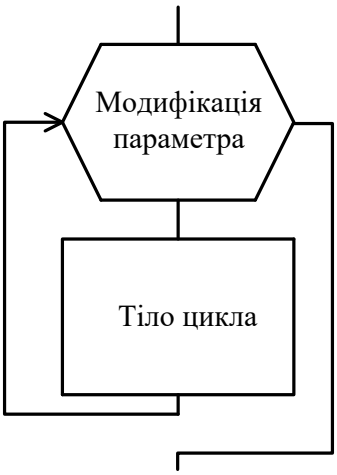


Рисунок 4.11 – Цикл з параметрами *for*

Параметром циклу називається змінна, яка містить значення, що визначає кількість виконаних циклів. Після кожного виконання тіла циклу значення параметра циклу збільшується або зменшується, тобто модифікується. Використовуючи параметр циклу можна відстежувати номер поточного виконання тіла циклу. Також параметр циклу часто використовується для перевірки умови закінчення циклічних дій. В *C/C++* цикл з параметром реалізований оператором for.

for (ініціалізація; умова; модифікація)
    оператор;

Синтаксис оператора містить 3 параметра. Параметр <*ініціалізація*> складається з одного або декількох описів і операторів, які виконуються один раз перед першим виконанням тіла циклу. Параметр <*умова*> є виразом логічного типу, яка визначає умова виконання циклічних дій. Якщо вираз в параметрі <*умова*> має значення true, то циклічні дії будуть повторюватися. Якщо вираз в параметрі *умова* має значення *false*, то виконання циклу припиняється. У разі, коли вираз в параметрі <*умова*> має тип відмінний від логічного, то його значення буде автоматично приведено до логічного типу. Параметр <*модифікація*> містить одне або кілька виразів, які обчислюються після завершення тіла циклу. Параметр <*оператор*> є тілом циклу.

Працює оператор наступним чином. Починається виконання оператора циклу з виконання операторів, які містяться в параметрі <*ініціалізація*>. Далі обчислюється значення виразу параметра <*умова*>. Якщо значення цього виразу одно *true*, то виконується оператор тіла циклу. Потім виконуються вирази, що входять в параметр <*модифікація*>. Після чого знову обчислюється значення виразу параметра умова і приймається рішення про подальше виконання циклічних дій. Таким чином, після кожного виконання тіла циклу виконується параметр <*модифікація*> і перевіряється параметр <*умова*>. Цикл закінчується, коли вираз в параметрі <*умова*> приймає значення *false*.

Часто в тілі циклу необхідно виконати декілька операторів. У цьому випадку використовується складений оператор.

for (ініціалізація; умова; модифікація)

```
{
  оператор1;
  оператор2;
  ...
  оператор n;
}
```

Найпоширенішим варіантом використання циклу з параметром є наступний. У прикладі необхідно обчислити довжину вектора в 20–вимірному просторі. Координати вектора вводяться з клавіатури. для вектора $\overline{X} = \{x_1, x_2, ..., x_{20}\}$ довжина дорівнюватиме $D = \sqrt{x_1 + x_2 + ... + x_{20}}$

```
#include <iostream>
#include <math.h>
void main()
{
int sum = 0, val;
for (int i = 0; i <20; ++ i)
{
        cin >> val;
        s + = val * val;
}
cout << «Сума =« << sqrt (sum) << '\ n';
}
```

Параметром циклу в прикладі є змінна *i*. При виконанні оператора циклу змінна приймає значення 0, після виконання кожного тіла циклу значення змінної инкрементируется. Умовою виконання циклу є вираз *i*<20. Таким чином, значення змінної буде змінюватися від 0 до 20. При значення 20 умова прийме помилкове значення і цикл завершиться. Виконуватися цикл буде при значеннях і від 0 до 19. Тіло циклу складається з двох операторів. Один оператор здійснює введення значень з клавіатури. Другий оператор виконує накопичення суми квадратів координат вектора.

У наступному прикладі параметри оператора циклу містять кілька виразів. Вирази розділені комами.

```
int i, j;
for (i= 0, j = 10; i! = j; ++ i, —j)
  оператор;
```

Цикл має два параметри – змінні і *i* і *j*. Початковими значеннями параметрів є значення 0 і 10 відповідно. Цикл закінчується при рівних значеннях параметрів. У параметрі модифікації також міститься два вирази – інкремент параметра *i* та декремент параметра *j*. Такий цикл виконається 5 разів при значеннях *i* від 0 до 4 і значеннях змінної *j* від 10 до 6. Потім значення параметрів стануть рівними 5 і цикл завершиться.

Слід обережно використовувати подібні алгоритми. Наприклад, внесемо такі зміни.

```
int i, j;
for (i= 0, j = 9; i! = j; ++ i, —j)
  оператор;
```

Можна помітити, що такий цикл буде виконуватися нескінченно – значення параметрів циклу ніколи не стануть рівними.

Параметри в операторі *for* можуть бути опущені. При цьому роздільники повинні зберігатися. Нескінченний цикл можна організувати наступним оператором

```
for (;;)
{
  ...
}
```

Також нескінченний цикл, який містить умову виходу можна реалізувати так. Наприклад, необхідно виконувати деякі циклічні дії до тих пір, поки користувач не ввів з клавіатури символ ‘0’.

```
char sym = ‘1’;
for (; sym! = ‘0’; )
{
  ...
```

```
  cin >> sym;
}
```

Вкладені цикли. Вкладеними називаються циклічні оператори, коли один з них є тілом циклу або частиною тіла циклу іншого. У кожній ітерації зовнішнього циклу виконуються всі ітерації внутрішнього циклу. Після завершення внутрішнього циклу виконується повернення управління у зовнішній. Розрахувати кількість виконань внутрішнього циклу можна так. Якщо внутрішній цикл виконується *N* раз, а зовнішній *M* раз, то тіло внутрішнього циклу буде виконано *M*N* раз.

Принцип роботи програми, який реалізує вкладений цикл, заснований на тому, що внутрішній цикл повністю виконується на кожному кроці зовнішнього циклу від початку і до кінця. Іншими словами, поки програма не вийде з вкладеного циклу – виконання зовні не продовжитися.

Наприклад: написати програму, яка виводить на екран таблицю множення. Блок-схема до даної програми представлена на рис. 4.12.

```
#include <iostream>
using namespace std;
int main ()
{
   int i, j, p;
   for (i = 1; i <10; i ++)
   {
       for (j= 1; j <10; j ++)
       {
              p = i * j;
              cout << p << «\ t»;
       }
   cout << '\ n';
   }
   return 0;
}
```

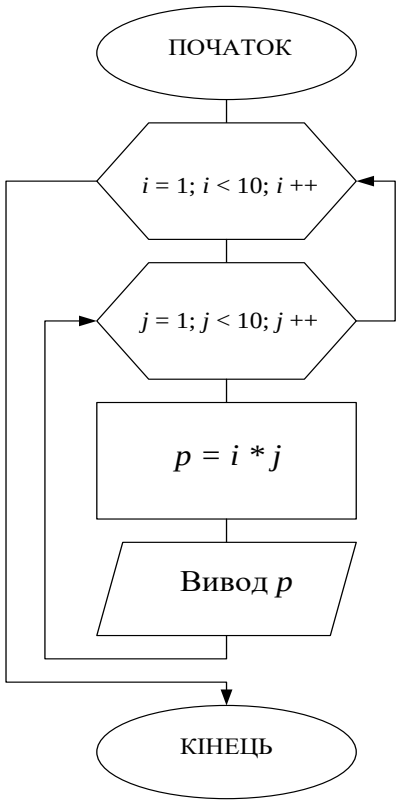


Рисунок 4. 12 – Блок-схема алгоритму програми

Коментар до програми.

1. Керуючі змінні зовнішнього і внутрішнього циклів (*i, j*) здійснюють функції множників.

2. Керуюча змінна *i* створюється та ініціалізується значенням 1.

3. Програма перевіряє умову *i*<10, так як 1 менше 10 то умова є *true* і програма входить у зовнішній цикл.

4. Керуюча змінна *j* створюється та ініціалізується значенням 1.

5. Програма перевіряє умова *j*<10, так як 1 менше 10 умова є *true* і програма входить у внутрішній цикл.

6. Здійснюється показ на екран результату *i**(*j* – 1).

7. Здійснюється зміна керуючої змінної j.

8. Знову перевіряється умова *j*<10, тому що 2 менше 10 умова є *true* і програма знову входить у внутрішній цикл.

9. Здійснюється показ на екран твори *i* на *j* – 2.

10. Здійснюється зміна керуючої змінної *j*.

Дія з 5 по 7 пункти повторюються до тих пір, поки не стає дорівнює 10, при цьому поточне значення *i* (1) множиться на кожне значення *j* (від 1 до 9 включно), результату показується на екран. Виходить рядок таблиці множення на 1. Потім програма виходить з внутрішнього циклу і переводить екранний курсор на два рядки вниз. Після цього, здійснюється збільшення змінної і на одиницю і знову вхід у внутрішній цикл. Тепер уже для виведення ланцюжка множення на 2. Таким чином, на екрані з'являється таблиця множення.

```
1 2 3 4 5 6 7 8 9
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

У наступному прикладі необхідно визначити площу поверхні води в озері по знімках, що одержуються з штучного супутника Землі. Таке завдання вирішується, наприклад, для проведення екологічного моніторингу, коли необхідно визначити, як змінюється площа озера в часі в зв'язку з промисловою діяльністю людини. Крім того необхідно мати координати ділянок водної поверхні, щоб відстежувати зміни форми озера в часі. Вихідними даними є фотографія ділянки земної поверхні. Контури водойми мають складну форму, тому знаходження його площі прямими методами є складним завданням. Зображення складається з окремих елементів – пікселів. Кожен піксель характеризується числом, яке відповідає його кольором. Для спрощення завдання можна вважати, що колір озера на знімку синій і, отже, можна підрахувати кількість загальна кількість синіх пікселів. Ця кількість буде пропорційно площі озера. Крім того, відомими є загальна кількість пікселів на зображенні, а також площа ділянки земної поверхні. Тоді площа озера можна обчислити, склавши пропорцію $\frac{S_{общ}}{M^2} = \frac{S_{оз}}{P_{син}}$.

тоді $S_{оз} = \frac{P_{син} S_{общ}}{M^2}$,

де $S_{заг}$ – загальна площа поверхні на знімку,

$S_{Оз}$ – площа поверхні озера,

M – загальна кількість пікселів на знімку,

$P_{сін}$ – кількість пікселів синього кольору на знімку.

Рішення завдання може бути реалізовано в такий спосіб:

```
#define SQ_COMMON100000000
#define M 10000
#define BLUE 0xFF0000UL
void main()
{
  ...
  long pixel;                  // колір пікселя зображення
  long sum_pix_blue =0; // загальне число синіх пікселів
  float sq_lake;
  for (long latitude = 1; latitude <M; latitude ++)
        for (long longitude = 1; longitude <M; longitude ++)
        {
              // отримання чергового пікселя зображення
              ...
              if (pixel == BLUE)
              {
                    sum_pix_blue ++;
        // перетворення latitude і longitude в географічні
        // координати і збереження їх для подальшого порівняння
                    ...
              }
        }
        sq_lake = (sum_pix_blue * SQ_COMMON) / (M * M);
  ...}
```

У фрагменті програми описано кілька констант. Константа *SQ_COMMON* містить значення загальної площі земної поверхні на знімку в м2. Вважаємо, що знімок квадратний і константа М має значення

дорівнює кількості пікселів зображення по горизонталі і вертикалі. Константа *BLUE* містить числове значення для визначення пікселя синього кольору. Далі описані кілька змінних. Мінлива *pixel* містить значення кольору поточного пікселя на зображенні. Мінлива *sum_pix_blue* використовується для підрахунку загального числа пікселів синього кольору. В змінної *sq_lake* зберігається значення обчисленої площі озера. У програмі реалізовані вкладені цикли для перебору пікселів зображення зовнішній цикл виконує перебір по широті, внутрішній цикл – виконує перебір по довготі місця. Якщо використовувати стандартну орієнтацію географічних зображень, коли північ розташований вгорі, то виконується перебір пікселів зображення по горизонталі в зовнішньому циклі і по вертикалі у внутрішньому. Це означає, що спочатку послідовно перебираються пікселі першого рядка зображення, потім другого рядка і т.д. У тілі внутрішнього циклу виконується визначення кольору поточного пікселя знімка. Це може бути виклик спеціального програмного модуля, який зв'язується із супутником і в реальному часі отримує поточний елемент зображення. Якщо колір отриманого пікселя є синім, то на одиницю збільшується значення змінної *sum_pix_blue*. Також в тілі умовного оператора виконується перетворення змінних *latitude* і *longitude* в географічні координати, і отримані координати зберігаються для подальшого порівняння.

4.4. Оператор завершення, продовження циклу

Оператор завершення використовується в операторах *while, do..while, for, switch*. Призначення оператора – припинення виконання зазначених вище операторів, тобто раніше, ніж це передбачено синтаксисом.

Оператор завершення задається ключовим словом *break*.

Розглянемо виконання оператора на прикладі рішення наступної задачі. Необхідно обчислити суму довільної кількості введених чисел, додаток припиняє виконання при введенні негативного числа.

```
#include <iostream>
void main()
{
```

```
int sum = 0, val;
for (int i = 0; i <100; ++ i)
        {
        cin >> val;
        if (val> = 0)
                s + = val;
        else
                break;
        }
cout << «Сума =« << sum << '\ n';
}
```

У прикладі змінна sum використовується для накопичення суми, змінна *val* служить для введення поточного значення послідовності, змінна і оголошена в параметричному циклі і є його лічильником. У тілі циклу здійснюється введення нового значення послідовності. Далі виконується перевірка, чи є введене число позитивним. Якщо умова виконується, то відбувається накопичення часткової суми послідовності. Якщо ж було введено від'ємне значення, то умова оператора *if* стає хибним, виконується оператор *break* і виконання циклу закінчується.

Оператор продовження. Оператор продовження використовується в операторах *while, do..while, for*. Призначення оператора – припинення виконання поточної ітерації циклу і перехід на перевірку умови завершення циклічних дій.

Оператор продовження задається ключовим словом *continue*.

Розглянемо такий приклад. Обчислимо суму невід'ємних чисел, що вводяться з клавіатури, причому введених чисел повинно бути 10.

```
#include <iostream.h>
void main()
{
int sum = 0, val;
for(int i =0; i <10; ++ i)
        {
        cin >> val;
        if(val <0)
```

```
            continue;
        sum + = val;
        }
cout << «Сума =« << sum << '\ n';
}
```

Якщо виконується умова *val*<0, то виконується оператор continue. При цьому пропускається в тілі циклу всі оператори, які йдуть за оператором continue. Відбувається перехід до заголовку циклу, змінюється значення параметра циклу, перевіряється умова виконання і т.д.

***Оператор безумовного переходу*.** Оператор безумовного переходу використовується, коли необхідно здійснити перехід з однієї частини програми в іншу. Адресою переходу, який вказує, який оператор буде виконаний наступним, є параметр мітка. Формат цього оператора безумовного переходу наступний:

```
goto ідентифікатор;
...
ідентифікатор: оператор;
```

Гнучкість конструкцій мови *C/C++* дозволяє без оператора безумовного переходу і використовувати цей оператор не рекомендується. Його застосування порушує ієрархічну структуру програми і призводить до створення нечитабельною, складного для розуміння коду.

Виправданим є використання даного оператора в деяких випадках. Наприклад, якщо необхідно здійснити вихід з вкладених циклів. Оператор *break* в цьому випадку непридатний, тому що він припиняє тільки той цикл, в якому викликаний.

Ще один варіант, коли можна використовувати оператор переходу це обробка помилок. Розглянемо такий код:

```
int error = 0;
// обробка даних, де може виникнути помилка
...
if(error! =0)
        goto exit
...
```

exit:

...

У прикладі змінна error зберігає код помилки. Якщо значення цієї змінної дорівнює 0, то операції обробки даних прийшли успішно. Якщо виникли якісь помилки, то значення змінної error буде змінено на нульове. У цей випадку подальше виконання програми втрачає сенс і наступні оператори потрібно пропустити. У прикладі це досягається переходом на мітку *exit*.

Практичні завдання

1. Знайти максимальну цифру в запису тризначного числа. Визначити, чи є дана цифра парним числом.

2. Ввести ціле число. Визначити приналежність числа інтервалах [–100..0]; [23..90]; [145..158]. Якщо число не належить необхідним інтервалах – вивести повідомлення. Якщо число потрапляє в інтервал [23..90], то знайти залишок від ділення введеного числа і 38.

3. Написати програму, яка обчислює функцію $y(x) = 10x!$. Результат вивести на екран.

4. Підрахувати кількість цифр в десяткового запису цілого невід'ємного числа *n*,. введеного з клавіатури.

5. Дано довільні числа *a, b, c*. Якщо не можна побудувати трикутник з такими довжинами сторін, то надрукувати 0, інакше надрукувати 3, 2 або 1 в залежності від того, рівносторонній цей трикутник, рівнобедрений або будь-якої іншої.

6. Скласти програму визначальну, чи є ціле число *n* введене з клавіатури – простим.

7. Скласти програму, яка виводить перші *n* чисел Фібоначчі.

8. Визначити, чи потрапляє точка з координатами *x, y* в коло радіусом *r*. Якщо не потрапляє, то обчислити радіус кола, в яку вона потрапляє.

9. Написати програму, яка обчислює суму ряду $y = \cos x^2 + \cos x^3 + \cos x^4 + ... + \cos x^{20}$. Результат вивести на екран.

10. Обчислити кількість точок з цілочисельними координатами, що потрапляють в коло радіусом R ($R > 0$) з центром на початку координат.

11. Надрукувати в порядку зростання всі числа від 100 до 999 в десяткового запису, яких немає однакових цифр.

12. Визначити кількість тризначних цілих чисел, сума цифр яких дорівнює *n*. Результат вивести на екран.

13. Визначити, чи є задане натуральне число паліндром, тобто таким, яке читається однаково зліва направо і справа наліво.

14. Скласти програму знаходження твори всіх дільників цілого числа n, що вводиться з клавіатури. Результат вивести на екран.

15. Скласти програму визначення, чи є, введене з клавіатури ціле число n парних двозначним числом.

16. Скласти програму, яка перевіряє, ділитися задане тризначне число на кожну зі своїх цифр.

17. Скласти програму пошуку всіх простих чисел в проміжку [*n*1; *n*2], де *n*1 і *n*2 – два цілих числа введених з клавіатури.

18. Введіть послідовність цілих чисел, поки не введете 0. Скласти програму знаходження середнього арифметичного чисел кратних 3.

19. З клавіатури вводиться ціле число *n*. Визначити скільки разів в ньому зустрічається цифра *a*.

20. Надрукувати в порядку зростання всі числа від 100 до 999 в десяткового запису яких немає однакових цифр.

21. Якщо остання цифра тризначного числа 2, то знайти різницю чисел даного числа, інакше - надрукувати назву першої цифри тризначного числа.

22. Обчислити $y=1+x/x_2$ для будь-якого значення *x*. Округлити отримане значення до найближчого цілого. Надрукувати назву старшої цифри отриманого числа.

23. Визначити кількість понеділків в році, що припадають на 13-е число, вважаючи, що рік не високосний і 1 січня припадає на понеділок.

24. Дано 3 десяткові цифри. Ввести на друк назву меншою з цих цифр і надрукувати її назву.

25. Якщо перша цифра чотиризначного числа 1, то знайти суму цифр даного числа, якщо 2 – знайти різницю двох останніх цифр числа, якщо перша цифра 8, то знайти твір двох останніх цифр числа, інакше, вивести на друк назву першої цифри числа.

26. Знайти максимальну цифру в запису тризначного числа і вивести на друк її назву.

27. Вивести на друк найменшу цифру тризначного числа і надрукувати її назву. Визначити, чи є отримана цифра парним числом.

28. Для натурального числа до надрукувати фразу «ми знайшли до грибів в лісі», погодивши закінчення слова «гриб» з числом *k*.

29. Написати програму, яка запитує у користувача номер місяця народження, потім виводить назву місяця або повідомлення про помилку.

30. Скласти програму, яка для будь-якого натурального n меншого 100 дає найменування «рік», «роки», «років» вважаючи, що n вік людини.

31. Знайти мінімальну цифру тризначного числа. Вивести на екран її назву. Визначити парність цього числа.

32. Якщо остання цифра тризначного числа 5, то знайти суму цифр даного числа, якщо 3 - різниця цифр даного числа, інакше – надрукувати назву другої цифри тризначного числа.

33. Якщо перша цифра чотиризначного числа 2, то знайти різницю двох останніх цифр числа, якщо перша цифра більше 2 і менше 8, то знайти твір двох останніх цифр числа, інакше, вивести на друк назву першої цифри числа.

34. Для вводиться з клавіатури цілого числа *n* надрукувати фразу «мені *n* років», з огляду на при цьому, що при деяких значеннях n слово «років» треба замінити на слово «року» або «рік».

35. Ввести з клавіатури ціле чотиризначне число. Вивести на екран назву другої цифри числа.

36. Написати програму, яка виводить на екран назву максимальної цифри тризначного числа, введеного з клавіатури.

37. Обчислити $y=(4+x)/x_2$ для будь-якого значення *x*. Округлити отримане значення до найближчого цілого. Надрукувати назву старшої цифри отриманого числа.

38. За датою визначити день тижня, вважаючи, що рік не високосний і 1 січня припадає на понеділок.

39. Підрахувати кількість цифр в десяткового запису цілого невід'ємного числа *n* і вивести результат на екран.

40. Визначити, чи є задане натуральне число n паліндромом, тобто таким, яке читається однаково зліва направо і справа наліво.

41. Визначити число, отримане виписуванням в зворотному порядку цифр заданого натурального числа *n*. Результат вивести на екран.

42. Скласти програму знаходження суми всіх дільників цілого числа n, що вводиться з клавіатури. Результат вивести на екран.

43. Скласти програму знаходження твори всіх дільників цілого числа *n*, що вводиться з клавіатури. Результат вивести на екран.

44. Скласти програму знаходження середнього арифметичного всіх дільників числа цілого числа *n*, що вводиться з клавіатури. Результат вивести на екран.

45. Скласти програму обчислення суми квадратів непарних чисел від a до *b*, де a і *b* – цілі числа, введені з клавіатури.

46. Скласти програму обчислення виразу *y*=(3+*n*)! Результат вивести на екран.

47. Скласти програму знаходження найменшого спільного кратного (НОК) двох натуральних чисел.

48. Введіть послідовність дійсних чисел, поки не введете 0. Скласти програму визначення суми введених чисел.

49. Скласти програму, що визначає, всі цілі числа з проміжку від 100 до 300, у яких сума дільників дорівнює 50. Результат вивести на екран.

50. Скласти програму пошуку перших 20 натуральних чисел, що діляться без остачі на 13 або 17 і великих 500. Результат вивести на екран.

51. Скласти програму, що визначає, кількість точок з цілочисельними координатами, що належать колу радіуса *R* з центром на початку координат.

52. Скласти програму пошуку кількості «щасливих» квитків, у яких сума перших трьох цифр і останніх трьох чисел рівні.

53. Скласти програму пошуку кількості тризначних натуральних чисел, сума квадратів яких дорівнює заданому цілому числу *n*. Результат вивести на екран.

54. З клавіатури вводиться ціле число *n*. Скласти програму, яка виводить на екран цифри даного числа *n*.

55. Обчислити найбільший спільний дільник двох цілих чисел, введених з клавіатури. Результат вивести на екран.

56. Підрахувати кількість цифр в десяткового запису цілого невід'ємного числа n, введеного з клавіатури.

57. Скласти програму пошуку перших n парних чисел (значення зміною n вводиться з клавіатури) в послідовності чисел Фібоначчі.

58. Скласти програму, яка виводить перші *n* чисел Фібоначчі.

59. Написати програму, яка обчислює суму ряду

$y = 1! + 2! + 3! + ... + n!$, ($n > 1$). Результат вивести на екран.

60. З клавіатури вводиться число Фібоначчі. Скласти програму, яка визначить його порядковий номер у послідовності Фібоначчі

61. Написати програму пошуку всіх натуральних чисел менше 100, які при зведенні в квадрат дають паліндром.

62. Написати програму пошуку максимального натурального числа з інтервалу від *a* до *b* з максимальною сумою дільників.

64. Якщо до суми цифр двозначного числа додати квадрат різниці цифр, то вийде те ж саме число. Скласти програму пошуку всіх таких чисел.

Контрольні питання

1. Оператори управління в *C/C++*. Призначення. Види операторів управління.

2. Циклічні алгоритми. Призначення. Приклади використання.

3. Умовний оператор і умовна операція. Призначення, синтаксис, принцип роботи.

4. Оператори циклу. Типи операторів циклу. Приклади.

5. Оператор–перемикач. Призначення, синтаксис і принцип роботи.

6. Цикл з передумовою. Призначення, синтаксис і принцип роботи.

7. Цикл з умовою поста. Призначення, синтаксис, принцип роботи.

8. Цикл з параметром. Призначення, синтаксис, принцип роботи.

9. Оператор завершення циклу. Синтаксис і принцип роботи.

10. Алгоритми знаходження складних сум.

11. Оператор продовження циклу. Синтаксис, принцип роботи.

12. Автоматичне перетворення типів у виразі.

13. Алгоритми знаходження екстремумів. Призначення, приклади використання.

14. Основні типи блоків в блок–схемах алгоритмів. Правила складання блок–схем алгоритмів.

15. Оператор безумовного переходу. Призначення і принцип роботи.

РОЗДІЛ 5.

МАСИВИ ЗМІННИХ І ВКАЗІНИКІВ. СОРТУВАННЯ МАСИВУ

5.1. Масиви змінних як однорідні статичні структури даних

У попередніх розділах було представлено такий механізм роботи з даними в програмі як використання змінних. Опис змінних в програмі виконується за певними правилами і служить для завдання ряду характеристик – тип змінної, перелік значень, які може приймати змінна, набір операцій для роботи з такою змінною, розмір пам'яті, який відводиться для зберігання значення змінної. Але всі змінні, розглянуті раніше в прикладах, були простими, тобто кожна така змінна зберігала тільки одне значення – число, машинне представлення символу і т.д. При розробці програмного забезпечення часто доводиться мати справу з необхідністю представляти одноманітну інформацію у вигляді впорядкованої структури значного розміру. Це можуть бути таблиці, вектора, рядки символів і т.д. Буде неоптимальним використовувати прості змінні для роботи з такого роду даними. Найбільш простим виходом в даній ситуації є застосування однорідних структур заданого розміру. Тобто структура являє собою набір простих змінних однакового типу, задано кількість таких змінних, тобто розмір структури і кожен елемент структури має власний номер званий індексом. Додатковими перевагами використання таких структур є їх природне відображення на лінійну архітектуру пам'яті обчислювальної машини, а також спрощення реалізації рішення деяких обчислювальних алгоритмів. задано кількість таких змінних, тобто розмір структури і кожен елемент структури має власний номер званий індексом. Додатковими перевагами використання таких структур є їх природне відображення на лінійну архітектуру пам'яті обчислювальної машини, а також спрощення реалізації рішення деяких обчислювальних алгоритмів. задано кількість таких змінних, тобто розмір структури і кожен елемент структури має власний номер званий індексом. Додатковими перевагами використання таких структур є їх природне відображення на лінійну архітектуру пам'яті обчислювальної машини, а також спрощення реалізації рішення деяких обчислювальних алгоритмів.

Структурним аналогом математичних векторів в мові *C* (як і в більшості інших мов програмування) є масиви змінних, тобто впорядковані послідовності елементів даних одного типу. З кожним таким масивом зв'язується його ім'я, тип елементів (а отже і обсяг ОЗУ, необхідної для розміщення одного елемента) і їх кількість (тобто довжина масиву).

Масив – це пронумерована послідовність величин однакового типу, що позначається одним ім'ям. Масив даних в програмі розглядається як змінна структурованого типу. Масиву присвоюється імя_массіва, за допомогою якого можна посилатися як на масив даних в цілому, так і на будь–яку з його компонент (рис. 5.1). При цьому кожна змінна в масиві є самостійною одиницею під назвою – елемент_массіва. Кожен елемент має свій порядковий номер – індекс. За індексом можна звертатися до конкретного елементу масиву. Індекс в позначенні елемента масиву може бути константою, змінною або виразом порядкового типу (цілочисельний, що перераховується діапазон). Кількість елементів в масиві визначає розмірність масиву. За цією ознакою масиви поділяються на одномірні (лінійні), двовимірні, тривимірні і т.д.

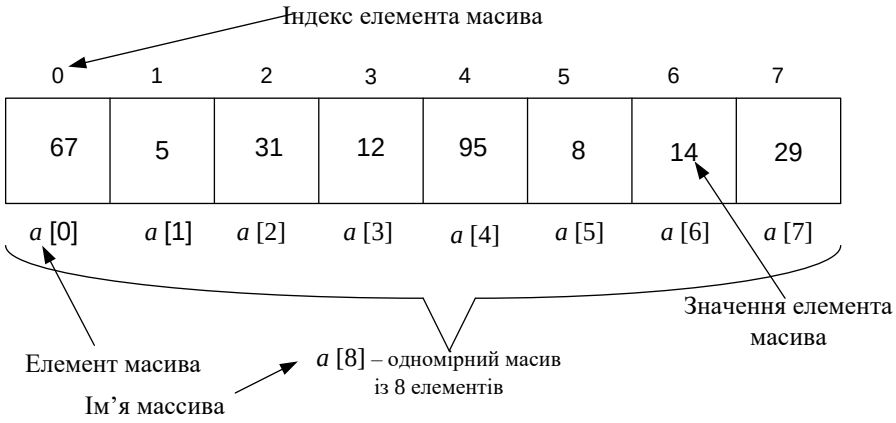


Рисунок 5.1 – Схема одновимірного масиву з восьми елементів

Для опису масивів в *C*-програмах використовуються інструкції, подібні до тих, які були введені для опису простих змінних. Також при описі обов'язково повинен задаватися розмір масиву.

тип ідентифікатор [розмірність];

Наприклад:

int marks[5];

Такий масив може використовуватися для подання сесійних оцінок студента і в ньому можна зберігати оцінки п'яти іспитів, кожен елемент масиву представлений змінною цілого типу. Слід зауважити, що розмірність масиву невелика, і оцінки можна зберігати в п'ять простих змінних. У прикладі:

float u_out[100];

описаний масив з 100 дійсних змінних, кожна з яких містить значення напруги на виході деякого пристрою, отримані при випробуваннях цього пристрою. Створення для кожного значення вимірювання власної простої змінної було б незручно у використанні. Доступ до конкретного елементу масиву здійснюється через його номер – індекс. Синтаксис звернення наступний:

ідентифікатор[індекс]

Слід звернути увагу, що елементи масиву мають індекси з 0 до розмір –1. Тобто

cout << «Початковий елемент масиву << u_out[0] *<< '\ n';*
cout << «Кінцевий елемент масиву << u_out[99] *<< '\ n';*

Таким же чином використовуються елементи масиву в виразах. Наприклад, описаний деякий довільний вектор в двомірному просторі, необхідно унормувати цей вектор. Нормування вектора – це операція приведення довжини вектора до 1 (рис. 5.2).

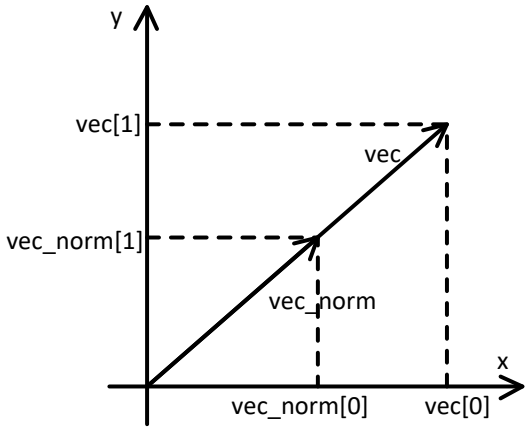


Рисунок 5.2 – Приведення довжини вектора до 1

```
float vec[2];          // вихідний вектор
float vec_norm[2];     // нормований вектор

cin >> vec[0] >> vec [1];

vec_norm[0] = vec [0] / (sqrt (pow (vec [0], 2) + pow (vec [1], 2));
vec_norm[1] = vec [1] / (sqrt (pow (vec [0], 2) + pow (vec [1], 2));
```

Можна помітити, що оператори присвоювання в прикладі відрізняються незначно. Як вже вище було сказано, масиви зручно використовувати при реалізації деяких видів обчислювальних алгоритмів – наприклад в циклічних алгоритмах.

Ініціалізація масивів. Раніше було вказано, що будь–який масив є структурованою змінної. Той факт, що масив є змінною, дозволяє задавати значення елементів масиву при його описі. Як і для звичайної змінної таку ініціалізацію масиву можна виконати за допомогою ініціаторів. Синтаксис такого опису наступний

тип імя_массіва[розмірність] = {ініціатор0, ініціатор1, ...};

Параметр розмір задає кількість елементів описуваного масиву. Значення параметра ініціатор0 буде присвоєно елементу масиву з індексом 0, значення параметра ініціатор1 буде присвоєно елементу масиву з індексом 1 і т.д. Типи ініціаторів повинні відповідати типу масиву.

Кількість ініціаторів має відповідати параметру розмір. Якщо їх кількість більше ніж розмір масиву – це викличе помилку. Якщо кількість ініціаторів менше розміру масиву, то значення будуть присвоєні тільки першим елементам масиву.

У прикладі нижче описаний масив, в якому задано кількість студентів в групах факультету

int group[12] = {19, 17, 16, 20, 19, 18, 21, 18, 18, 20, 16, 21};

Такий опис означає, що буде створена змінна – масив з 12 цілих елементів і елементу *group* [0] буде присвоєно значення 19, елементу *group*[1] буде присвоєно значення 17 і т.д.

У наступному прикладі ініціаторів менше, ніж елементів в масиві. Робота з рядками як з масивом типу char зустрічається досить часто.

char surname[30] = { ‘I’, ‘B’, ‘A’, ‘H’, ‘O’, ‘B’, ‘\0’};

Мінлива surname зберігає значення деякої прізвища. При створенні змінної резервується пам'ять для зберігання 30 елементів типу *char*. Зроблено це щоб у змінній можна було зберігати довільну прізвище як рядок довжиною до 29 символів. Останнім елементом рядка обов'язково повинен бути символ з *ASCII* кодом 0 – ‘\0’. На практиці прізвище може містити меншу кількість символів, ніж зарезервовано. Значущими є тільки перші символи масиву до символу ‘\0’. Такий підхід називається надлишковим резервуванням пам'яті для зберігання значення змінної. У представленому прикладі елемент масиву *surname*[0] прийме значення ‘'I’', елемент surname [1] прийме значення ‘B’ і т.д. Останнім елементом масиву, для якого буде встановлено значення, буде surname [6] і його значення буде рівним ‘\0’. Для елементів масиву surname [7] ..

У разі якщо кількість ініціаторів і розмір масиву збігаються параметр, що задає розмір масиву, можна опускати.

double sinus[] = {0, 0.5, 0.87, 1, 0.87, 0.5, 0, –0.5, –0.87, –1, –0.87, –0.5};

В даному прикладі задані значення функції sin(*x*) для кутів, кратних 300. Це опис еквівалентно опису

double sinus[12] = {0, 0.5, 0.87, 1, 0.87, 0.5, 0, –0.5, –0.87, –1, –0.87, –0.5};

Ініціалізація елементів масиву за допомогою оператора присвоювання

імя_массіва [індекс] = значення;

int a [5];
a[0] = 12; // привласнення значення 12–му елементу масиву з індексом 0
a[1] = 0; // привласнення значення 0–му елементу масиву з індексом 1
a[2] = 36; // привласнення значення 36–му елементу масиву з індексом 2
a[3] = 59; // привласнення значення 59–му елементу масиву з індексом 3
a[4] = 7; // привласнення значення 7–му елементу масиву з індексом 4
cout << *a* [2]; // вивід на екран значення елемента з індексом 2.

Ініціалізація елементів масиву за допомогою введення значень з клавіатури.

cin >> *імя_массіва*[індекс];

Приклад.
int a [5];
cin>>*a*[0]; // *введення з клавіатури значення елемента масиву з індексом* 0
cin>>*a*[1]; // *введення з клавіатури значення елемента масиву з індексом* 1
cin>>*a*[2]; // *введення з клавіатури значення елемента масиву з індексом* 2
cin>>*a*[3]; // *введення з клавіатури значення елемента масиву з індексом* 3
cin>>*a*[4]; // *введення з клавіатури значення елемента масиву з індексом* 4

Введення одновимірного масиву з клавіатури за допомогою циклу *for*

int a[10];
for (*i* =0; *i* <10; *i* ++)
cin>> *a* [*i*];

В даному циклі змінна *i* є індексом елемента масиву *a*. Значення, прийняте змінної *i* при першому проходженні циклу, дорівнює 0, так як

індекс першого елемента 0, вихід з циклу здійснюється при досягненні змінної і значення 9, так як останній індекс елемента масиву дорівнює 9.

Ініціалізація елементів масиву за допомогою генератора випадкових чисел. Для використання в програмі функції генератора випадкових чисел необхідно підключити спеціальну бібліотеку:

```
# include <stdlib.h>
```

Функція, яка генерує послідовність псевдовипадкових цілих чисел в діапазоні від 0 до *RAND_MAX*. Значення *RAND_MAX* дорівнює 32 767.

```
int rand ();
```

Приклад.

```
int x;
x = rand();
```

Змінної *x* цілого типу буде присвоєно будь-яке ціле число з діапазону від 0 до 32 767.

Завдання значень елементів одновимірного масиву за допомогою генератора випадкових чисел:

```
int a[10];
for (i = 0; i <10; i ++)
        a[i] = rand ()% 250;
```

Елементам одновимірного масиву а будуть задані значення цілих чисел з діапазону від 0 до 250 випадковим чином.

Для того щоб підвищити універсальність програми, розмір масиву краще вказувати через константу.

Приклад.

```
# include <iostream>
using namespace std;
const int n =5; // оголошення константи
int main ()
```

```
{
        int a[n], i;
        for (i =0; i <n; i ++)
                a[i] =–30 + rand ()% 131;
}
```

Розглянемо приклад знаходження суми елементів деякої послідовності. Необхідно знайти суму 10 дійсних чисел $S = \sum_{i=1}^{10} a_i$.

Елементи послідовності зручно представити як масив дійсних чисел. Схема алгоритму представлена на рис. 5.3.

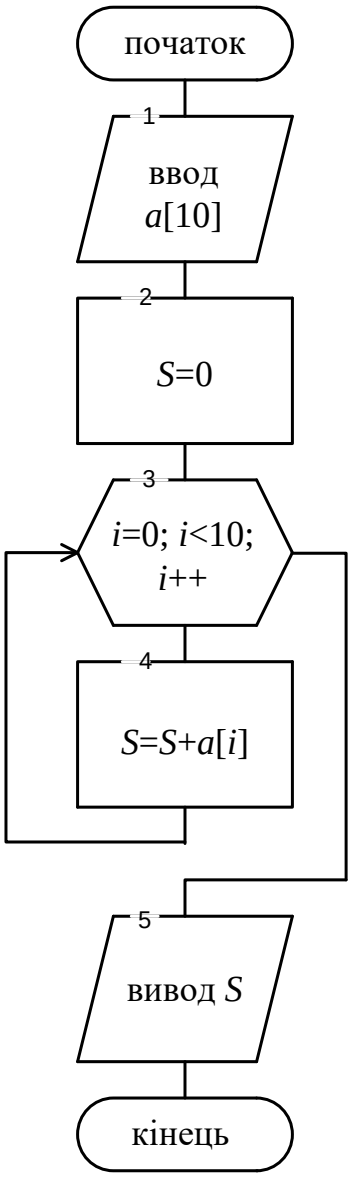


Рисунок 5.3 – Блок–схема алгоритму програми

В алгоритмі виконуються циклічні операції – додавання чергового елемента послідовності суму попередніх елементів. Слід звернути увагу, що індекс в такому алгоритмі теж задається як значення деякого виразу. Таким чином, виконується модифікація індексу після кожного оператора присвоювання і забезпечується доступ до наступного елементу послідовності. Початковий індекс матиме значення 0, вихід з циклу буде здійснений, коли значення змінної і буде менше 10, тобто для цілочисельних значень це – 9. Отже, останнім елементом масиву, для якого виконається операція додавання буде *a*[9].

Програма, реалізована у цій алгоритму, буде наступною

```
#include <iostream>
void main()
{
  float S = 0.f, a[10];
  ...
  for(int i = 0; i <10; ++ i)
          S + = a[i];
}
```

5.2. Багатовимірні масиви

У попередньому розділі були розглянуті масиви змінних, що мають один індекс. Для реалізації більш складних структур даних, таких як таблиці або матриці зручно використовувати елементи масиву з великою кількістю індексів. Так для опису елементів матриці один індекс буде задавати номер стовпця елемента, а другий – номер рядка елемента. Опис багатовимірного масиву наступне

тип ідентифікатор[розмірність1] [розмірність2] [розмірність3] ...;

Багатовимірний масив подібно одномерному масиву може бути ініціалізованим першим за допомогою списку ініціаторів. Першими инициализируются елементи з найменшими індексами:

int array_2d [3] [3] = {0, 1, 2, 3, 4, 5, 6, 7, 8};

Початкові значення отримують такі елементи масиву:

array_2d[0] [0] == 0
array_2d[0] [1] == 1
array_2d[0] [2] == 2
array_2d[1] [0] == 3
array_2d[1] [1] == 4
array_2d[1] [2] == 5
array_2d[2] [0] == 6
array_2d[2] [1] == 7
array_2d[2] [2] == 8

Додаткові фігурні дужки в ініціатора дозволяють форматувати окремі фрагменти багатовимірного масиву. Кожна пара фігурних дужок специфицирует значення, що відносяться до однієї розмірності. Це дозволяє більш наочно задавати ініціатори. Порожні фігурні дужки не допускаються.

```
int array_2d [3] [4] = {
                        {0, 1},
                        {200, 210, 300, 400},
                        {1000}
                        };
```

В результаті виконання такого опису будуть задані наступні елементи масиву *MyArra*y:

array_2d[0] [0] == 0
array_2d[0] [1] == 1
array_2d[0] [2] не заданий
array_2d[0] [3] не заданий
array_2d[1] [0] == 200
array_2d[1] [1] == 210
array_2d[1] [2] == 300
array_2d[1] [3] == 400
array_2d[2] [0] == 1000

array_2d[2] [1] не заданий
array_2d[2] [2] не заданий
array_2d[2] [3] не заданий

За аналогією з одновимірним масивом, при явній ініціалізації масиву входить до складу багатовимірного масиву його найлівіша розмірність може не вказуватися. Вона визначається на основі ініціатора.

```
int array_2d [] [4] = {
                    {0, 1, 2, 3},
                    {200, 210, 220, 230},
                    {1000, 1100, 1200, 1300}
                    };
```

У прикладі розглянемо створення т.зв. одиничної матриці – матриці, у якій елементи на головній діагоналі рівні 1, а всі інші елементи рівні 0.

$$\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

```
#include <iostream>
#define M 4
void main()
{
  float matrix [M] [M];
  ...
  for (int i = 0; i <M; ++ i)
          for (int j =0; j <M; ++j)
          {
                  if (i == j)
                          matrix [i][j] = 1;
                  else
                          matrix [i][j] =0;
```

} ...}

Масив має розмірність 2. Всього елементів в масиві 20. У прикладі організований вкладений цикл по рядках і стовпцях матриці. Збіг індексів *i* та *j* означає, що елемент матриці знаходиться на головній діагоналі і цьому елементу присвоюється значення 1.

Розглянемо приклад тривимірного масиву. Нехай є комп'ютерна гра. Поле гри являє собою двовимірну майданчик. Майданчик розбита на клітинки. Кожна клітинка може бути порожньою або містити один з об'єктів: гравець, рис (противник), дерево, будинок. Крім того, гра може містити кілька рівнів (етапів), тому на поле є осередки, в яких розташовані переходи на верхній або на нижній рівні (рис 5.4).

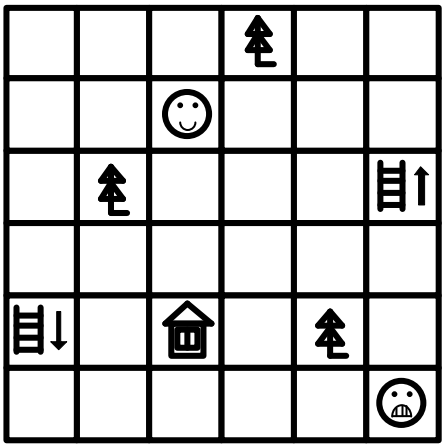


Рисунок 5.4 – Поле комп'ютерної гри

За допомогою багатовимірного масиву можна описати ігрове поле.

```
#define EMPTY       0
#define PLAYER      1
#define IMP         2
#define TREE        3
#define HOME        4
#define LEV_UP      5
#define LEV_DOWN    6
....
#define SIZE        10
#define LEVELS      8
```

int battlefield [SIZE] [SIZE] [LEVELS];

Ігрове поле рівня задається визначенням *SIZE* і має розмір 10x10. Всього рівнів 8, що описано константою *LEVELS*. Крім того, задані константи вмісту осередків: якщо в осередку знаходиться гравець (*PLAYER*), то значення елемента масиву *battlefield* дорівнює 1, тоді як осередку зростає дерево (*TREE*), то значення елемента масиву *battlefield* дорівнює 3, і т.д. Таким чином, описується весь ігровий простір. Логіка роботи може виглядати наступним чином

```
// якщо гравець зробив черговий крок і потрапив в клітинку з деревом,
// то дерево зникає, збільшується кількість ресурсів гравця і
// гравець займає цей осередок
if (battlefield[2] [1] [4] == TREE)
{
  ++resources;
  battlefield[2] [1] [4] = PLAYER;
}
// якщо гравець зробив черговий крок і потрапив в клітинку з чортом,
// то гра закінчується
if (battlefield[4] [3] [7] == IMP)
{
  // закінчення гри
  ...}
```

5.3. Алгоритми пошуку та сортування

Масиви використовуються для реалізації цілої групи алгоритмів, пов'язаних з пошуком і сортуванням інформації в послідовності.

Розглянемо алгоритм, який дозволяє виконати пошук максимального (мінімального) значення в матриці. Оброблювану матрицю необхідно представити у вигляді двовимірного масиву. У представленій задачі необхідно знайти значення максимального та мінімального елементів в масиві, а також індекси цих елементів в масиві. Схема алгоритму представлена на рис. 5.5.

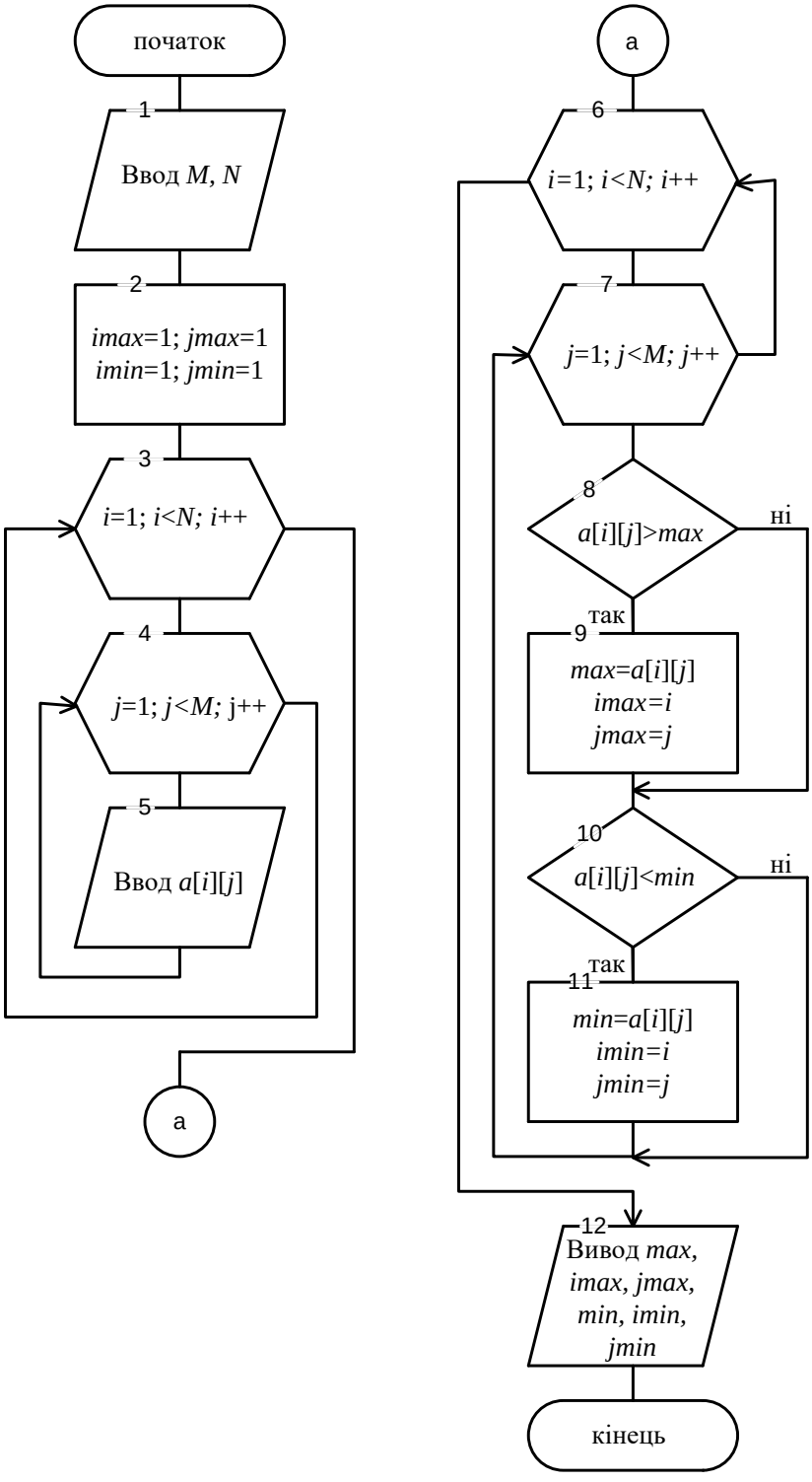


Рисунок 5.5 – Алгоритм програми перебування максимального і мінімального елементів двовимірного масиву

У блоці 1 схеми алгоритму виконується введення даних матриці. Важливою особливістю алгоритму є припущення, що верхній лівий елемент матриці є максимальним (мінімальним). Тому в блоці 2 значення

цього елемента присвоюється змінної *max (min)*, а змінні *imax* та *jmax (imin і jmin)*, які використовуються для зберігання індексів максимального (мінімального) елемента, приймають значення 1. За результатами роботи алгоритму змінна *max(min)* буде містити значення максимального (мінімального) елемента матриці, а змінні *imax* та *jmax (imin і jmin)* будуть містити індекси цього елемента. Далі в схемі алгоритму організовані цикли по рядках і стовпцях для перебору елементів матриці. У тілі циклу на кожній ітерації вибирається поточний елемент матриці. Цей елемент порівнюється з поточним максимальним (мінімальним) значенням.

Програма, яка реалізує представлений алгоритм.

```
#include <iostream>
#define N 4
#define M 5
void main()
{
        int a[N] [M];
        int max, min;
        int imax = 0, jmax = 0;
        int imin = 0, jmin = 0;
        for (int i = 1; i <N; i ++)
                for (int j = 1; j <M; j ++)
                        cin >> a[i] [j];
        max = a[0] [0];
        min = a[0] [0];
        for (int i = 1; i <N; i ++)
                for (int j = 1; j <M; j ++)
                {
                        if (a[i] [j]> max)
                        {
                                max = a[i] [j];
                                imax = i;
                                jmax = j;
                        }
                        if (a[i] [j] <min)
                        {
```

```
                    min = a[i] [j];
                    imin = i;
                    jmin = j;
                }
            }
        cout << «max =« << max << '\ n';
        cout << «min =« << min << '\ n';
        cout << «imax =« << imax << '\ n';
        cout << «jmax =« << jmax << '\ n';
        cout << «imin =« << imin << '\ n';
        cout << «jmin =« << jmin << '\ n';
}
```

У наступному прикладі необхідно для цілочисельного масиву з 10 елементів, що вводиться з клавіатури, визначити кількість парних елементів, розташованих між максимальним і мінімальним елементами.

На початку програми необхідно визначити, де в масиві розташовані максимальний і мінімальний елементи, а точніше знайти їх місце розташування (індекси). Далі необхідно перебрати всі елементи, розташовані між ними і, якщо значення парне, збільшити лічильник елементів на одиницю. Очевидно, що порядок розташування елементів у масиві заздалегідь не відомий, і спочатку може слідувати як максимальний, так і мінімальний елемент, крім того вони можуть збігатися. Тому, перш ніж переглядати масив в пошуках кількості парних елементів, потрібно визначити, який з цих індексів більше.

Для того, щоб визначити місце розташування максимального елемента, необхідно задати початкове значення для індексу максимального елемента (*max_i*), наприклад, рівне нулю. Початкове значення для змінної максимального елемента (*max*) задається рівним першому елементу масиву *a*[0]. Далі необхідно переглянути всі елементи масиву, по черзі порівнюючи кожен елемент *a*[*i*] зі значенням максимуму. Якщо черговий елемент (*a*[*i*]) більше раніше знайденого максимуму (*max*), то прийняти цей елемент за новий максимум, а також в змінної *max_i* запам'ятати індекс даного елемента.

```
max = a[0];
```

```
max_i =0;
for (i =0; i <n; i ++)
        if (a[i]> max)
        {
        max = a[i];
        max_i = i;
        }
```

Для того, щоб визначити місце розташування мінімального елемента, необхідно задати початкове значення для індексу мінімального елемента (*min_i*), наприклад, рівне нулю. Початкове значення для змінної мінімального елемента (min) задається рівним першому елементу масиву a [0]. Далі необхідно переглянути всі елементи масиву, по черзі порівнюючи кожен елемент *a*[*i*] зі значенням мінімуму. Якщо черговий елемент (*a*[*i*]) менше раніше знайденого мінімуму (min), то прийняти цей елемент за новий мінімум, а також в змінної *min_i* запам'ятати індекс даного елемента.

```
min = a[0];
min_i =0;
for (i =0; i <n; i ++)
        if (a[i] <min)
        {
        min = a[i];
        min_i = i;
        }
```

Для знаходження кількості парних елементів між максимумом і мінімумом, спочатку необхідно визначити межі перегляду масиву. Якщо максимальний елемент розташований раніше, ніж мінімальний елемент (*max_i*<*min_i*), то необхідно організувати цикл *fo*r з параметрами – (*i* = *max_i*+1; *i*<*min_i*; *i* ++). Якщо мінімальний елемент розташований раніше, ніж максимальний елемент (*min_i*<*max_i*), то необхідно організувати цикл *for* з параметрами: (*i*=*min*_i+1; *i*<*max_i*; *i* ++). Якщо індекси максимального та мінімального елемента збігаються, то кількість парних елементів дорівнює 0. Зміну, яка приймає кількість парних елементів, до циклу потрібно обнулити (*k*=0). Для того, щоб перевірити чи є число парних,

необхідно перевірити залишок від ділення цього числа на 2, якщо залишок дорівнює 0 – число парне,

```
k =0;
if (max_i <min_i)
{
  for (i = max_i + 1; i <min_i; i ++)
        if (a [i]% 2 = = 0)
              k ++;
   cout << «кількість парних чисел між» << max << «і» << min << «=« << k << '\ n';
}
if (min_i <max_i)
{
        for (i = min_i +1; i <max_i; i ++)
        if (a [i]% 2 = = 0)
              k ++;
cout << «кількість парних чисел між» << min << «і» << max << «=« << k << '\ n';
}
if (min_i = = max_i)
cout << «максимальний елемент –» << a[max_i] << «мінімальний елемент» << a [min_i] << «збігаються \ n»;
```

Блок–схема алгоритму розв'язання задачі представлена на рис. 5.6.

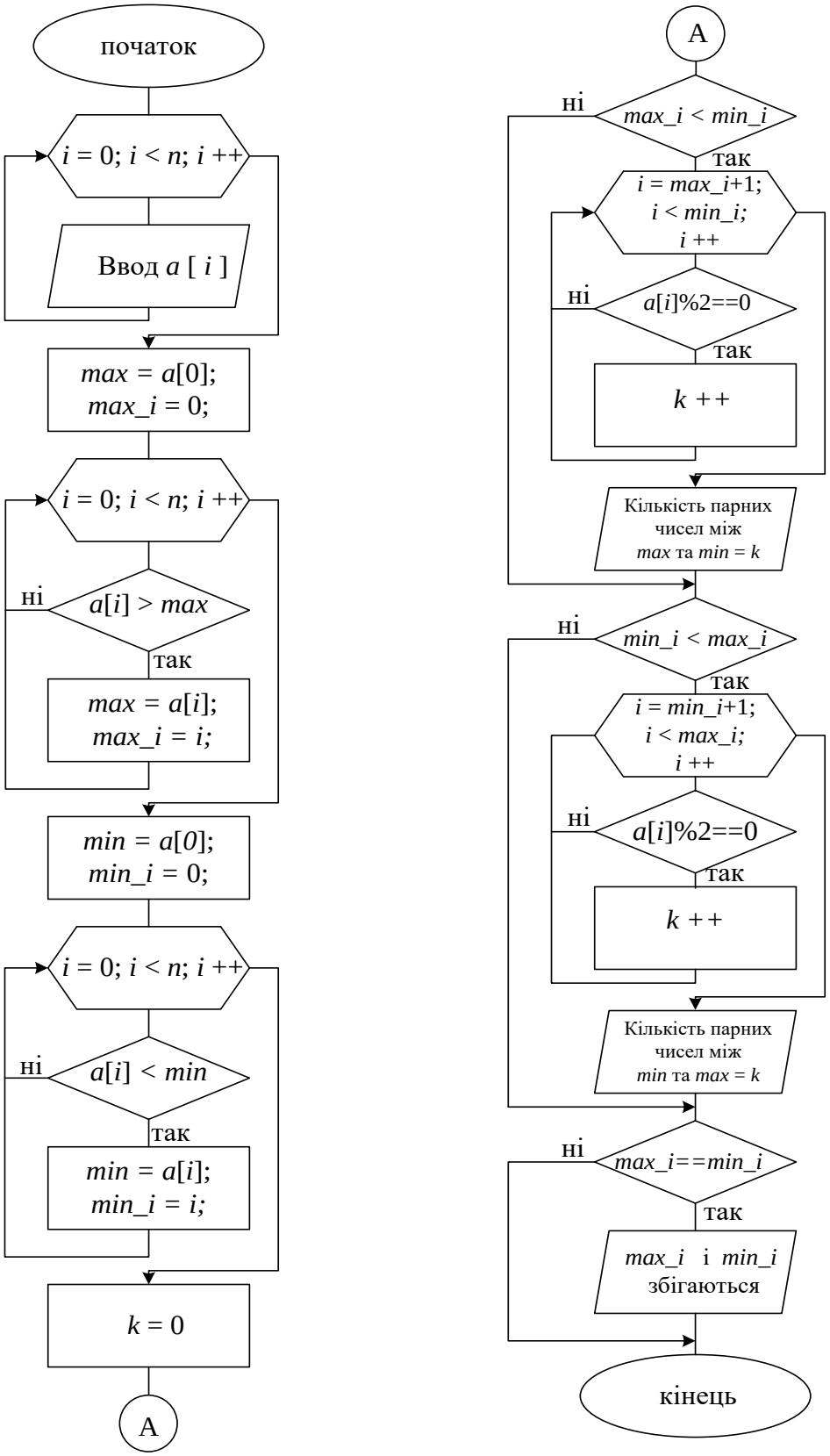


Рисунок 5.6 – Блок–схема алгоритму програми

```
# include <iostream>
const int n = 10;
int main ()
{
    int a [n], k, i, min, max, min_i, max_i;
    for (i = 0; i <n; i ++)
    {
       cout << «введіть «<< i <<« елемент масиву \ n «;
       cin >> a [i];
    }
    max = a[0];
    max_i= 0;
    for (i = 0; i <n; i ++)
       if (a [i]> max)
       {
            max = a [i];
            max_i = i;
       }
    min= a[0];
    min_i= 0;
    for (i =0; i <n; i ++)
       if (a[i] <min)
       {
            min = a [i];
            min_i = i;
       }
    k = 0;
    if (max_i <min_i)
    {
    for (i= max_i + 1; i <min_i; i ++)
      if (a [i]% 2 = = 0)
       k ++;
    cout << «кількість парних чисел між» << max << «і» << min <<
«=« << k << '\ n';
    }
    if (min_i <max_i)
```

```
        {
        for (i = min_i + 1; i <max_i; i ++)
              if (a[i]% 2 == 0)
                  k ++;
        cout << «кількість парних чисел між» << min << «і» << max <<
«=« << k << '\ n'
        }
        if (min_i = = max_i)
           cout<< «максимальний елемент –» << a[max_i] << «і
мінімальний елемент –» << a [min_i] << «збігаються \ n»;
        return 0;
    }
```

Алгоритми сортування даних також засновані на використанні масивів. Існує велика кількість різних методів сортування даних. Конкретний метод сортування вибирають з залежності від завдання, яку треба вирішити, даних, які треба обробити і т.д. Одними з найпоширеніших способів сортування є:

- сортування методом обміну;
- сортування методом вибору;
- сортування методом простої вставки.

Метод обмінного сортування ще називається «бульбашковим» методом. Алгоритм складається в повторюваних проходах по сортованого масиву. За кожен прохід елементи послідовно порівнюються попарно і, якщо порядок в парі невірний, виконується обмін елементів. Проходи по масиву повторюються до тих пір, поки на черговому проході не опиниться, що обміни більше не потрібні, що означає – масив відсортований. При проході алгоритму, елемент, що стоїть нема на своєму місці, «спливає» до потрібної позиції як бульбашка у воді, звідси і назва алгоритму. Схема алгоритму представлена на рис. 5.7.

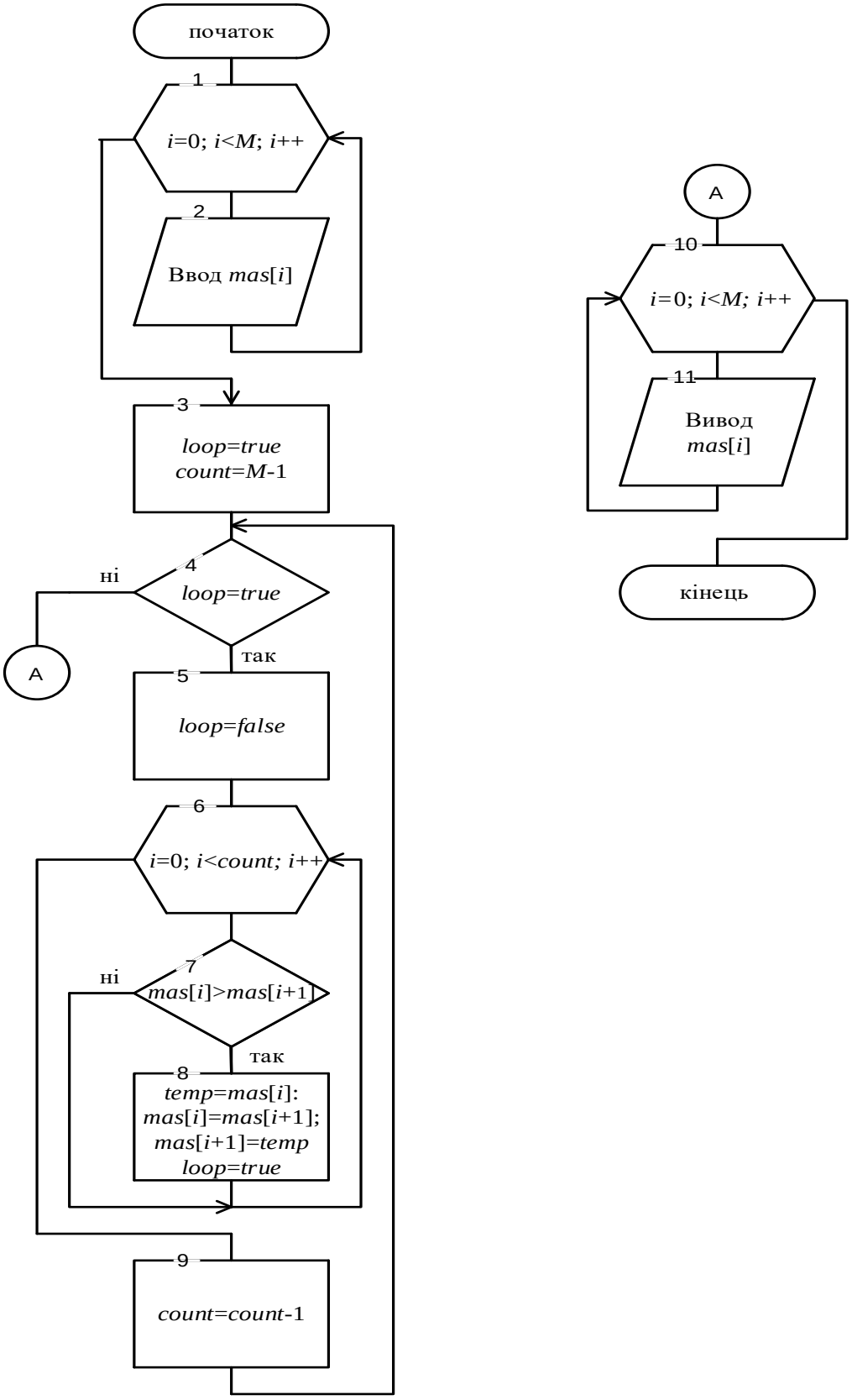


Рисунок 5.7 – Алгоритм програми сортування одновимірного масиву

Програма, що реалізує представлений алгоритм

```
#include <iostream>
#define M 10
void main()
{
	float mas[M], temp;
	for(Int i = 0; i <M; ++i)
		cin >> mas[i];
	bool loop = true;
	int count = M – 1;
	while (loop)
	{
		loop = false;
		for (int i = 0; i <count; ++ i)
			if (mas[i]> mas [i + 1])
			{
				temp = mas[i];
				mas[i] = mas [i + 1];
				mas[i + 1] = temp;
				loop = true;
			}
		count– –;
	}
	for (int i = 0; i <M; ++ i)
		cout << mas[i] << '\ n';
}
```

У програмі сортування проводиться в масиві *mas* [*M*]. В результаті роботи програми елементи масиву будуть впорядковані за зростанням їх значень. Слід також зазначити, що в результаті кожного перебору масиву максимальний елемент стає на своє місце. Тому для скорочення числа операцій використовується змінна *count*, яка має початкове значення, рівне розміру масиву, але після виконання кожного проходу зменшується на 1, зменшуючи тим самим довжину масиву. У найгіршому разі упорядковуватися буде масив, що складається з двох елементів. Може скластися ситуація, коли масив довжиною більше двох елементів вже впорядкований. При проході за таким масиву не буде виконано жодного

обміну, і процедуру сортування можна зупинити. Для визначення такої ситуації використовується логічна змінна *loop*.

Ще одним прикладом алгоритму простий сортування є ***алгоритм простий вставки***. Цей метод сортування менш ефективний, ніж більш складні алгоритми (наприклад, алгоритм швидкого сортування). Цей метод має ряд переваг – простота реалізації, ефективність на невеликих наборах даних, ефективність на частково впорядкованих даних, стійкість (алгоритм не міняє порядок елементів, які вже відсортовані), можливість виконання сортування вхідної послідовності даних (наприклад, при отриманні даних в телекомунікації, в кожен момент часу входить послідовність є відсортованої, тоді як для обмінного сортування, якщо в послідовність додаються дані, то сортування повинна бути в полнена ще раз повністю). Суть методу полягає в тому, що на кожному кроці алгоритму вибирається один з елементів вхідних даних і вставляється його на потрібну позицію в уже відсортованому списку, до тих пір, поки набір вхідних даних не буде вичерпаний (рис. 5.8).

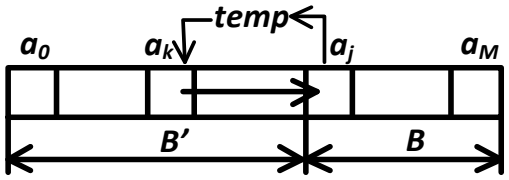


Рисунок 5.8 – Схема сортування методом простої вставки

Вважаємо, що на певному етапі виконання сортування масиву є послідовність B' з j–1 відсортованих значень. Елемент a_j масиву порівнюється послідовно з кожним із значень масиву від a_1 до a_{j-1} і поміщається на своє місце в відсортованої послідовності. На практиці це означає що, якщо елемент повинен бути розміщений в масиві в k–тій позиції, то він буде розміщений в цій позицію. При цьому впорядковані елементи масиву $a_k..a_{j-1}$ будуть зміщені в позиції $a_k + 1..a_j$. Далі виконуються тіж операції для елементів масиву $a_j + 1$, $a_j + 2$, ... поки набір вхідних даних не буде вичерпаний (рис. 5.8).

Вважаємо, що на певному етапі виконання сортування масиву є послідовність B' з j–1 відсортованих значень. Елемент a_j масиву

порівнюється послідовно з кожним із значень масиву від a_1 до a_{j-1} і поміщається на своє місце в відсортованої послідовності.

На практиці це означає що, якщо елемент повинен бути розміщений в масиві в *k*–тій позиції, то він буде розміщений в цій позицію. При цьому впорядковані елементи масиву $a_k..a_{j-1}$ будуть зміщені в позиції $a_k + 1.. a_j$.

Далі виконуються ті ж операції для елементів масиву $a_j + 1$, $a_j + 2$, ... поки набір вхідних даних не буде вичерпаний (рис. 5.9).

Вважаємо, що на певному етапі виконання сортування масиву є послідовність *B*' з *j*–1 відсортованих значень. Елемент a_j масиву порівнюється послідовно з кожним із значень масиву від a_1 до a_{j-1} і поміщається на своє місце в відсортованої послідовності. На практиці це означає що, якщо елемент повинен бути розміщений в масиві в *k*–тій позиції, то він буде розміщений в цій позицію.

При цьому впорядковані елементи масиву $a_k..a_{j-1}$ будуть зміщені в позиції $a_k + 1.. a_j$. Далі виконуються ті ж операції для елементів масиву a_j *j* + 1, $a_j + 2$, ... Елемент a_j масиву порівнюється послідовно з кожним із значень масиву від a_1 до a_{j-1} і поміщається на своє місце в відсортованої послідовності. На практиці це означає що, якщо елемент повинен бути розміщений в масиві в *k*–тій позиції, то він буде розміщений в цій позицію. При цьому впорядковані елементи масиву $a_k..a_{j-1}$ будуть зміщені в позиції $a_k + 1.. a_j$.

Далі виконуються ті ж операції для елементів масиву $a_j + 1$, $a_j + 2$, ... Елемент a_j масиву порівнюється послідовно з кожним із значень масиву від a_1 до a_{j-1} і поміщається на своє місце в відсортованої послідовності. На практиці це означає що, якщо елемент повинен бути розміщений в масиві в *k*–тій позиції, то він буде розміщений в цій позицію.

При цьому впорядковані елементи масиву $a_k..a_{j-1}$ будуть зміщені в позиції $a_k + 1.. a_j$. Далі виконуються ті ж операції для елементів масиву $a_j + 1$, $a_j + 2$, ...

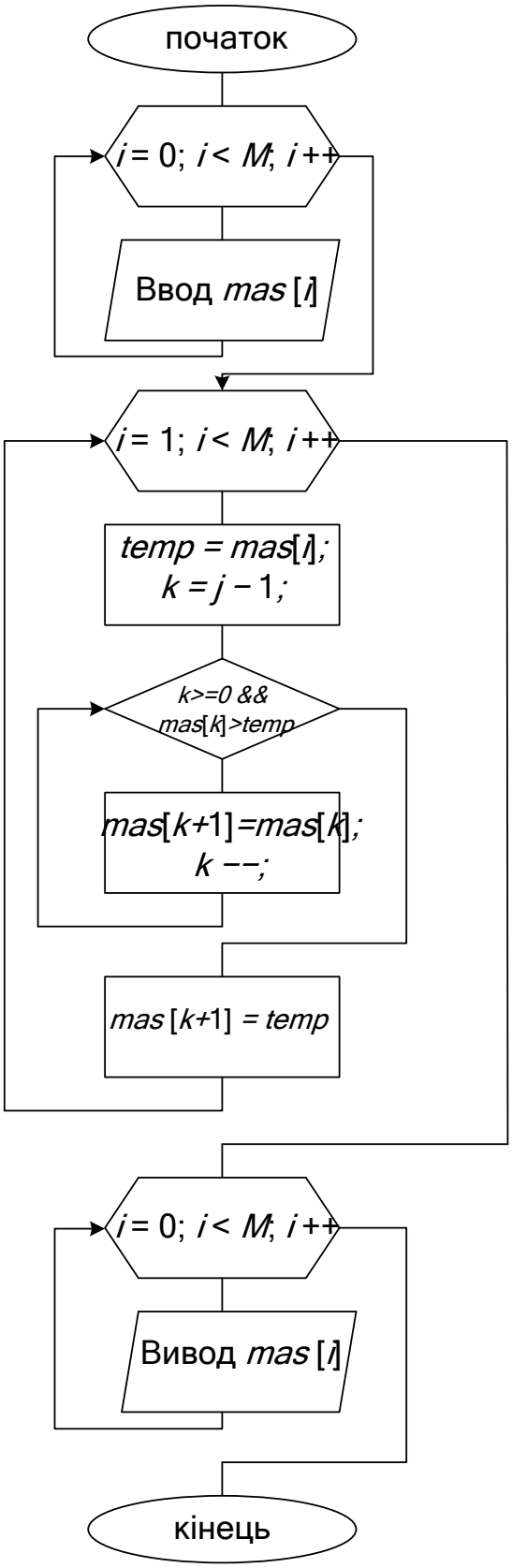


Рисунок 5.9 – Блок–схема алгоритму сортування методом простої вставки

Програма, що реалізує представлений на рис. 5.9 алгоритм:

```
#include <iostream>
#define M 5
void main ()
{
        float mas[M], temp;
        int k;
        for (int i = 0; i <M; ++ i)
                cin >> mas[i];
        for (int j = 1; j <M; ++ j)
        {
                temp = mas[j];
                k = j –1;
                while ((k> = 0) && (mas [k]> temp))
                {
                        mas[k +1] = mas [k];
                        k– –;
                }
                mas[k+1] = temp;
        }
        for (int i = 0; i <M; ++ i)
                cout << mas[i] << '\ n';
}
```

Сортування вибором характеризується квадратичною залежністю часу сортування *t* від кількості елементів:

$$t = a * N^2 + b * N * \lg N$$

де *a* і *b* – константи, що залежать від програмної реалізації алгоритму.

Іншими словами, для сортування масив потрібно переглянути *N* раз.

Ідея алгоритму полягає в наступному. Він полягає в тому що з масиву вибирається найменший елемент і міняється місцями з першим елементом, потім розглядаються елементи, починаючи з другого, і

найменший з них міняється місцями з другим елементом і так далі n–1 раз (при останньому проході циклу при необхідності міняються місцями передостанній і останній елементи масиву).

Наприклад є вихідна несортованими послідовність $a[0…n–1]$. Для сортування за зростанням масиву вибираємо з нього найменший елемент і ставимо його на перше місце. Тобто міняємо знайдений найменший елемент і перший. Потім в послідовності починаючи з 2–го елемента і до кінця – аналогічно шукаємо найменший елемент і міняємо його місцями з 2–м. І так далі до передостаннього елемента. Сортування закінчується, коли в залишилася невідсортоване послідовності залишається один елемент. Таким чином, алгоритм має n–1 ітерацій. На кожній i (= [0...n–1]) вибирається найменший і міняється місцями з $x[i]$. Коли $i=n–1$ сортування припиняється, і ми отримаємо відсортовану послідовність, блок–схема алгоритму програми представлена на рис. 5.10.

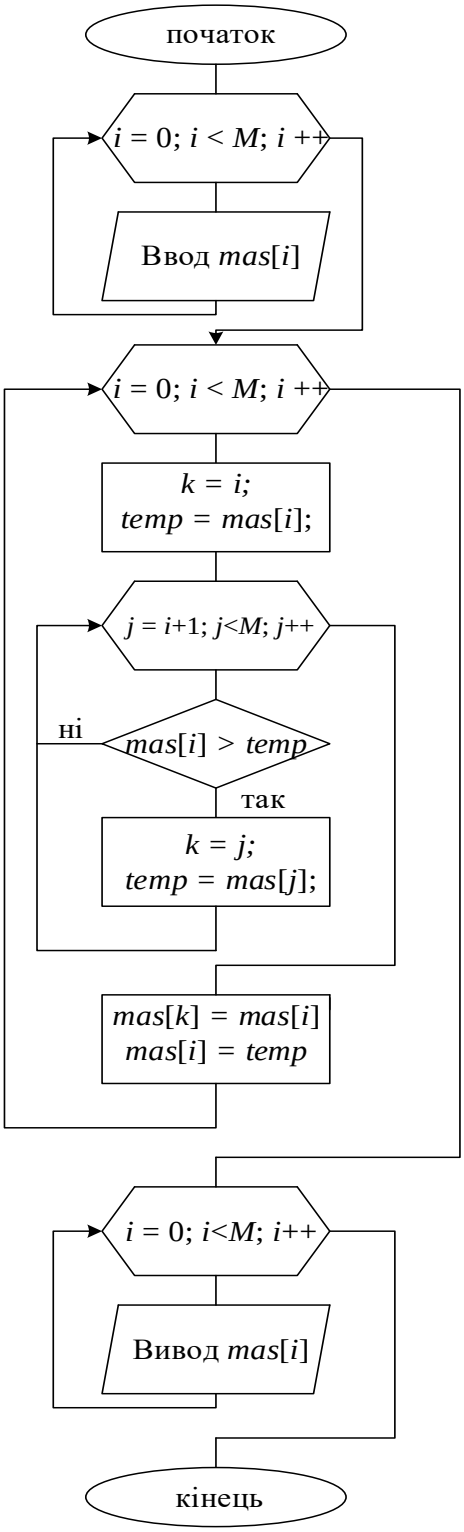


Рисунок 5.10 – Алгоритм сортування методом вибору

Наприклад, є послідовність: 7, 2, 6, 5, 10, 4. n = 6.
0 крок i=0, min=2 (i=1) міняємо місцями $x[0]$ = 7 та $x[1]$ = 2
2, 7, 6, 5, 10, 4

1 крок i=1, *min*=4 (i=5) міняємо місцями x[1] = 7 і x[5] = 4
2, 4, 6, 5, 10, 7
2 крок i=2, *min*=5 (i=3) міняємо місцями x[2] = 6 і x[3] = 5
2, 4, 5, 6, 10, 7
3 крок i=3, *min*=6 (i=3) міняємо місцями x[3] = 6 та x[3] = 6
2, 4, 5, 6, 10, 7
4 крок i=4, *min*=4 (i=5) міняємо місцями x[4] = 10 та x[5] = 7
2, 4, 5, 6, 7, 10

Програма, що реалізує представлений алгоритм:

```
#include <iostream>
#define M 5
void main ()
{
        float mas[M], temp;
        int k, i, j;
        for (int i = 0; i <M; ++i)
                cin >> mas[i];
        for (i = 0; i <M; i ++)
        {
                k = i;
                temp = mas [i];
                for (j= i + 1; j <M; j++)
                {
                        if (mas[j] <temp)
                        {
                                k = j;
                                temp = mas[j];      }
                }
                mas[k] = mas [i];
                mas[i] = temp;
        }
        for (int i = 0; i <M; ++i)
                cout << mas[i];
}
```

5.4. Адреси та покажчики

Під час виконання програми, дані що використовуються нею, розміщуються в оперативній пам'яті ЕОМ. Кожної змінної ставиться у відповідність її індивідуальну адресу – номер комірки пам'яті, де зберігається ця змінна. При реалізації багатьох алгоритмів і поданні складних логічних структур даних часто виявляється корисною можливість безпосередньої роботи з адресами пам'яті.

Така робота пов'язана, перш за все, з використанням покажчиків. Покажчик – це адреса пам'яті, що розподіляється для розміщення ідентифікатора (в якості ідентифікатора може виступати ім'я змінної, масиву, структури, строкового літерала). У тому випадку, якщо змінна оголошена як покажчик, то вона містить адресу пам'яті, по якому може знаходиться змінна будь–якого типу. При оголошенні змінної типу покажчик, необхідно визначити тип об'єкта даних, адреса якого буде містити змінна.

На рис. 5.11 показано розміщення в пам'яті двох змінних. Одна змінна представляє собою безпосередньо дані, які обробляються в програмі. Друга змінна є покажчиком, який вказує на дані, з якими працює програма.

Рисунок 5.11 – Схема розміщення в пам'яті змінних

Оскільки покажчик є змінною, то для його використання в програмі має бути виконано його опис. Покажчики бувають типізовані і не типізовані. В описі типізованого покажчика повинен бути безпосередньо

тип даних, на які цей покажчик буде вказувати. Опис не типізованого покажчика виконується наступним чином:

*тип * ідентифікатор;*

Наприклад:

*float * ptr_float;*
*double * prt_dbl1, * prt_dbl2;*
*int cnt, * ptr_int;*

Видно, що перед ім'ям змінної покажчиком повинен бути вказаний символ *. Крім того, в прикладах показано, як можна описувати покажчики списками змінних. Також показано, що в список змінних можуть бути включені як звичайні змінні, так і покажчики. В останньому прикладі змінна cnt не є покажчиком, а змінна *ptr_int* є вказівник.

Не типізовані покажчики описуються так

*void * ідентифікатор;*

У мові *C/C++* описаний спеціальний тип *void*, який вказує, що якийсь об'єкт програми, який описаний з таким типом не має ніякого типу.

Слід зазначити, що покажчики будь–якого типу мають однакову структуру. Тому часто покажчики використовують для передачі значень довільного типу всередині програми і між додатками. Для іменування покажчиків прийнято використовувати ідентифікатори, які починаються з символу *p*. Для роботи з покажчиками використовуються різні операції. Операція отримання адреси дозволяє отримати адресу деякої змінної і привласнити цю адресу вказівником. Синтаксис операції наступний:

& ім'я_змінної

Для маніпулювання даними, на які вказує змінна–вказівник використовується операція разименованія, синтаксис якої

** покажчик*

Розглянемо використання цих операцій на наступному прикладі

```
float val1, val2 = 0.672f;
float * pval;

pval = & val1;
val1 = sqrt(val2) +1;
cout << * pval << '\ n';
* pval = –4.728f;
cout << val1 << '\ n';
pval = & val2;
cout << * pval << '\ n';
```

У прикладі описані змінні *val1* та *val*2, а також покажчик на змінну *pval*. У першому операторі виконується операція отримання адреси змінної *val1*. Адреса цієї змінної зберігається в покажчику *pval*. Далі в операторі виведення виконується операція разименованія покажчика pval. На екран буде виведено значення, на яке вказує покажчик, а в поточний момент це значення змінної *val*1. У наступному операторі знову виконується операція разименованія. В даному операторі дійсне значення –4.728 буде збережено за адресою, який зберігається в покажчику pval. Оскільки в даний момент покажчик містить адресу змінної *val*1, то буде встановлено значення цієї змінної. Переконатися в цьому можна виконавши, оператор виведення, який йде слідом. Ще однією особливістю покажчиків є можливість змінювати в ході виконання програми область даних, на які покажчик посилається. Так в прикладі в двох останніх операторах покажчик *pval* отримує адресу змінної *val*2 і надалі працює зі значенням цієї змінної.

Також з покажчиками можна виконувати і інші операції, які можуть модифікувати значення покажчиків. Наприклад, покажчик можна модифікувати операцією інкремента (декремента) *pval*++ (*pval*—). Значення покажчика буде змінено на розмір типу, на який він вказує. У прикладі розглянуто тип покажчика *float*, розмір якого 4 байта, тому зміна значення покажчика в результаті операції відбудеться на 4 одиниці. Слід зазначити, що безпосередня модифікація покажчиків вимагає уважності. В результаті такої модифікації покажчик може вказувати на захищену

область пам'яті, операції з якою можуть викликати крах додатки або системи.

5.5. Масиви та вказівники. Адресна арифметика

У мові *C/C++* між покажчиками і масивами існує тісний зв'язок. Наприклад, коли оголошується масив у вигляді *int array*[25], то цим визначається не тільки виділення пам'яті для двадцяти п'яти елементів масиву, але і для покажчика з ім'ям array, значення якого дорівнює адресою першого за рахунком (нульового) елемента масиву, тобто сам масив залишається безіменним, а доступ до елементів масиву здійснюється через покажчик з ім'ям array. З точки зору синтаксису мови покажчик *array* є константою, значення якої можна використовувати у виразах, але змінити це значення не можна.

Оскільки ім'я масиву є вказівником припустимо, наприклад, таке присвоєння:

```
int array[25];
int * ptr;
ptr = array;
```

Тут покажчик *ptr* встановлюється на адресу першого елемента масиву, причому присвоювання *ptr=array* можна записати в еквівалентній формі *ptr*=& *array*[0].

Для доступу до елементів масиву існує два різні способи. Перший спосіб пов'язаний з використанням звичайних індексних виразів у квадратних дужках, наприклад:

```
array[16] = 3;
array[i + 2] = 7;
```

При такому способі доступу записуються два вирази, причому другий вираз у квадратних дужках. Одне з цих виразів має бути дороговказом, а друге – вираженням цілого типу.

Другий спосіб доступу до елементів масиву пов'язаний з використанням адресних виразів і операції разименованія

```
*(array + 16) = 3;
*(array + i + 2) = 7;
```

При реалізації на комп'ютері перший спосіб приводиться до другого, тобто індексний вираз перетворюється до адресного. Для наведених прикладів *array*[16] перетвориться в *(*array*+16).

Для доступу до початкової елементу масиву (тобто до елементу з нульовим індексом) можна використовувати просто значення покажчика *array* або *ptr*. Будь-яке з присвоювання привласнює початковому елементу масиву значення 2:

```
* array=2;
array[0]=2;
*(array + 0)=2;
* ptr=2;
ptr[0]=2;
*(ptr + 0)=2;
```

Як уже згадувалося вище над покажчиками можна виконувати унарні операції інкремента і декремента. При виконанні цих операцій значення покажчика збільшується або зменшується на довжину типу, на який посилається використовуваний покажчик. Цей підхід можна використовувати для обробки масивів

```
int * ptr, a[10];
...
ptr = & a[5];
cout << *(++ ptr) << '\ n';
cout << *(– –ptr) << '\ n';
```

У прикладі спочатку на екран буде виведено значення елемента масиву *a*[6], а потім значення елемента масиву *a*[5]. Строго кажучи, операція ++*ptr* є еквівалентною оператору

```
ptr = ptr + sizeof (int);
```

Тут використовується функція *sizeof*, яка має параметр, що задає ім'я типу і повертає значення рівне розміру цього типу в байтах.

У бінарних операціях додавання і віднімання можуть брати участь покажчик і величина типу *int*. При цьому результатом операції буде покажчик на вихідний тип, а його значення буде на вказане число елементів більше або менше вихідного.

```
int * ptr1, * ptr2, a[10];
int i = 2;
ptr1 = a + (i + 4);
ptr2 = ptr1 – i;
```

В даному прикладі покажчик *ptr*1 буде вказувати на елемент масиву *a*[6], а покажчик *ptr*2 буде вказувати на елемент *a*[4].

В операції віднімання можна використовувати два покажчика на один й той же тип. Результат такої операції має тип *int* та дорівнює числу елементів вихідного типу між зменшуваним і віднімаються, причому якщо перша адреса молодше, то результат має від'ємне значення.

```
int * ptr1, * ptr2, a[10];
int i;
ptr1 = a + 4;
ptr2 = a + 9;
i = ptr1 – ptr2;
i = ptr2 – ptr1;
```

В такому прикладі покажчик *ptr*1 буде вказувати на елемент масиву *a*[4], а покажчик *ptr*2 на елемент *a*[9]. Мінлива і в першому випадку прийме значення 5, а в другому випадку значення –5.

Значення двох покажчиків на однакові типи можна порівнювати в операціях відносини при цьому значення покажчиків розглядаються просто як цілі числа, а результат порівняння буде дорівнює *true* або *false*.

```
int * ptr1, * ptr2, a[10];
ptr1 = a + 5;
ptr2 = a + 7;
```

*if (prt*1> *ptr*2) *a*[3] = 4;

В такому прикладі покажчик *ptr*1 буде вказувати на елемент масиву *a*[5], а покажчик *ptr*2 на елемент *a*[7]. Значення покажчика буде *ptr*1 менше значення *ptr*2 і тому оператор *a*[3] = 4 не буде виконано.

5.6. Робота з рядками символів

У мові *C/C++* немає спеціального типу, який можна було б використовувати для опису рядків. Замість цього вони представляються у вигляді масивів елементів типу char. Таке рішення проблеми означає, що символи рядка розташовуються в сусідніх комірках пам'яті – по одному символу в комірці. У той же час, настільки спрощене уявлення про рядках дещо ускладнює практичну роботу з ними, оскільки в більшості випадків рядки цікавлять нас як цілісні в логічному відношенні одиниці інформації, а не як набори символів, з яких вони складаються. Частковий вихід із ситуації досягається шляхом приміщення символу ‘\0’ в кінець кожному рядки. Це означає, що під рядком слід розуміти послідовність символів, що починається в нульовому елементі масиву типу *char* і закінчується символом ‘\0’. При цьому потрібно пам'ятати, що описуючи символьний масив необхідно резервувати одну додаткову осередок пам'яті під зберігання ‘\0’. Так, наприклад, для подання рядка, що містить 15 символів, в програмі необхідно мати опис виду

char string[16];

Розмірність масиву повинна бути, принаймні, на одиницю більше істинної довжини рядка.

Ініціалізацію рядки можна зробити різними способами. Стандартною формою ініціалізації рядка є поелементне завдання значень символів

char group[13] = {‘Г’, ‘р’, ‘у’, ‘п’, ‘а’, ‘–’, ‘К’, ‘І’, ‘Т’, ‘5', ‘1’, ‘8’, ‘А’};

або

char group[] = {‘Г’, ‘р’, ‘у’, ‘п’, ‘а’, ‘–’, ‘К’, ‘І’, ‘Т’, ‘5', ‘1’, ‘8’, ‘А’};

Коротшим є спосіб ініціалізації строкової константою

char group[ ] = «Група КІТ–518А»;

Кількість елементів масиву на один більше, ніж кількість символів в рядку. Можливий доступ до символів рядка як до елементів масиву

group[1] == ‘р’, group [5] == ‘– ’.

Також можлива робота з рядком з використанням покажчика на неї. Наведена вище рядок може бути описана як

*char *group* = «Група КІТ–518А»;

У такому випадку доступ до символів рядка можливий через модифікацію покажчика

**(group* + 2) == *‘y’*, **(group* + 7) == *‘I’*.

Проте, з огляду на те, що в мові *С* немає коштів для роботи з масивами як з єдиним цілим, то і не існує можливості маніпулювати рядками як неподільними одиницями інформації. Це складне становище знімається за рахунок використання функцій із стандартної бібліотеки мови. Підключення бібліотеки виконується директивою препроцесора^

#include <string.h>

Основні функції для роботи з рядками, представлені в зазначеній бібліотеці наступні.

Функція ***strlen*** дозволяє визначити довжину рядка. Функція має один параметр – рядок символів, розмір, якої треба визначити. Повертає функція значення типу int, яке дорівнює кількості символів в рядку без завершального нульового символу.

*char * string;*
int strlen (string)
#include <string.h>
#include <iostream>

```
void main()
{
  char * faculty = «Комп'ютерні інформаційні технлогії»;
  int string_lenght = strlen (faculty);
  cout << «Довжина рядка «<< * faculty << '' << string_lenght <<« символів «<< '\ n';
}
```

Функція ***strcpy*** дозволяє копіювати дані з однієї строкової змінної в іншу. Функція має два параметри – перший вказує на рядок, в яку будуть скопійовані символи, другий параметр вказує на рядок, з якої будуть копіюватися символи (рис. 5.12)

```
char * string1, * string2;
char * strcpy(string1, stirng2)
```

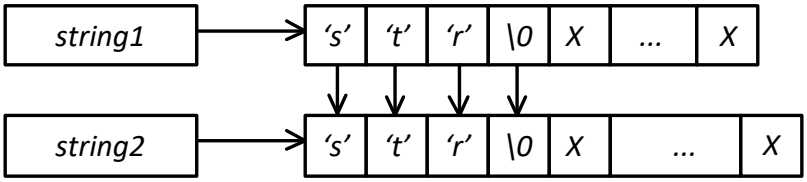


Рисунок 5.12 – Покажчики для строкових змінних

Функція ***strcat*** дозволяє зчепити два рядки в одну. Функція має два параметри – перший вказує на один рядок, другий параметр вказує на другу, результат зберігається в рядку, на яку вказує перший параметр функції (рис. 5.13).

```
char * string1, * string2;
char * strcat(string1, stirng2)
```

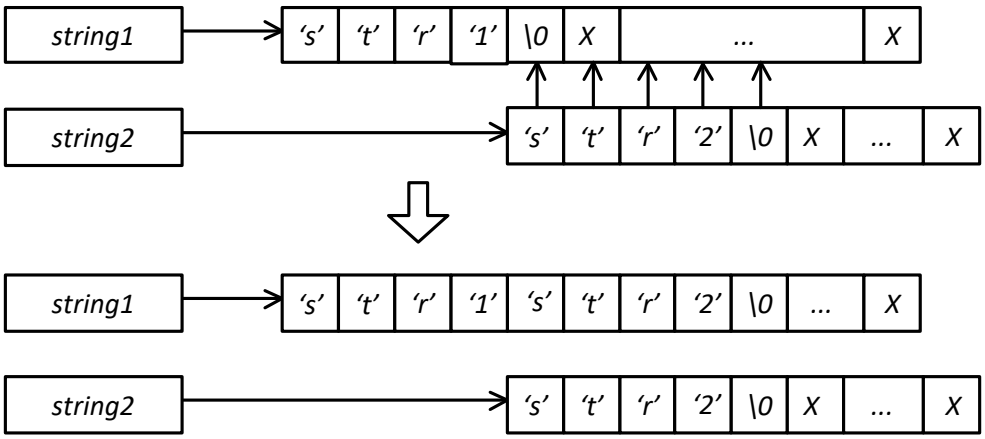


Рисунок 5.13 – Схема роботи функції *strcat*

У наступному прикладі в залежності від числа, що вводиться формується рядок для назви групи, у змінній *group*. Для формування рядка використовуються функції *strcpy* та *strcat*.

```
#include <string.h>
#include <iostream>
void main()
{
  int num;
  char group[10];
  char * faculty = "КІТ–";
  char * g518А = "518А";
  char * g518Б = "518Б";
  char * g518В = "518В";
  cin>>num;
  strcpy (group, faculty);
  switch (num)
  {
        case 1:
              strcat(group, g518A);
              break;
        case 2:
              strcat(group, g518Б);
```

```
            break;
        case 3:
            strcat(group, g518B);
            break;

  }
  cout << «група» << group << '\ n';
}
```

Використання строкових масивів. Масиви рядків також широко використовуються в розробці прикладних програм. Описати такий масив можна різними способами.

Оскільки рядок є сама по собі масивом, то масив рядків може бути представлений двовимірним масивом символів

```
char strarr[3] [20];
```

Такий масив може містити 3 рядки довжиною до 20 символів кожна. Використовувати масив можна так

```
strcpy(strarr [1], «рядок»);
```

У прикладі елементу масиву з індексом 1 буде присвоєна рядок символів «рядок».

Також строковий масив може бути описаний як масив покажчиків на рядки. У прикладі виконано опис масиву рядків з ініціалізацією

```
char *courses[3] =
{
        { 'М', 'а', 'т', 'е', 'м', 'а', 'т', 'і', 'до', 'а', '\ 0'},
        { 'ф ',' і ',' з ',' і ',' до ',' а ',' \ 0 '},
        { 'і ',' н ',' ф ',' про ',' р ',' м ',' а ',' т ',' і ',' до ',' а ',' \ 0 '},
};
```

або більш коротка форма

```
char *courses[] = { «Математика», «фізика», «інформатика»};
```

Практичні завдання

1. Створити масиву з 20 дійсних чисел. Знайти різницю між мінімальним і максимальним елементами масиву.

2. Створити масив з 20 цілих чисел. Знайти порядковий номер другого максимального числа.

3. Створити масив з 15 дійсних чисел. Визначити, чи утворюють елементи масиву зростаючу послідовність.

4. Створити одновимірний масив з 25 цілих чисел. Визначити кількість позитивних чисел, які діляться без залишку на 3.

5. Створити масив з 20 дійсних чисел. Визначити, скільки з них більше своїх «сусідів», тобто попереднього і подальшого чисел.

6. Створити масив з 20 цілих чисел. Визначити, скільки разів у введеної послідовності відбувається чергування позитивних і негативних чисел, тобто змінюється знак.

7. Створити одновимірний масив з 15 цілих чисел, які є числами Фібоначчі.

8. Створити масив з 20 дійсних чисел. Поміняти місцями максимальний і мінімальний елементи масиву. Результат вивести на екран.

9. Дан масив з 20 дійсних чисел. Знайти середнє арифметичне елементів масиву між максимальним і мінімальним елементами, включаючи їх.

10. Створити одновимірний масив, що складається з подільників цілого числа n, введеного з клавіатури. Вивести на екран елементи масиву, що є простими числами.

11. Знайти суму позитивних елементів двох діагоналей масиву. Визначити, чи є отримане округлене число простим.

12. Визначити, чи є задана квадратна матриця симетричною відносно головної діагоналі.

13. Обчислити середнє арифметичне значення негативних елементів не головною діагоналі матриці. Знайти факторіал отриманого округленого числа.

14. Дано два одновимірних масиву. Визначити елементи одного масиву, які не входять до іншого масив.

15. Позитивні елементи двовимірного масиву записати в одновимірний масив. Визначити максимальний і мінімальний елемент отриманого масиву.

16. Створити програму, яка вводить с клавіатури масив рядків і ще два рядки і замінює в масиві все символи, знайдені в першому рядку, на відповідні символи другого рядка (транслює рядки). Видати оттранслировать рядки і масив пар «символ» – «число замін».

17. Дано пропозицію. Визначити число входжень в нього деякого поєднання з двох заданих символів.

18. Дан текст. Замінити кожен символ тексту його *ASCII*–кодом.

19. Створити програму, яка вводить с клавіатури масив рядків, підраховує реальну довжину кожного рядка без врахування пробілів і виводить на екран (разом із самою рядком). Програма повинна видавати також число прогалин в кожному рядку.

20. Створити програму, яка вводить с клавіатури масив рядків, шукає в ньому рядки коротше зазначеного числа (наприклад, 70) і видає такі рядки на екран, доповнені пробілами між словами до потрібної довжини (text justification). Решта рядків обрізаються до тієї ж потрібної довжини. Слова розділяються пробілами

21. Створити програму, яка вводить с клавіатури масив рядків, і виводить список комбінацій «слово» – «рядка, в яких воно зустрічається»: слово1 4 12 слово2 5 14 відсортованого за алфавітом слів.

22. Дано слово з 12 букв. Поміняти місцями його третини наступним чином: першу третину слова розмістити на місці третин, другу третину – на місці першої, третю третину – на місці другої.

23. Дан текст. Скласти програму перекладу тексту за таким правилом: літери стоять до першої голосної, пересуваються в кінець слова, і до новоутвореного слова додається закінчення «ціус». Наприклад: «кіт у чоботях» – «откціус вціус апогахсціус».

24. Створити програму, яка вводить с клавіатури масив рядків, підраховує число слів в кожному рядку і виводить на екран разом з рядками. Видати також список пар «число слів» – «число рядків з такою кількістю слів». Слова розділяються пробілами.

25. Створити програму, яка вводить с клавіатури масив рядків, знаходить в ньому все повторювані послідовності (aa, bbb і т.д.) і замінює кожну з них на два елементи: повторюваний символ і довжину

послідовності. Видає на екран результуючий масив і список пар «довжина послідовності» – «число послідовностей із заданою довжиною».

26. Створити масив з 20 дійсних чисел. Знайти різницю між мінімальним і максимальним елементами масиву.

27. Створити масив з 15 дійсних чисел. Визначити найбільшу з негативних чисел, округлити його до найближчого цілого.

28. Створити масив з 20 цілих чисел. Знайти порядковий номер другого максимального числа.

29. Створити масив з 15 дійсних чисел. Визначити, чи утворюють елементи масиву зростаючу послідовність.

30. Створити одновимірний масив з 25 цілих чисел. Визначити кількість позитивних чисел, які діляться на 3 без залишку.

31. Створити масив з 30 цілих чисел випадковим чином. Вивести на екран елементи, порядкові номери яких є непарними.

32. Створити масив з 20 дійсних чисел. Визначити скільки елементів більше своїх «сусідів», тобто попереднього і подальшого чисел.

33. Створити масив з 20 цілих чисел. Визначити скільки разів у введеної послідовності відбувається чергування позитивних і негативних чисел, тобто змінюється знак.

34. Створити масив з 30 цілих чисел. Обчислити суму тих з них, порядкові номери яких - прості числа.

35. Створити одновимірний масив з 15 цілих чисел, які є числами Фібоначчі.

36. Створити масив з 20 дійсних чисел. Поміняти місцями максимальний і мінімальний елементи масиву. Результат вивести на екран.

37. Дан масив з 20 дійсних чисел. Знайти середнє арифметичне елементів масиву між максимальним і мінімальним елементами, включаючи їх.

38. Створити масив з 30 цілих чисел. Вивести на екран ті елементи, які є числами Фібоначчі.

39. Ввести з клавіатури масив з 20 цілих чисел. Вивести на екран ті елементи, які є досконалими числами.

40. Створити одновимірний масив, що складається з подільників цілого числа n, введеного з клавіатури. Вивести на екран елементи масиву, що є простими числами.

41. Створити масив з 10 цілих чисел. Перевірити, чи є масив паліндромом.

42. Масив з 10 дійсних чисел заданий випадковим чином. Замінити всі негативні елементи масиву їх модулями.

43. Створити масив з цифр цілого числа n, введеного з клавіатури. Знайти кількість парних чисел створеного масиву.

44. Створити двовимірний масив, що складається з дійсних чисел. Знайти номер рядка, в якій знаходиться максимальний елемент масиву. Знайти середнє арифметичне елементів, розташованих тільки в парних шпальтах масиву. Результат вивести на екран.

45. Створити двовимірний масив, що складається з дійсних чисел. Обчислити середнє арифметичне значення елементів головної та побічної діагоналі матриці. Знайти місце розташування максимального елемента двовимірного масиву. Результат вивести на екран.

46. У створеному двовимірному масиві вибрати елементи, які більше суми головною діагоналі масиву, записати отримані значення в одновимірний масив. Відсортувати отриманий одновимірний масив в порядку убування.

47. Створити двовимірний масив, що складається з цілих чисел. Знайти суму чисел, які розташовані вище головної діагоналі. Визначити, чи розташовані максимальний і мінімальний елементи масиву в одному рядку або в одному стовпці.

48. Створити двовимірний масив, що складається з дійсних чисел. Знайти добуток двох максимальних і різниця двох мінімальних елементів двовимірного масиву. Створити одновимірний масив, що складається з сум елементів рядків двовимірного масиву. Результат вивести на екран.

49. Створити двовимірний масив *a*, що складається з цілих чисел. Отримати масив *b* з масиву a, шляхом видаленням *n*-го рядка і *k*-го стовпця, де n і k - цілі числа, введені з клавіатури.

50. Створити двовимірний масив, що складається з цілих чисел. Знайти суму елементів двовимірного масиву, які діляться на п'ять. Визначити подільники отриманого числа. Створити одновимірний масив, що складається з творів елементів стовпців.

51. Створити матрицю, що складається цілих чисел, розміром 5x5. Побудувати матрицю, зворотну заданої. Створити одновимірний масив, що складається з мінімальних елементів кожного рядка. Відсортувати отриманий масив в порядку убування.

52. У двовимірному масиві, що складається з дійсних чисел, знайти максимальний елемент масиву і поміняти його місцями з елементом, що стоїть на перетині двох діагоналей.

53. У двовимірному масиві, що складається з цілих чисел, знайти суму елементів, що стоять тільки на парних рядках. Перевірити, чи є отримане число – числом Фіббоначі. Створити одновимірний масив, що складається з максимальних значень елементів кожного стовпця.

54. Створити двовимірний масив, що складається з цілих чисел. У створеному масиві в рядках, які мають хоча б один нульовий елемент знайти кількість елементів кратних 5. Створити одновимірний масив, що складається з значень середніх арифметичних елементів рядків.

55. Створити двовимірний масив, що складається з цілих чисел. У створеному масиві знайти мінімальний елемент побічної діагоналі. Створити одновимірний масив, що складається з значень творів елементів рядків двовимірного масиву.

56. Створити двовимірний масив, що складається з цілих чисел. У створеному масиві поміняти місцями рядок, що містить елемент з мінімальним значенням, з рядком, що містить елемент з максимальним значенням. Створити одновимірний масив, який складається з простих чисел двовимірного масиву.

57. Створити матрицю, що складається з цілих чисел Написати програму, яка встановлює, чи є дана матриця симетричною відносно головної діагоналі. Створити одновимірний масив, що складається кількостей парних елементів рядків двовимірного масиву.

58. У двовимірному масиві замінити нулем всі елементи, розташовані на головній діагоналі і вище неї. Знайти суму елементів *k*-го стовпця і *p*-го рядка (*k, p* – вводяться з клавіатури).

59. У двовимірному масиві знайти кількість парних елементів в рядках, що містять хоча б один нульовий елемент. Визначити, чи є отримане число – простим. Створити одновимірний масив, що складається з мінімальних елементів стовпців двовимірного масиву.

60. Дан двовимірний масив з цілих елементів. Знайти середнє арифметичне елементів, у яких сума індексів дорівнює *k* (*k* – вводиться з клавіатури). Створити одновимірний масив, що складається з парних елементів двовимірного масиву.

Контрольні питання

1. Масиви змінних. Призначення. Опис масивів.

2. Одномірні масиви на мові програмування С ++.

3. .Многомерние масиви. Опис багатовимірних масивів.

4. Ініціалізація одновимірних і багатовимірних масивів.

5. Дороговкази в програмі. Призначення. Необхідність використання покажчиків.

6. Алгоритми сортування. Різні типи алгоритмів, порівняльний аналіз.

7. Алгоритм сортування «бульбашковим» методом. Принцип дії, реалізація алгоритму.

8. Алгоритм сортування методом вставки. Принцип дії, реалізація алгоритму.

9. Рядки символів в програмі. Опис рядків. Функції рядків.

10. Ініціалізація символьних масивів. Строкові масиви.

11. Масиви та вказівники. Адресна арифметика.

12. Алгоритми знаходження екстремумів. Призначення, приклади використання.

13. Алгоритми знаходження складних сум. Призначення, приклади використання.

14. Алгоритм знаходження місця розташування максимальних і мінімальних елементів масиву.

15. Використання строкових функцій в програмі.

РОЗДІЛ 6.
ФУНКЦІЇ В МОВІ ПРОГРАМУВАННЯ *C* ++

6.1. Опис і структура функцій

Принципи програмування на мові *C/C++* засновані на понятті функції. Вище вже розглянуті кілька функцій *printf*(), s*canf*(), математичні функції модуля *math* (*fabs*(), *s*in(), ...). Ці функції є системними. Крім того головною частиною програми також є функція – це функція *main*(). Виконання програми завжди починається з команд, що містяться в функції *main*(), а всередині її можуть викликатися інші функції.

Функція – це самостійна одиниця програми, спроектована для реалізації конкретного завдання. Виклик функцій призводить до виконання деяких дій. Наприклад, при зверненні до функції *printf*() здійснюється виведення даних на екран. Використання функцій в додатках дозволяє досить ефективно побудувати процес розробки програмних проектів:

- використання функцій позбавляє від повторного програмування, тому що якщо деяка задача в програмі повинна бути виконана кілька разів, то доцільно оформити її рішення як функцію;
- реалізація рішення у вигляді функції підвищує рівень модульності програми, і , Отже, полегшують її читання, внесення змін і корекцію помилок, а також дозволяє виконувати розробку програмного проекту групою програмістів;
- також функції забезпечують абстракцію в програмі, коли окремі частини програми розглядаються як «чорні ящики», на вхід яких надходять вихідні дані, а на виході виходить шуканий результат (за умови правильного використання функції).

Опис функції в програмі має наступну структуру:

```
тип ідентифікатор (формальні параметри)
{
  опис1;
  опис2;
  ...
  оператор1;
  оператор2;
```

```
  return вираз;
}
```

Параметр ідентифікатор функції це символьне ім'я, по якому виконується виклик даної функції. Параметр тип функції визначає тип значення, яке повертає дана функція. В процесі виконання функція може обчислювати деяке значення і повертати його викликає функції. Наприклад, функція, яка обчислює суму деякої послідовності чисел, може повертати значення цієї суми. Також, значення, які повертаються функціями, використовуються для діагностики правильності виконання функції – якщо функція виконана успішно, то функція повертає деяке значення, якщо в процесі виконання функції виникла помилка, то функція може повернути її код. Тип функції може бути стандартним або призначеним для користувача. Крім того, функція може не повертати значення. У цьому випадку тип функції вказується як *void*. Формальні параметри функції є список змінних, через які функція обмінюється із зухвалою функцією. Через формальні параметри можна передати ті чи інші значення в функцію, а також отримати значення з функції. Усередині функція являє собою набір описів і операторів, які складають тіло функції.

Виконання функції завершується при виконанні оператора повернення *return*. Параметром оператора є деякий вираз, значення якого повертає функція. Функція може містити кілька операторів повернення. Для функцій типу *void* параметр вираз в операторі повернення відсутня, і оператор повернення може бути опущений. У цьому випадку вихід з функції виконується після завершення виконання останнього оператора тіла функції. Через формальні параметри можна передати ті чи інші значення в функцію, а також отримати значення з функції. Усередині функція являє собою набір описів і операторів, які складають тіло функції.

Виклик функції. Для виклику функції в програмі необхідно вказати її ім'я і задати перелік необхідних параметрів виклику. Можна розглянути механізм виклику функцій на прикладах.

У наступному прикладі необхідно за запитом користувача виводити інформацію про автора програми.

```
#include <iostream>
void about()
{
```

```
  cout << «Цю програму розробив Іванов І.І» << '\ n';
}

void main()
{
  char answer;
  ...
  cout << «Хочете дізнатися хто розробив програму?[Y / n] «<< '\ n';
  cin >> answer;
  if ( (Answer == 'y') || (Answer == 'Y'))
        about();
  ...
}
```

Виведення інформації про автора в прикладі реалізований як безтипових функція. Функція повинна бути описана до її виклику, тому в тексті програми функція *about*() розташована раніше функції *main*(), в якій вона викликається. В основній функції *main*() на екран виводиться запит про необхідність виводячи інформації про автора. Відповідь користувача зберігається в змінної answer. В умовному операторі перевіряється символ, який Ви самі ввели. Якщо символ відповідає позитивної відповіді, то викликається функція *about*(). Оскільки функція безтипових, виклик функції виконується окремим оператором в програмі. При виконанні функція *about*() виводить рядок інформації та закінчує своє виконання. Після чого здійснюється повернення в точку виклику, тобто в функцію *main*() і функція *main*() продовжує своє виконання.

У наступному прикладі реалізована функція визначення мінімального значення з двох аргументів.

```
#include <iostream>

float min(Float fst, float sec)
{
  if(Fst <sec)
        return fst;
  else
        return sec;
```

```
}

void main ()
{
  float val1, val2;

  cin >> val1 >> val2;

  cout << «Мінімальне значення» << min (val1, val2) << '\ n';
}
```

Функція *min*(*a, b*) викликається в функції *main*() як частина оператора виведення. При виконанні цього оператора відбуватися виклик функції min, яка обчислює деяке значення і це значення підставляється в оператор виведення і виводиться на екран. Функція min містить два параметри. При виконанні функції значення змінної *val*1 копіюється в формальний параметр *fst*, а значення змінної *val*2 копіюється в параметр *sec*. При виконанні функції формальні параметри порівнюються. Функція повертає значення меншого з формальних параметрів.

6.2. Передача параметрів в функцію

У попередньому прикладі викликалася функція, в яку передавалися деякі значення, а потім ці значення використовувалися в функції. Вище вже згадувалося, що передача значення виконується з використанням списку формальних параметрів. Значення і змінні, які вказуються в дужках при виконанні функції, називаються фактичними параметрами.

Формальні параметри в заголовку функції задаються списком. У списку кожне значення є ідентифікатор формального параметра із зазначенням його типу. Один від одного параметри в списку відділені комами. При виконанні функції фактичні параметри також представлені у вигляді списку елементів, розділених комами. Список фактичних параметрів повинен відповідати списку формальних параметрів. При виконанні функції це відповідність дотримується наступним чином – значення першого фактичного параметра присвоюється першому формальному параметру, значення другого фактичного параметра присвоюється другому формальному параметру і т.д. (рис. 6.1)

// опис функції

тип функція(Тип формал_парам1, тип формал_парам2, тип формал_парам3, ...)

{

}

...

//виклик функції

функція(Фактич_парам1, фактич_парам2, фактич_парам3, ...);

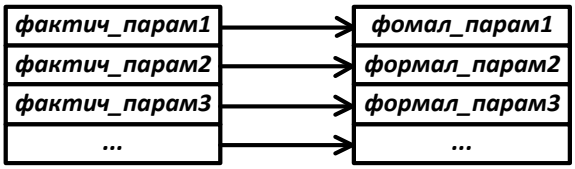


Рисунок 5.1 – Передача в функцію фактичних параметрів

Передача значень може бути виконана кількома способами:

- передача параметра за значенням;
- передача параметра за вказівником;
- передача параметра за посиланням.

Передача параметра за значенням. Даний метод є найпоширенішим і використовується, коли необхідно передати деяке значення в функцію для використання всередині функції. В цьому випадку формальні параметри є локальними змінними всередині функції. При виконанні функції значення фактичних параметрів копіюються у відповідні їм формальні параметри.

У наступному прикладі функція використовується для визначення потрапляє деяке значення в заданий діапазон.

```
bool in_range (float val, float low, float hi)
{
  if ((val> low) && (val <hi))
        return true;
  else
        return false;
```

```
}
void main ()
{
  float x, min, max;
  ...
  if (in_range (x, min, max))
        cout << «Потрапляє» << '\ n';
  ...
  if (in_range (x, 0.f, 1.f))
        cout << «Знову потрапляє» << '\ n';
  ...
}
```

Функція *in_range* має три формальних параметра. Перший параметр *val* містить значення, яке перевіряється на потрапляння в діапазон. Параметри *low* і *hi* містять значення нижньої і верхньої меж цього діапазону. При виконанні функції змінна *val* порівнюється з межами діапазону. Якщо її значення більше нижньої межі і менше верхньої, то функція поверне значення true, інакше функція поверне значення *false*.

При першому виклику функції в прикладі фактичні параметри є змінними. Значення змінної *x* присвоюється формальному параметру *val*, значення змінної min присвоюється формальному параметру *low*, а значення змінної max присвоюється формальному параметру *hi*. І в даному випадку функція *in_range* визначить лежить чи ні значення *x* в діапазоні значень *min .. max*.

При другому виконанні функції показано, що в якості фактичних параметрів можна вказувати не тільки змінні, але і константи. У цьому прикладі формальному параметру *low* буде присвоєно значення 0.*f*, а формальному параметру *hi* буде присвоєно значення 1.*f*. Функція визначить чи лежить значення *x* в діапазоні значень 0..1.

Можна також запропонувати більш короткий спосіб реалізації функції *in_range*.

```
bool in_range (float val, float low, float hi)
{
  return ((val> low) && (val <hi));
}
```

Таке можливо, якщо згадати якого типу результат повертають операції відносини і логічні операції. При виконанні функції буде обчислено значення виразу в операторі повернення. Результат цього виразу має логічний тип зі значеннями *true* або *false*.

Передача параметра за вказівником. При використанні передачі в функцію параметра за значенням, є можливість передати тільки значення в функцію, але не можна отримати значення з функції. Далі приклад функції, яка обмінює значення двох цілих змінних.

```
void swap (int u, int v)
{
  int t = u;
  u = v;
  v = t;
}

void main ()
{
  int x = 10, y = 124;
  ...
  cout << «До x = «<< x << '' <<« y = «<< y << '\ n';
  swap (x, y);
  cout << «Після x = «<< x << '' <<« y = «<< y << '\ n';
  ...
}
```

При виконанні висновок програми буде таким

До *x*= 10 *y* = 124.
Після *x*= 10 *y* = 124.

Таким чином, незважаючи на те, що змінилися значення формальних параметрів всередині функції, відповідні їм значення фактичних параметрів залишилися незмінними.

Для передачі в зухвалу функцію змінених значень параметрів використовується передача параметра за вказівником. При використанні

цього методу формальний параметр повинен бути заданий як покажчик. Відповідний йому фактичний параметр при цьому повинен задавати адресу деякої змінної. При виконанні функції в неї передається не значення фактичного параметра, а адреса де зберігається змінна фактичного параметра. Всі маніпуляції всередині функції виконуються із значенням, яке зберігається за поданою адресою і, отже, всі зміни, які зробила функція з цим значенням, збережуться після її завершення. Наведений вище приклад повинен бути реалізований так

```
void swap (int * u, int * v)
{
  int t = * u;
  * u = * v;
  * v = t;
}
void main ()
{
  int x = 10, y = 124;
  ...
  cout << «До x = «<< x << '' <<« y = «<< y << '\ n';
  swap ( & x, & y);
  cout << «Після x = «<< x << '' <<« y = «<< y << '\ n';
  ...
}
```

Крім того в функції swap описана змінна *t*. Така змінна називається локальної змінної, і використовувати її можна тільки в цій функції.

Передача параметра за посиланням. Ще один спосіб передачі значень з деякої функції в зухвалу функцію є передача параметра за посиланням. Посилання – це альтернативне ім'я деякої змінної. Формальний параметр може бути описаний як посилання наступним чином

```
тип & параметр
```

Можна розглянути використання посилань на наступному прикладі. Реалізована нижче функція змінює значення деякої дійсної змінної на її абсолютне значення.

```
void to_abs (double & arg)
{
  if (arg <0)
        arg = –arg;
}
```

Виклик функції виконується так

```
double double_val;
...
to_abs (double_val);
```

У зухвалій функції оголошена змінна *double_val*. Формальний параметр *arg*, оголошений як посилання є, по суті, ще одним ім'ям змінної *double_val*. І це альтернативне ім'я використовується всередині функції для обчислень. Таким чином всі зміни, які будуть виконуватися в функції зі змінною *arg* будуть виконані зі змінною *double_val*.

Використання посилань для передачі параметрів у функцію ще цікаво тим, що при такій передачі не виконується копіювання даних з фактичного параметра в формальний. При передачі у функцію змінної великого розміру за значенням таке копіювання може займати значний час і в деяких випадках доцільно замінити передачу параметра за значенням на передачу параметра за посиланням. Крім того, при передачі параметра за значенням фактичний параметр не змінюється після завершення функції. Якщо Ви використовуєте дзвінок параметра за посиланням як альтернатива передачі за значенням і передбачається, що параметр не буде змінений всередині функції можна заборонити можливість такої зміни. Для цього використовуються константні посилання. У прикладі функція обчислює довжину вектора в тривимірній системі координат по його відомим координатам.

```
#include <math.h>
```

*void vec*3d_len (const double & x, const double & y, const double & z, double & len)

{

len = sqrt(*pow* (*x*, 2) + *pow* (*y*, 2) + *pow* (*z*, 2));

}

Функція буде викликатися так

*double coord*1, *coord*2, *coord*3, *vector*;

...

*vec*3*d_len* (*coord*1, *coord*2, *coord*3, *vector*);

У функції *vec3d_len* перші три формальних параметра є константними посиланнями і їх не можна змінити всередині функції. Четвертий параметр описаний як звичайне посилання і його значення обчислюється всередині функції. Виклик функції виконується без використання покажчиків.

Використання масивів як формальних параметрів функцій. Завдання передачі в функцію деякого масиву виникає досить часто. Наприклад, при обчисленні максимального елемента послідовності, при знаходженні суми чисел послідовності і т.д.

float array[20];

...

cout << max (array) << '\ n';

Передача масиву в функцію як формального параметра має кілька особливостей.

По–перше, масив не можна передати у функцію за значенням або за посиланням, масив у функцію може бути переданий тільки як покажчик на його нульовий елемент. Отже, заголовок функції знаходження максимального елемента масиву може бути описаний різними способами

*float max (float * mas)*

float max (float mas[])

float max (float mas[20])

По-друге, розмір масиву не є частиною параметра, і функція не знає реального розміру переданого масиву, передати в функцію розмір масиву це завдання програміста, це можна зробити, наприклад, використовуючи ще один формальний параметр

*float max (float * mas, int size)*

Параметр *size* дозволяє задати реальний розмір переданого масиву.

По-третє, так як формальним параметром є покажчик, то операції з масивом в функції виконуються не з локальною копією, а з вихідної змінної. Отже, якщо немає необхідності всередині функції міняти значення елементів масиву, то можна використовувати константних покажчик. Дані, на які він вказує, не можуть бути змінені.

*float max (const float * mas, int size)*

Функція з масивом як параметр може бути реалізована так

```
float max (const float * mas, int size)
{
  float res = mas[0];
  for (int j = 0; j <size; ++ j)
        if (mas [j]> res)
                res = mas [j];
  return res;
}
```

Функція може бути викликана в такий спосіб

```
float array[20];
...
cout << max(Array, sizeof (array) / sizeof (array [0])) << '\ n';
```

Тут другий фактичний параметр обчислюється автоматично. Функція *sizeof* (*array*) повертає розмір всього масиву *array* [20] в байтах, функція *sizeof* (*array* [0]) повертає розмір одного елемента масиву в байтах. Ставлення цих значень дорівнює кількості елементів масиву.

У наступному прикладі функція використовується для перебору елементів масиву і змінює знак кожного елемента на протилежний.

*void neg (float * m, int size)*

```
{
  for (int j = 0; j <size; ++ j)
        m [j] = –m [j];
}
```

Елементи масиву змінюються всередині функції, тому формальним параметром є звичайний неконстантний покажчик. Виклик цієї функції наступний

```
neg (array, sizeof(Array) / sizeof (array [0]));
```

Багатовимірні масиви також можна використовувати в якості формальних параметрів. При цьому масив всередині функції розглядається як одновимірний.

```
float matrix[2] [3] = {
        {1, 2, 3},
        {11, 12,13}
        };
neg(& Matrix[0] [0], sizeof (matrix) / sizeof (matrix [0] [0]));
```

Слід також розглянути передачу рядків в якості формальних параметрів. Рядок являє собою символьний масив. На відміну від масивів інших типів розмір рядка при передачі її в функцію може бути визначений. Рядок закінчується символів з ASCII кодом '\ 0'. У прикладі приведена функція, яка підраховує кількість зазначених символів в рядку.

```
int char_count (const char * string, char c)
{
  int count = 0, j = 0;
  while (string [j]! = '\ 0')
  {
        if (string [j] == c)
              count ++;
        j ++;
  }
  return count;
}
void main ()
{
  ...
```

```
    int cn = char_count ( «AAAA», 'A');
  ...
}
```

Перший формальний параметр містить вихідну рядок. Другий параметр задає символ, який треба шукати в рядку. Використовуючи цикл з передумовою виконується перебір всіх символів рядка до символу з кодом '\ 0'. При досягненні цього символу цикл закінчується. У кожній ітерації циклу поточний символ рядка порівнюється з шуканим. Якщо ці символи збігаються, то инкрементируется лічильник символів локальна змінна count. Мінлива cn після виконання функції отримає значення 4.

6.3. Прототипи функцій

Раніше було відзначено, що функція повинна бути описана до її першого використання. Тому, у всіх наведених прикладах спочатку розглядалася реалізація функції, а потім її виклик. У деяких випадках неможливо дотримати таку структуру програми. Тоді використовують прототипи функцій. Прототипом функції в мові C/C ++ називається оголошення функції, яке не містить тіло функції, але вказує ім'я функції, кількість і типи формальних параметрів, і повертається функцією тип даних.

```
void func (int arg1, float arg2);//прототип функції

void main()
{
  ...
  float x;
  func(20, x);                    //виклик функції
  ...
}
void func(Int arg1, float arg2)  //реалізація функції
{
  // тіло функції
}
```

Також прототипи функцій часто використовуються в заголовних файлах для можливості організації програмного інтерфейсу. Наприклад,

розробляється програма, яка виконує обробку зображень. З точки зору архітектури такого додатка доцільно розділити функції, які працюють з файлами зображень, і функції, які дозволяють працювати з зображеннями вже завантаженими в пам'ять. З точки зору користувача такого додатка повинна бути можливість завантажити зображення з графічного файлу і виконати деяку операцію з цим зображенням, наприклад, поворот цього зображення на деякий кут. Програмний модуль, який реалізує завантаження графічних файлів, називається file_manager і реалізація цього модуля буде виконана в файлі *file_manager.cpp*.

Файл *image_proc.h*

```
...
// прототип функції
void rotate (DWORD * pimage, int height, int width, double angle);
...
```

файл image_proc.cpp

```
// включення заголовки
#include «image_proc.h»
...
// реалізація функції
void rotate (DWORD * pimage, int height, int width, double angle)
{
  // тіло функції
}
```

файл file_manager.cpp

```
// включення програмного модуля image_proc
#include «image_proc.h»
void main ()
{
  ...
  DWORD * pimg;
  char * fname;
  ...
  if (load_file (fname, pimg))
  {
        ...
        rotate (pimg, h, w, pi);
```

```
        ...
    }
    ...}
```

Функція повороту зображення *rotate*() реалізована в файлі *image_proc.cpp*. Функція має три формальних параметра. Зображення являє собою двовимірний масив, кожен елемент якого це колір деякого пікселя на зображенні. Для передачі масиву в функцію використовується покажчик. Далі передаються два параметри, які задають висоту і ширину зображення. Останній параметр дозволяє задати кут повороту зображення. Для того щоб функція могла бути викликана в іншому програмному файлі необхідно включити її прототип в заголовки програмного модуля *image_proc*. Також необхідно включити заголовки *image_proc.h* в файл реалізації *image_proc.cpp* директивою *define*. Таким же чином програмний модуль *image_proc* підключається до модуля *file_manager*. Після чого функцію *rotate* () можна використовувати.

6.4. Перевантаження функцій

У мові С ++ є можливість використовувати однакові імена для різних функцій. Така необхідність виникає, коли, наприклад, потрібно виконувати однакові дії над змінними різних типів. Потрібно написати функції, які сортує масиви елементів різних типів. Функції можуть бути описані як

```
void int_sort (int * parr, int size);
void double_sort (double * parr, int size);
void float_sort (float * parr, int size);
void long_sort (long * parr, int size);
```

Логічніше було б мати однакові назви функцій для перерахованих типів. З точки зору користувача, виконується лише одна операція – сортування масиву, а деталі її реалізації великого інтересу не представляють.

Таку можливість дає механізм перевантаження функцій. У С ++ двом або більше функцій може бути дано одне й те саме ім'я за умови, що їх списки параметрів розрізняються або числом параметрів, або їх типами. Реалізація матиме такий вигляд

```
// прототипи функцій
```

```
void sort (int * parr, int size);
void sort (double * parr, int size);
void sort (float * parr, int size);
void sort (long * parr, int size);
// реалізація функцій
void sort (int * parr, int size)
{
  // тіло функції
}
void sort (double * parr, int size)
{
  // тіло функції
}
void sort (float * parr, int size)
{
  // тіло функції
}
void sort (long * parr, int size)
{
  // тіло функції
}
```

Рекурсія. Рекурсією називається конструкція, коли деяка функція викликає саму себе. Прикладом рекурсивного вираження є обчислення факторіала

$n! = 1 * 2 * ... (n–1) * n$

З іншого боку можна помітити, що

$(N–1)! = 1 * 2 * ... * (n–1)$

і, отже, обчислення n! можна поставати як

$n! = (N–1)! * n$

Це є рекуррентная формула для обчислення факторіала деякого цілого числа. Тобто можна написати наступну функцію.

```
int factorial (int n)
{
  if(N> 1)
```

```
        return factorial(N – 1);
  else
        return1;
}
```

Тут для обчислення значення функції для параметра n викликається ця ж функція для параметра (*n*–1). Але рекурсивна програма не може викликати себе нескінченно, інакше вона ніколи не зупиниться, таким чином в програмі (функції) повинна бути присутнім ще один важливий елемент – так зване термінальне умова, тобто умова при якому програма припиняє рекурсивний процес. У прикладі такою умовою є повертається функцій значення при *n* == 1.

6.5. Час життя змінних, класи пам'яті

Раніше розглядалася структура програми. Звичайна програма являє собою визначення функції main, яка для виконання необхідних дій викликає інші функції. Зв'язок між функціями здійснювалася за даними за допомогою передачі параметрів і повернення значень функцій. Також розглядалося, що компілятор мови дозволяє також розбити програму на кілька окремих частин (вихідних файлів), транслювати кожну частину окремо, і потім об'єднати всі частини в один виконуваний файл за допомогою редактора зв'язків.

Для того щоб визначається функція могла виконувати будь–які дії, вона повинна використовувати змінні. Всі змінні повинні бути оголошені до їх використання. Оголошення встановлюють відповідність імені та атрибутів змінної, функції або типу. Визначення змінної викликає виділення пам'яті для зберігання її значення. Клас виділяється пам'яті визначається специфікатором класу пам'яті, і визначає час життя і область видимості змінної, пов'язані з поняттям блоку програми.

Блоком вважається послідовність оголошень, визначень і операторів, укладена в фігурні дужки.

```
{
      // блок
      int j;
      cin >> j;
```

```
        func(j, sizeof (j));
        ...
}
```

Існують два види блоків – складовою оператор і визначення функції, що складається з складеного оператора, що є тілом функції, і попереднього тілу заголовка функції (в який входять ім'я функції, типи значення, що повертається і формальних параметрів). Блоки можуть включати в себе складові оператори, але не визначення функцій. Внутрішній блок називається вкладеним, а зовнішній блок – осяжний.

Час життя – це інтервал часу виконання програми, протягом якого програмний об'єкт (змінна або функція) існує. Час життя змінної може бути локальним або глобальним. Змінна з глобальним часом життя має розподілену для неї пам'ять і певне значення протягом усього часу виконання програми, починаючи з моменту виконання оголошення цієї змінної. Змінна з локальним часом життя має розподілену для нього пам'ять і певне значення тільки під час виконання блоку, в якому ця змінна визначена або оголошена. При кожному вході в блок для локальної змінної розподіляється нова пам'ять, яка звільняється при виході з блоку. Всі функції мають глобальне час життя і існують протягом усього часу виконання програми. Глобальні змінні завжди инициализируются,

```
float eps = 0.001f;
void calc_time();
void main()
{
        ...
        calc_time();
...}
void calc_time()
{
  float interval = 0;
  ...
  if(interval < eps)
  {
        ...
```

}
...}

У прикладі описана глобальна змінна eps, яка може використовуватися в будь-якому місці програми. Така змінна створюється при запуску програми і знищується перед завершенням програми. У функції *calc_time* () створюється локальна змінна interval. Дана змінна доступна тільки всередині функції.

Також існує поняття області видимості. Область видимості – це частина тексту програми, в якій може бути використаний даний об'єкт. Об'єкт вважається видимим в блоці або в вихідному файлі, якщо в цьому блоці або файлі відомі ім'я і тип об'єкту. Об'єкт може бути видимим в межах блоку, вихідного файлу або у всіх вихідних файлах, що утворюють програму. Задавати область видимості можна спеціальними модифікаторами, які задають класи пам'яті для даного об'єкта.

Найпоширенішим випадком є використання автоматичних змінних. Автоматична змінна має локальний час життя і видно тільки в блоці, де вона оголошена. При повторному входженні в блок програми для цієї змінної може бути виділена інша область пам'яті. Автоматична змінна задається модифікатором auto.

auto double velocity;

Модифікатор auto є модифікатором за замовчуванням для неініціалізируемих змінних, тому його можна не ставити. Тобто якщо опис змінної не містить модифікатора, що визначає клас пам'яті цієї змінної, а також не містить ініціатора, то така змінна є автоматичною. Змінна з класом пам'яті auto автоматично НЕ ініціалізується. Вона може бути проініціалізувати явно при оголошенні шляхом присвоєння їй початкового значення. Значення неініціалізованої змінної з класом пам'яті auto вважається невизначеним.

Змінні, оголошені на внутрішньому рівні зі специфікатором класу пам'яті static, забезпечують можливість зберегти значення змінної при виході з блоку і використовувати його при повторному вході в блок. Така змінні має глобальне час життя і область видимості всередині блоку, в якому вони оголошені. У наступному прикладі статична змінна calls

використовується як лічильник викликів деякої функції func (). Функція повертає це значення.

```
void main ()
{
        ...
}
int func ()
{
 ...
 static int calls =0;
 ...
 return ++ calls;
}
```

Змінні класу пам'яті static можуть бути ініційовані константним виразом. Якщо явною ініціалізації немає, то такий змінної присвоюється нульове значення. Ініціалізація виконується один раз при першому вході в блок. Якщо змінна оголошена без вказівки класу пам'яті, але з явною ініціалізацією, то такий змінної за замовчуванням присвоюється клас пам'яті static.

Змінна, оголошена локально з класом пам'яті extern, є посиланням на змінну з тим же самим ім'ям, певну глобально в одному з вихідних файлів програми. Мета такого оголошення полягає в тому, щоб зробити визначення змінної глобального рівня видимим всередині цього блоку. Тобто модифікатор використовується, якщо є змінна, оголошена глобально в цьому ж програмному файлі або в іншому програмному файлі, але цю змінну необхідно використовувати раніше, ніж вона описана.

```
void main ()
{
 ...
 extern char * filename;
 ...
 int len = strlen (filename);
 ...}
char filename[20] = { '\ 0'};
void set_filename ()
{
```

cout << «Введіть ім'я файлу» << '\ n';
cin >> filename;
...}

Оголошення змінної зі специфікатором extern інформує компілятор про те, що пам'ять для змінної виділяти не потрібно, так як це виконано десь в іншому місці програми. Специфікатор класу пам'яті extern для глобальних змінних використовується, як і для локального оголошення, в якості посилання на змінну, оголошену в іншому місці програми, тобто для розширення області видимості змінної. При такому оголошенні область видимості змінної розширюється до кінця вихідного файлу, в якому зроблено оголошення. В оголошеннях з класом пам'яті extern не допускається ініціалізація, так як ці оголошення посилаються на вже існуючі та визначені раніше змінні. Змінна, на яку робиться посилання за допомогою специфікатор extern, може бути визначена тільки один раз в одному з вихідних файлів програми.

Оголошення своїх типів. У мові С є можливість визначати свої типи даних і використовувати їх потім в програмах. Новий тип може бути створений як на базі вбудованого типу, так і певного раніше. Так, відомо, що для представлення даних в обчислювальній техніці використовується структура, що складається з 8 двійкових розрядів – біт, яка називається байт. Кількість значень, які можна закодувати байтом невелика – 256. Для кодування більшого числа значень широко використовуються такі структури як машинне слово і подвійне машинне слово. Машинне слово складається з 2 байт (16 біт) і дозволяє задати 65536 (216) значень, а подвійне машинне слово складається з 4 байт (32 біта) і дозволяє задати 4294967296 (232) значень. Ці типи даних є беззнаковими, тобто машинне слово, наприклад, задає діапазон значень 0..65535. Використовуючи стандартні типи мови в програмах можна працювати з такими структурами. Так вбудований тип long має розмір 4 байта і його можна описати як беззнаковий, застосувавши модифікатор unsigned. У прикладі виконано опис змінної для зберігання деякого кольору і змінна инициализирована білим кольором.

unsigned long color = 0xFFFFFFFF;

Такий опис не завжди зручно, тобто містить багато символів. У загальному випадку опис типу виглядає так

typedef ісходний_тіп новий_тіп

В *MS Visual C++* в файлі windef.h описані *WORD* і *DWORD* для описів машинного слова і подвійного машинного слова

typedef unsigned short WORD
typedef unsigned long DWORD

Підключивши цей заголовки змінну для обробки кольору можна описати так

DWORD color = 0xFFFFFFFF;

Практичні завдання

1. Ввести з клавіатури число *N*. Створити масив, що складається з цифр даного числа. Створити функцію знаходження мінімального елемента створеного одновимірного масиву, перевірити чи є отримане число простим.

2. Для заданого числа *N*>1 побудувати послідовність чисел Фібоначчі, які змінюються за законом *A* (0) = 0; *A* (1) = 1; *A* (*i*) = *A* (*i*–1) + *A*(*i*–2). Створити функцію, що визначає значення третього парного числа.

3. Для заданого числа *N* отримати всі скоєні числа менше *N*. Створити функцію знаходження суми отриманих чисел.

4. Дан двовимірний масив цілих чисел. Поміняти місцями максимальний елемент масиву і мінімальний елемент його головної діагоналі. Створити функцію знаходження суми елементів другого рядка двовимірного масиву.

5. Поміняти місцями рядок і стовпець квадратної матриці, на перетині яких знаходиться мінімальний елемент масиву. Створити функцію сортування по спадаючій другого шпальти матриці.

6. Створити двовимірний масив розміром 10*15. Створити одновимірний масив, що складається з максимальних елементів стовпців двовимірного масиву. Створити функцію сортування отриманого одновимірного масиву по зростанню.

7. Створити двовимірний масив 10*10. Поміняти місцями максимальний і мінімальний елементи масиву. Створити функцію, знаходження суму елементів головної діагоналі масиву.

8. Створити одновимірний масив, що складається з подільників цілого числа n, що вводиться з клавіатури. Створити функцію, яка перетворює отриманий масив в зворотний.

9. Створити двовимірний масив розміром 10*15. Знайти суму елементів кожного рядка двовимірного масиву і записати ці суми в одновимірний масив. Створити функцію знаходження максимального елемента отриманого масиву.

10. У двовимірному масиві визначити добуток суми індексів двох максимальних елементів не головною діагоналі. Створити функцію сортування останнього стовпця масиву по спадаючій.

11. Дана прямокутна матриця. Поміняти місцями рядок, що містить елемент з максимальним значенням, з рядком, що містить елемент з найменшим значенням. Створити функцію сортування першого стовпчика по спадаючій.

12. У двовимірному масиві знайти суму максимальних значень елементів її рядків. Створити функцію перетворення двовимірного масиву в одновимірний.

13. Заданий одновимірний масив цілих чисел ділиться на три частини двома елементами: максимальним і мінімальним. Визначте суму елементів в кожній частині масиву. Використовуйте функцію для знаходження індексів мінімального і максимального елементів і підрахунку суми елементів в заданій частині масиву.

14. Дано два натуральних числа. З'ясувати, в якому з них більше цифр. Для розрахунку кількості цифр в числі визначити функцію.

15. Дано дві послідовності цілих числі: a_1, a_2, a_3, ..., a_n і b_1, b_2, ..., b_n. Знайти кількість парних чисел в першій з них і кількість непарних у другій. Для вирішення завдання визначити функцію, що дозволяє розпізнавати парні числа.

16. Отримати числа Армстронга, що складаються з трьох цифр. Число з N цифр є числом Армстронга, якщо сума його цифр, зведених в N–ю ступінь, дорівнює самому числу (наприклад, 153 = 1 + 5 + 3).

Контрольні питання

1. Функції в програмі. Призначення.

2. Структура функції. Опис і використання функцій. Прототипи функцій.

3. Виклик функції в головній програмі.

4. Безтіповие функції, їх виклик.

5. Призначення return в функції.

6. Передача значень в функцію. Формальні параметри.

7. Передача в функцію формальних параметрів за значенням

8. Передача формальних параметрів за вказівником і по посиланню.

9. Рекурсія. Призначення, принцип роботи.

10. Передача одновимірних масивів у функцію.

11. Передача двовимірних масивів у функцію.

12. Використання масивів як формальних параметрів. Особливості використання.

13. Прототипи функцій в заголовних файлах.

14. Перевантаження функцій. Inline функції.

15. Поняття області видимості. Час життя змінної. Класи пам'яті. Призначення, використання.

РОЗДІЛ 7.

ФАЙЛИ І СТРУКТУРОВАНІ ТИПИ ДАНИХ

7.1. Структури даних

Рішення обчислювальних завдань дуже часто призводить до використання наборів даних, що мають досить складну логічну організацію. Такими, наприклад, є всілякі каталоги, статистичні таблиці, інформаційні бюлетені та набори даних, що створюються компілятором при обробці програм на мовах високого рівня. У той же час, оперативна пам'ять ЕОМ організована вкрай примітивно в логічному відношенні, будучи послідовність занумерованих осередків довгою в один байт кожна. Така невідповідність логічної складності структур даних і можливостей їх комп'ютерного уявлення викликає значні труднощі при розробці програмного забезпечення. Можливий шлях розв'язання виникаючої проблеми полягає в створенні мов програмування, що підтримують таку логічну організацію даних, яка найбільш типова для широкого кола прикладних задач. До подібних мов відноситься і мова С, що володіє надзвичайно потужними засобами подання та обробки складних агрегатів даних. Одним із способів реалізації структурних логічних конструкцій є використання одно– і багатовимірних масивів змінних, які визначаються як впорядковані послідовності елементів даних одного типу. З іншого боку масив можна розглядати як єдину логічну структуру, яка містить осередків однакового типу. Однак статичність і однорідність масивів в тому вигляді, як вони були визначені, дещо обмежує можливість їх застосування для опису внутрішніх логічних зв'язків реальних інформаційних систем. І, все таки, масиви можна розглядати як структурований тип даних. І змінна такого типу – це змінна, яка містить кілька значень, тобто деяку інформаційну структуру.

Крім того в мові С можна використовувати більш складні агрегати даних, звані структурами, для яких вимога однорідності вже не має значення.

Під структурою в мові С розуміється набір однієї або більшого числа змінних, можливо мають різний тип і об'єднаних під єдиним ім'ям, званим ім'ям структури. Опис структури має такий вигляд

```
struct
{
  тип поле1;
  тип поле2;
  ...
} ідентифікатор1, ідентіфікатор2, ...;
```

В результаті такого опису будуть створені змінні з іменами *ідентіфікатор*1, *ідентіфікатор*2 і т.д. Кожна така змінна буде містити кілька значень. Кожне значення зберігається в параметрі поле, а кожне поле може мати власний тип. Поля одного типу можуть об'єднуватися в списку, як при описі змінних

```
struct
{
  тип поле1, поле2, поле3;
  тип поле4;
  ...
} ідентифікатор1, ідентіфікаторя2, ...;
```

У прикладі описана структура даних для роботи з датою

```
struct
{
  int year, month, day;
  char week_day[3];
  char month_name[10];
  bool work_day;
} Date;
```

Такий опис створить змінну *date*. Мінлива використовується для зберігання дати та містить такі значення:

- рік в числовому форматі, наприклад 2009 року;
- місяць в числовому форматі, наприклад 10 (жовтень);
- число місяця, наприклад 16;
- назва для тижні в форматі «Пн», «Вт», «Ср» і т.д .;

- назва місяця в форматі «Январь», «Февраль», «Март» і т.д .;
- поле логічного типу, яке дорівнює true, якщо день є робочим і false, якщо вихідним.

На рис. 6.1 схематично представлено зберігання в пам'яті змінної date. Розмір, який займає структурована змінна, дорівнює сумі розмірів складових її полів.

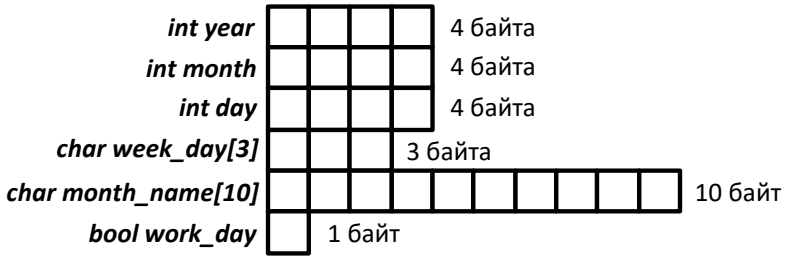


Рисунок 6.1 – Зберігання в пам'яті структурованої змінної

Доступ до полів структури має такий синтаксис

ідентіфікатор.поле

Для роботи з описаної вище змінної date в програмі можна виконувати будь-які операції, як зі звичайною змінною, вказуючи при цьому необхідні поля.

date.year = 2010 року;
date.workday = true;
strcpy (date.week_day, «Пт»);
cout << date.month_name << '\ n';

Окремі структури з довільним загальним шаблоном, як і звичайні змінні будь–якого типу, можуть бути об'єднані в масиви фіксованої довжини. Описи масивів структур в програмі будуються на тій же самій синтаксичній основі, що і опису звичайних масивів.

```
struct
{
  int num;
  char fio[30];
```

```
  float weght;
  int math, phiz, alg;
} ap17a [21], ap17b [18], ap57 [23];
```

У прикладі створені масиви для зберігання списків студентів трьох груп. Розмір кожного масиву відповідає фактичній кількості студентів в групі. Елементи всіх масивів мають однаковий шаблон, і кожен елемент містить інформацію про один студента:

- номер студента за списком в журналі – поле *num*;
- прізвище студента – поле *fio*;
- вага студента – поле *weight;*
- оцінки з математики, фізики та програмування за останню сесію – поля *math, phiz* і *alg* відповідно.

Можна знайти середню вагу студентів групи АП–17Б

```
float aver_weight = 0.f;
for (int j = 0; j <18; ++ j)
  aver_weight + = ap17b [i] .weight;
aver_weight / =18;
```

Цей найпростіший спосіб визначення структур вимагає щоразу повторювати шаблон структури в кожному новому описі, навіть якщо ці описи визначають структури з однією і тією ж загальною схемою. Щоб уникнути такого повторення, мова С надає можливість забезпечити шаблон створюваної структури деяким ім'ям, яке називається тег структури. Синтаксис опису структури з тегом такий

```
struct тег
{
  тип поле1;
  тип поле2;
  ...
} ідентифікатор1, ідентіфікатор2;
```

У цьому випадку структура отримає ім'я, і будуть описані змінні цієї структури. Після такого визначення в програмі, тег структури може бути використаний як синтаксичний еквівалент імені типу даних для опису змінних.

```
struct тег ідентифікатор3;
```

Структура *ідентіфікатор*3 матиме ті ж поля, що і змінні *ідентіфікатор*1 і *ідентіфікатор*2. У програмі можна виконати просто опис тега структури без опису змінних. Змінні можуть бути описані пізніше. Також можна опустити ключове слово struct при опис структурованих змінних за допомогою тега.

```
struct тег
{
  тип поле1;
  тип поле2;
  ...
};
```

*тег ідентифікатор*1, *ідентіфікатор*2, *ідентіфікатор*3;

У прикладі визначена структура для опису точки в тривимірній системі координат масив з 10 таких точок:

```
struct point3d
{
  double x, y, z;
};
point3d points[10];
```

Окремо необхідно розглянути роботу зі структурами з використанням покажчиків. Така необхідність може виникнути, наприклад, при передачі в функцію параметра через покажчик. Якщо структурована змінна є покажчиком, то синтаксис доступу до полів цієї змінної наступний

(* Покажчик) .полі

Також є альтернативний синтаксис цієї операції, який використовується частіше

указатель–> поле

У прикладі реалізована функція, яка дозволяє заповнити описану вище структуру дати.

```
struct s_date
{
  int year, month, day;
  char week_day[3];
  char month_name[10];
  bool work_day;
};
void set_date(int _y, int _m, int _d, char * pwd, _char * _pmn, bool _wd,
s_date * d)
{
  d–>year = _y;
  d–> month = _m;
  d–> day = _d;
  strcpy(D–> week_day, _pwd);
  strcpy(D–> month_name, _pmn);
  d–> work_day = _wd;}
void main()
{
  s_date date;
  set_date(2009, 12, 30, «Ср», «Грудень», true, & date);
  ...}
```

Структура визначена з використанням тега *s_date*. Це зроблено для того, щоб опис структури було глобальним і його можна було використовувати як в основній програмі, так і в функціях. Мінлива date описана в основній програмі. Далі викликається функція *set_date*() і в неї передається адреса змінної *date*. Формальним параметром, який відповідає змінної *date*, є покажчик *d*. Власне в функції *set_date*() значення формальних параметрів присвоюються відповідним полів структури.

Вкладені структури. Мова C/C++ допускає вкладеність структур. Це дозволяє створювати досить складні логічні структури даних. Наприклад, необхідно визначити структуру, яка описує коло. Центр кола це точка в двовимірному просторі і її можна описати як структуру. Опис окружності можна виконає різними способами.

```
struct
{
  struct
  {
        double x, y;
  } center;
  double radius;
} Circle;
```

Тут описана структурована змінна sphere, яка містить два поля center і radius. Причому поле center також є структурою, що містить 3 координати центру сфери. Можна також описати поле center з використанням тега структури.

```
struct s_center
{
  double x, y;
};
struct
{
  s_center center;
  double radius;
} Circle;
```

Також з використанням тега структури можна визначити і саму сферу.

```
struct s_center
{
  double x, y;
};
struct s_circle
{
  s_center center;
  double radius;
};
s_circle circle;
```

Останній варіант є кращим, тому що він більш наочний і його зручніше використовувати в програмі.

Доступ до полів вкладених структур виконується так само, як до полів звичайних структур. Імена полів вкладених структур поділяються точками.

*ідентіфікатор.поле*1.*поле*2.*поле*3

У прикладі виконується обчислення точки, в якій окружність перетинає вісь Оу. Вирішується завдання наступним чином. Можливі три варіанти розташування кола в системі координат:

1. Якщо виконується умова $x_c > R$, то коло не перетинає вісь OY (рис. 7.2).

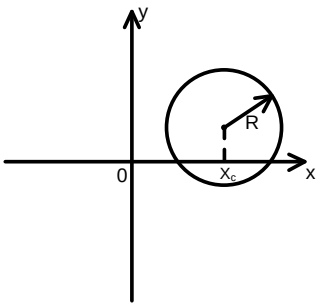


Рисунок 7.2 – Розташування окружності, якщо $x_c > R$

2. Якщо виконується умова $x_c = R$, то окружність перетинає вісь OY у точці y_c (рис. 7.3).

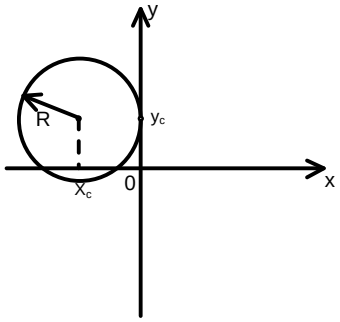


Рисунок 7.3 – Розташування окружності, якщо $x_c = R$

3. Якщо виконується умова $x_c < R$, то окружність перетинає вісь OY у двох точках y_1 і y_2 (рис. 7.4). Обчислити координати точок можна з прямокутного трикутника $y_1 y_c x_c$, використовуючи теорему Піфагора.

Знаючи довжину катета $|y_1y_c|$, координати шуканих точок буду рівні $y_1=y_c+|y_1y_c|$, $y_2 = y_c - |y_1y_c|$.

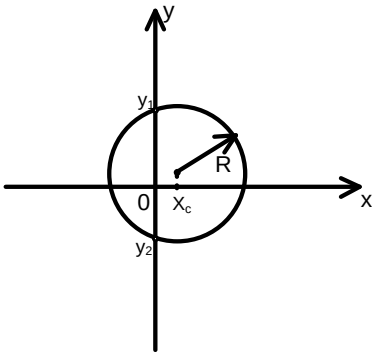


Рисунок 7.4 – Розташування окружності, якщо $x_c < R$

Програма, що реалізує запропонований алгоритм представлена нижче.

```
#include <iostream>
#include <math.h>
struct s_center
{
  double x, y;
};
struct s_circle
{
  s_center center;
  double radius;
};

void main ()
{
  s_circle circle;
  cout << «Введіть координату X» << '\ n';
  cin >> circle.center.x;
  cout << «Введіть координату Y» << '\ n';
  cin >> circle.center.y;
  cout << «Введіть радіус кола» << '\ n';
  cin >> circle.center.radius;
```

```
if (fabs(circle.center.x) > fabs (circle.center.radius))
{
    cout << «Коло не перетинає вісь Oy» << '\ n';
    return;
}
if(fabs (circle.center.x) == fabs (circle.center.radius))
{
    cout << «Коло перетинає вісь Oy у точці» << circle.center.y
<< '\ n';
    return;
}
if(fabs (circle.center.x) <fabs (circle.center.radius))
{
    double y1, y2;
    y1 = circle.center.y + sqrt (pow (circle.center.radius, 2) –
pow(circle.center.x, 2));
    y2 = circle.center.y – sqrt (pow (circle.center.radius, 2) –
pow(circle.center.x, 2));
    cout << «Коло перетинає вісь Oy в точках» << y1 << «і» << y2
<< '\ n';
    return;
 }
}
```

7.2. Перерахування

Мова С дозволяє створити змінну, для якої в описі можна перерахувати всі можливі її значення. Синтаксис опису такий

```
enum
{
 значення1,
 значення2,
 значення3,
 значення4,
 ...
} ідентифікатор1, ідентіфікатор2, ...;
```

Кожне значення в описі перерахування являє собою символічне ім'я, з якою пов'язано його числове значення цілого типу. Параметру *значення*1 буде за замовчуванням відповідати ціле *значення*0, параметру *значення*2 буде відповідати ціле *значення*1 і т.д. Змінні *ідентіфікатор*1, *ідентіфікатор*2 можуть приймати тільки перераховані значення.

Також як і для структур можливо опис тега перерахування.

```
enum тег
{
  значення1,
  значення2,
  значення3,
  значення4,
  ...
};
```

Тоді змінну можна описати як

```
enum тег ідентифікатор;
```

Ключове слово enum може бути опущено. Нижче описана змінна, за допомогою якої можна моделювати поведінку світлофора.

```
enum e_traffic_light
{
  TL_OFF,
  TL_FLASH_YELLOW,
  TL_RED,
  TL_YELLOW,
  TL_GREEN};
e_traffic_light tlight;
tlight = TL_RED;
...
switch(Tlight)
{
  case TL_OFF:
  case TL_FLASH_YELLOW:
        check_PDD();
        break;
```

```
  case TL_RED:
        stop();
        break;
  case TL_YELLOW:
        ready();
        break;
  case TL_GREEN:
        go();
        break;
}
```

Світлофор може бути в п'яти станах – вимкнений, блимати жовтим світлом, горіти червоним, жовтим або зеленим світлом. У перерахуванні цим станам відповідають значення TL_OFF, TL_FLASH_YELLOW, TL_RED, TL_YELLOW, TL_GREEN. Фактично описана нижче змінна *tlight* може приймати цілі значення в діапазоні 0..4 у вигляді символічних імен. Змінну можна використовувати, наприклад, в операторах присвоювання, перемикання і т.д. Так в операторі перемикачі перевіряється значення змінної. Якщо світлофор вимкнений або мигає жовтим, то викликається функція *check_PDD*(), в якій виконуються операції управління рухом деякого транспортного засобу відповідно правилами дорожнього руху – необхідно пропустити пішоходів, виконати вимоги дорожніх знаків і розмітки тощо Якщо світлофор горить червоним світлом, то викликається функція *stop*(), в якій виконуються операції зупинки транспортного засобу. Якщо світлофор горить жовтим світлом, то викликається функція *ready*(), в якій виконуються операції підготовки до руху транспортного засобу. Якщо світлофор горить зеленим світлом, то викликається функція *go*(), в якій виконуються операції руху транспортного засобу.

Можна також задавати константні значення елементів перерахування. У прикладі описано перерахування, яке можна використовувати для моделювання перемикача автоматичної коробки передач автомобіля. Перемикач може перебувати в таких положеннях:

- Parking – стоянка автомобіля;
- Neutral – нейтральний режим, наприклад, при короткочасній стоянці;
- Drive – режим руху;

- Low – режим руху на зниженій передачі в складних умовах;
- Reverse – режим заднього ходу.

Крім цих режимів можуть бути додаткові режими руху. З опису видно, що є кілька режимів руху і окремий режим парковки. Для обробки режимів руху зручно, щоб їх числові значення відрізнялися на 1. Значення режим парковки має відрізнятися від інших значно. Перерахування матиме такий вигляд.

```
enum e_autogearbox_lever
{
  AGL_PARKING = 100,
  AGL_REVERSE = –1,
  AGL_NEUTRAL,
  AGL_DRIVE,
  AGL_LOW
};
```

Для символічного імені AGL_PARKING явно задано числове значення 100, для символічного імені AGL_REVERSE явно задано числове значення –1. Числові значення для імен AGL_NEUTRAL, AGL_DRIVE, AGL_LOW на 1 більше попереднього, тобто символічного імені AGL_NEUTRAL відповідає значення 0, імені AGL_DRIVE відповідає значення 1 і імені AGL_LOW відповідає значення 2.

7.3. Об'єднання

Об'єднання – це засіб, що дозволяє розміщувати дані різних типів в одному і тому ж місці оперативної пам'яті. З точки зору граматики мови С, всяке об'єднання є змінної, що приймає в різний час виконання програми значення різних типів. Об'єднання можна вважати структурою, в якій в кожен момент часу можна використовувати для зберігання значення тільки одне поле. Синтаксис опису об'єднання наступний.

```
union
{
  тип поле1;
  тип поле2;
  ...
} ідентифікатор1, ідентіфікатор2, ...;
```

Схематично змінна типу об'єднання представлена на рис. 7.5. Розмір, який займає змінна типу об'єднання, дорівнює розміру максимального поля.

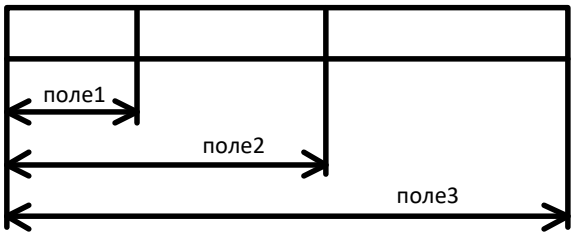


Рисунок 7.5 – Змінна типу об'єднання

Доступ до конкретного поля такий же, як для структур.

ідентіфікатор.поле

Тип значення, яке буде доступно за таким зверненням, відповідає типу параметра поле.

Також як і для структур можливо опис тега об'єднання.

```
union тег
{
  тип поле1;
  тип поле2;
  ...};
```

Тоді змінну можна описати як

union тег ідентифікатор;

Ключове слово *union* може бути опущено.

У прикладі реалізована передача значення довільного типу в деяку функцію.

```
enum e_val_type
{
  VT_INT,
  VT_LONG,
  VT_SHORT,
  VT_FLOAT,
```

```
  VT_DOUBLE};
union u_val
{
  int int_val;
  long long_val;
  short short_val;
  float float_val;
  double double_val;};
void func (u_val val, e_val_type type)
{
  switch(Type)
  {
        case VT_INT:
              // обробка значення типу int
              cout << val.int_val;
              ...
              break;
        case VT_LONG:
              // обробка значення типу long
              cout << val.long_val;
              ...
              break;
        case VT_SHORT:
              // обробка значення типу short
              cout << val.short_val;
              ...
              break;
        case VT_FLOAT:
              //обробка значення типу float
              cout << val.float_val;
              ...
              break;
        case VT_DOUBLE:
              // обробка значення типу double
              cout << val.double_val;
              ...
```

```
            break;
  }
}
void main()
{
  ...
  u_val param;
  param.float_val = 178.6f;
  func(Param, VT_FLOAT);
  ...}
```

У головній функції змінної param присвоюється значення типу float. Далі значення передається в функцію, в яку також передається тип значення, описаний в перерахуванні e_val_type. У самій функції перевіряється тип переданого значення і виконуватися ті операції, які необхідно виконати саме із значенням цього типу.

7.4. Файли даних і робота з ними

Файли даних являють собою іменовані набори даних на зовнішніх носіях інформації. Розміщення і доступ до файлів даних здійснюється за правилами операційної системи, де зберігаються файли. Цими правилами визначаються імена файлів, структура файлової системи і т.д.

Мова С має набір вбудованих функцій для доступу до файлів, що знаходяться на зовнішніх носіях. Модель файлу для роботи з ним з програми на мові С має вигляд, представлений на рис. 7.6

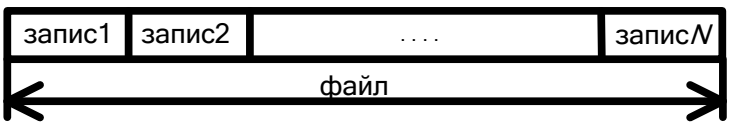


Рисунок 7.6 – Модель файлу даних

Відповідно до цієї моделі файл являє собою послідовність записів, що мають однакову структуру. Для роботи файл на зовнішньому носії повинен бути відкритий. При цьому поточної стає перша його запис. При операціях читання/запису положення поточного запису змінюється. Після

використання файл повинен бути закритий, що забезпечує коректність виконуваних з ним операцій.

Типи файлів.Файли бувають двох типів:

- текстові;
- виконавчі.

Різницю між цими типами файлів простіше зрозуміти на прикладі операції запису значення змінної в файл.

float val =12.9874*f*;

При записи значення змінної *val* в текстовий файл, в файл будуть виведені ASCII коди символів значущих цифр змінної і код десяткового дробу. При записи цього ж значення в двійковий файл будуть виведена цифрова інформація, яка відповідає внутрішньому формату зберігання змінної типу float.

Операції з файлами. Для роботи з файлом даних в програмі необхідно виконати відкриття цього файлу. Далі можна виконувати операції введення/виводу даних, а також операції перевірки положення покажчика файлу і операції переміщення покажчика по файлу. Після завершення операцій з файлом, файл повинен бути закритий.

Для виконання файлових операцій необхідно підключити бібліотеку stdio.h

#include <stdio.h>

Крім того необхідно створити файлову змінну (дескриптор файлу). Ця змінна має спеціальний тип і описується так

*FILE * файлова змінна;*

При відкритті файлу, файл на диску зв'язується з файлової змінної та всі наступні операції з ним виконуються з використанням файлової змінної.

Відкриття та закриття файлу. Операція відкриття файлу дозволяє зв'язати деякий файл на зовнішньому носії інформації з файлової змінної. Синтаксис операції наступний.

файлова змінна = fopen (ім'я, режим);

Функція має два параметри. Перший параметр є рядком, яка містить шлях і *ім'я файлу*. Другий параметр визначає, які операції можна буде виконувати з файлом після його відкриття. Якщо функція виконалася успішно, то файлова змінна приймає деяке валідності значення. Якщо файл з якихось причин не може бути відкритий, то функція повертає значення NULL. Таким чином, можна проводити перевірку результатів відкриття файлу.

Файл є рядком, яка може бути задана змінної або константою.

*char * f_name = «c: \\ ap17b \ lab1.cpp»;*

Параметр режим також є рядком, яка може містити такі символи. Перший символ рядка визначає, які операції можна буде виконувати з файлом і може бути наступним.

r – файл відкривається для читання, покажчик встановлюється на перший запис у файлі;

w – новий файл відкривається для запису, якщо такий файл вже існує, то він буде вилучений;

a – файл відкривається для запису, якщо файл вже існує, то інформація буде додаватися в файл, покажчик встановлюється на новий запис, яка розташована після останньої існуючої.

Другий символ вказує на тип файлу.

t – файл текстовий;

b – файл двійковий.

Крім того, рядок може містити символ '+', який вказує, файл відкривається як на читання, так і на запис.

Закриття файлу виконується викликом функції

fclose (файлова змінна);

У прикладі показано використання функцій відкриття і закриття файлу.

#include <stdio.h>

void main ()

```
{
  FILE * f;
  f = fopen( «Ap55.txt «,» rt «);
  if(F! = NULL)
  {
        // операції з файлом
        ...
        fclose(F);
  }
}
```

У прикладі оголошена файлова змінна f. Далі виконується відкриття текстового файлу «ap55.txt» для читання. Також виконується перевірка, успішно відкритий файл чи ні. Для цього після операції відкриття файлова змінна порівнюється зі значенням NULL. Якщо відкриття відбулося успішно, то виконуються операції з файлом. Після завершення всіх необхідних операцій викликається функція закриття файлу.

7.5. Файловий ввод/вивід

Основними операціями роботи з файлами є читання інформації з файлу і запис інформації в файл. Для текстових і двійкових файлів використовуються різні набори функцій.

Основною функцією для запису даних будь-яких типів в текстовий файл є функція *fprintf* (). Синтаксис функції наступний

fprintf (*файлова змінна*, «*формат*», *ідентифікатор*1, *ідентіфікатор*2, ...);

Цей синтаксис дуже схожий на синтаксис функції виведення тексту на екран *printf*. Параметр «*формат*» визначає тип даних, що виводяться і їх уявлення так само, як і у функції *printf*. Список змінних містить ідентифікатори змінних, значення яких будуть виводитися в файл. Відмінністю функцій є перший параметр, який вказує, в якій файл буде виводиться інформація. Для цього використовується дескриптор файлу. Для виведення рядків в текстовий файл використовується функція *fputs* (). Синтаксис функції наступний

fputs (*строка*, *файлова змінна*);

Параметр рядок є строковою константою або масивом символів. Параметр файловая_переменная визначає файл, в який буде виводитися рядок.

Введення даних з текстового файлу виконується функцією *fscans* ().

fscanf (*файлова змінна*, «*формат*», *ідентифікатор*1, *ідентіфікатор*2, ...);

Синтаксис функції схожий на синтаксис функції введення *scanf* (). Додатково в першому параметрі вказується дескриптор файлу, з якого буде введена інформація.

Нижче в прикладі виконується вивід в файл таблиці з результатами сесії студента. Дані про студента зберігаються в структурі і містять прізвище студента і оцінки з трьох предметів. Така структура була описана вище.

```
void main()
{
    FILE * of;
    char * fio[30];
    scanf( «% S», fio);
    ...
    // запис відомості в файл
    of = fopen( «Vedomost.txt», «wt»);
    if(of! = NULL)
    {
        for(int j = 0; j <21; ++ j)
        {
        if(strcmp ((char *) fio, ap17a [j] .fio) == 0)
        {
            //виводить таблицю
            fputs( «Відомість \ n», of);
            fputs( «Студент», of);
            fputs((Char *) fio, of);
            fputs( «\ N», of);
```

```
                fputs( «+ ––– + ––––––––––––––––– + ––––––– + \ n», of);
                fputs( «| N | Предмет | Оцінка | \ n», of);
                fputs( «+ ––– + ––––––––––––––––– + ––––––– + \ n», of);
                fprintf(Of, «| 1 | Математика |% d | \ n», ap17a [j] .math);
                        fprintf(Of, «| 2 | Фізика |% d | \ n», ap17a [j] .phiz);
                fprintf (of, «| 3 | Інформатика | % D | \ n «, ap17a[J] .alg);
                        break;
                }
                }
                fclose(Of);
        }
}
```

У прикладі вводиться прізвище студента. Далі проводиться пошук студента на прізвище в масиві ap17a. Якщо студент з таким прізвищем знайдений, то в файл виводиться таблиця з його оцінками.

Для двійкових файлів запис даних виконується функцією *fwrite* ().

fwrite (*покажчик*, *размер_блока*, *кол_блоков*, *файлова змінна*);

Перший параметр покажчик вказує на блок даних, який буде записаний у файл. Другий параметр размер_блока вказує розмір цього блоку даних в байтах. Третій параметр кол_блоков задає кількість блоків даних, які будуть записані в файл. Четвертий параметр є файлової змінної, пов'язаної з файлом, в який буде записуватися інформація. Після запису інформації в файл покажчик на файл автоматично переміщається на наступний запис.

Читання інформації з довічних файлів виконується функцією *fread* (). Її синтаксис аналогічний синтаксису функції *fwrite* ().

fread(*покажчик*, *размер_блока*, *кол_блоков*, *файлова змінна*);

Після читання інформації з файлу покажчик на файл автоматично переміщається на наступний запис. Розглянемо застосування функцій на прикладі читання інформації з графічного файлу. Графічний файл має просту структуру (рис. 7.7)

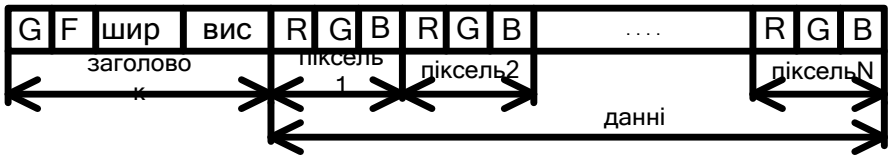


Рисунок 7.7 – Структура графічного файлу

У файлі є заголовок. Перші два байта файлу містять символи 'G' і 'F'. При читанні даних з файлу їх можна використовувати як ознака того, що файл має необхідний нам формат. Далі в файлі зберігаються два цілих беззнакових числа, які визначають розмір зображення в пікселях по горизонталі і по вертикалі. Для спрощення розмір зображення не може перевищувати 128x128 пікселів. Після заголовка в файлі розміщена інформація про квіти пікселів. Колір кожного пікселя складається з трьох складових – червоною, зеленою і синьою. Кожна складова може кодуватися значенням в діапазоні 0..255. Для зберігання прочитаної з файлу інформації використовуємо двовимірний масив *img_data*. Після відкриття файлу зчитується інформація про його заголовку в змінну *img_header*. Отримавши цю інформацію, можна обчислити розмір реальних графічних даних в фпйле. Цей розмір зберігається в змінної *size*.

```
struct s_header
{
  char sig1;
  char sig2;
  unsigned int img_width;
  unsigned int img_heght;
};
struct s_pixel
{
  unsigned short r, g, b;
};

void main ()
{
  s_header img_header;
```

```
s_pixel img_data[128] [128];
FILE * img_file;
img_file = fopen( «Image.dat», «rb»);
if (img_file! = NULL)
{
    fread(& Header, sizeof (s_header), 1, img_file);
    if(! ((Img_header.sig1 == 'G') && (img_header.sig2 == 'F')))
    {
        cout << «Неправильний формат файлу»;
        return;
    }
    else
    {
        unsigned int size = img_header.img_width * img_header.
img_heght;
        fread(Img_data, size * sizeof (s_pixel), size, img_file);
        // операції із зображенням
        ...
    }
 }
}
```

7.6. Інші операції з файлом

Крім перерахованих функцій файлового введення/виводу є функції, які дозволяють виконати з відкритим файлом деякі додаткові операції.

Запис інформації в файл виконується не відразу, а через деякий додатковий буфер. Такий підхід дозволяє не чекати закінчення повільних операцій запису даних в файл, а продовжувати виконання програми. У деяких випадках з міркувань збереження інформації необхідно примусово записати дані з проміжного буфера в файл. Ця операція виконується викликом функції

fflush(*файлова змінна*);

При виконанні функції *fclose*() дані з проміжного буфера записуються в файл. Але можлива ситуація, коли програма аварійно завершує роботу, при цьому функція *fclose*() не викликана і, отже, інформація з буфера може бути не збережена. Якщо проводиться запис в файл даних, збереження яких повинно бути виконано обов'язково, то після операції записи доцільно викликати функцію *fflush*(). У прикладі відкривається файл даних для запису і зв'язується зі змінною *ofile*. Після запису в файл деякою інформацією, що зберігається зі змінною *data*, викликається функція *fflush*, яка забезпечує примусову запис даних з проміжного буфера безпосередньо в файл на диску.

```
FILE * ofile;
ofile = fopen( «Out.dat», «wb»);
if(Img_file! = NULL)
{
  ...
  fwrite(data, sizeof (data), 1, ofile);
  fflush(ofile);
  ...
}
```

Функція *fseek* () надає користувачам можливість покажчик на записи у відкритому файлі, не виконуючи операції читання/запису. Синтаксис функції такий

fseek (*файлова змінна*, *зміщення*, *начальная_позіція*);

Параметр файловая_переменная визначає файл, в якому буде переміщений покажчик. Параметр зміщення задає кількість байт, на яке буде переміщений покажчик. Позитивне зміщення переміщує вказати до кінця файлу, негативний зсув переміщує покажчик на початок файлу. Параметр начальная_позіція задається константою, визначає, починаючи з якої позиції необхідно відраховувати зміщення, і може набувати таких значень:

SEEK_SET – зміщення відраховується від початку файлу;

SEEK_CUR – зміщення відраховується від поточної позиції вказівника;

SEEK_END – зміщення відраховується від останнього запису файлу.

У прикладі в файл записуються кілька 20 точок в двовимірному просторі, а потім зчитується 5 запис файлу в змінну *point.*

```
struct s_point
{
  float x, y;
};
...
s_point points[20];
s_ponit point;
...
FILE * points_file;
...
points_file = fopen( «Points.dat», «wb»);
if(points_file! = NULL)
{
  for(int j =0; j <20; ++ j)
  {
        fwrite(& points[j], sizeof (s_point), 1, points_file);
  }
  fflush(ofile);
  ...
  fseek(points_file, 5 * sizeof (s_point), SEEK_SET);
  fread(& point, sizeof (s_point), 1, points_file);
  ...
  fclose(points_file);
}
```

Функція *ftell*() дозволяє отримати поточну позицію покажчика у файлі. синтаксис функції

ftell(*файлова змінна*)

Функція повертає ціле значення, яке є номером запису, на яку встановлено вказівник файлу.

У прикладі функція використовується для отримання розміру файлу. Для файлу пов'язаного зі змінною *ifile* за допомогою функції *fseek* ()

покажчик встановлюється на останній запис. Це задається зсувом на 0 байт від кінця файлу. Далі функція *ftell* () повертає номер останнього запису, який, по суті, дорівнює розміру файлу.

```
fseek(Ifile, 0, SEEK_END);
int size = ftell(ifile);
cout << «Розмір файлу» << size << «байт» << '\ n';
```

Функція *feof* () дозволяє визначити, де знаходиться покажчик у відритому файлі – в середині файлу або вказує на кінець файлу. Функція повертає значення деякий нульове значення, якщо покажчик розташований в кінці файлу, і *значення* 0, якщо в середині. Використовуючи цю функцію можна прочитати всі записи з файлу, не знаючи заздалегідь їх кількість. У прикладі використовується описаний раніше файл, який містить координати набору точок. Кількість записів у файлі невідомо заздалегідь і програма зчитує всі наявні у файлі записи. Дані, які зчитуються з файлу, зберігаються в масиві *points* []. Розмір цього масиву заданий з надмірністю. Мінлива count після читання даних з файлу містить реальну кількість лічених записів. Мінлива res використовується для отримання результатів виконання функції *fread* (). Ця функція повертає розмір прочитаних даних. Коли буде досягнуто кінець файлу, функція поверне *значення* 0, тому що дані з файлу прочитані були.

```
s_point points[100];
FILE * ifile;
int count =0, res;
ifile = fopen( «Points.dat», «rb»);
if(ifile! = NULL)
{
        while(! feof (iof))
        {
                res = fread(& points [count], sizeof (s_point), 1, ifile);
                if(res)
                  ++ count;
  }
  ...
  fclose(Ifile);
}
```

cout << «Всього прочитано» << count << «записів» << '\ n';

– *int feof* (FILE * f)– функція перевіряє індикатор позиції файлу, щоб з'ясувати, чи досягнуто кінець файлу, пов'язаного елементом f. Якщо індикатор позиції файлу розташований в кінці файлу, повертається нульове значення, в іншому випадку повертається нуль.

– *long fteel* (FILE * f) – функція повертає поточне значення індикатора позиції файлу, для бінарних файлів – кількість байтів, які відокремлюють індикатор від початку файлу.

– *int remove* (ім'я файлу) – видаляє файл, при успішному видаленні функція повертає нуль.

– *void rewind* (FILE * f) – функція переміщує індикатор позиції файлу в початок файлу.

– *int fclose* (FILE * f) – функція закриває файл пов'язаний з файлової змінної f, якщо успішно закритий, то функція повертає нуль, в іншому випадку EOF.

Практичні завдання

1. Ввести в комп'ютер інформацію про студентів групи: прізвище, порядковий номер, отримує стипендію чи ні, середній бал за сесію. Вивести на екран повний список студентів. Вивести на екран список студентів, які отримують стипендію.
2. Сформувати список співробітників: прізвище, порядковий номер, місяць народження. Вивести на екран повний список співробітників. Вивести на екран інформацію про співробітників, які народилися в задані місяць.
3. Ввести в комп'ютер інформацію про заборгованість студентів (прізвище, кількість заборгованостей, предмет заборгованостей). Вивести на екран повний список студентів. Надрукувати список студентів в порядку зростання кількості заборгованостей.

4. Сформувати документ про фінансову діяльність підприємств (найменування підприємства, місяць, прибуток підприємства). Вивести на екран повний документ. Вивести на екран список підприємств в залежності від збільшення прибутку.

5. Ввести в комп'ютер інформацію про співробітників відділу (прізвище, вік, місяць народження). Вивести на екран повний список співробітників. Надрукувати прізвища співробітників, вік яких перевищує 28 років.

6. У відомості зберігається інформація про пропуски занять студентів (прізвище, факультет, група, кількість пропусків). Вивести на екран повний список студентів. Вивести на екран дані студента з найбільшому кількістю пропусків.

7. Створити документ про мікропроцесори зберігаються на складі: назва (Pentium, Celeron, Duron, Athlon), фірма (AMD, IDC, Texas Instrument, Motorola), частота, кількість контактів в корпусі. Вивести на екран повний список мікропроцесорів. Для найшвидшого процесора фірми AMD змінити назву на Duron.

8. Створити дані про файлах поточного каталога (ім'я файлу, розмір, атрибут, дата створення). Вивести на екран повний список файлів. Написати програму, яка для «системних» файлів змінює дату створення на дату виконання.

9. Ввести інформацію про транзисторах: назва (КТ315Г, КТ814В, КТ302, КТ853), потужність, максимальний робочий струм, кількість. Вивести на екран повний список транзисторів. Написати програму, яка знаходить транзистор з максимальною потужністю і вказує для нього кількість 200.

10. Ввести в комп'ютер дані про студентів групи: прізвище, порядковий номер, отримує стипендію чи ні, середній бал. Вивести на екран повний список студентів. Написати програму, яка виводить на екран інформацію про студентів в порядку убування середнього бала.

11. Сформувати документ, в якому зберігається інформація про асортимент продовольчих товарів в магазині: назва товару, код товару, кількість товару, ціна товару. Вивести на екран повний документ. Вивести на екран весь товар в порядку зростання ціни.

12. Ввести інформацію про пропуски занять студентами: прізвище, факультет, група, кількість пропусків. Вивести на екран повний список

студентів. Вивести на екран дані про студента, що має найбільшу кількість пропусків.

13. Вести в комп'ютер інформацію про студентів: прізвище, порядковий номер, місяць народження, отримує стипендію чи ні. Вивести на екран повний список студентів. Вивести на екран повідомлення про, те в якому місяці народилося найбільшу кількість студентів.

14. Створити документ про мікропроцесори зберігаються на складі: назва (Pentium, Celeron, Duron, Athlon), фірма (AMD, IDC, Texas Instrument, Motorola), частота, кількість контактів в корпусі. Вивести на екран повний список мікропроцесорів. Написати програму, яка змінює у мікропроцесора фірми AMD назву на Celeron.

15. Створити список, в якому зберігається інформація про факультети і їх академічної заборгованості: назва факультету, кількість заборгованостей. Вивести на екран повну інформацію про факультети. Надрукувати інформацію про факультети з максимальною і мінімальною заборгованостями.

16. Ввести в комп'ютер інформацію про користувачів бібліотеки: прізвище, рік народження, кількість книг на абонементі. Вивести на екран повну інформацію про користувачів бібліотеки. Надрукувати повідомлення про 5-ти користувачів, за якими числиться найбільшу кількість книг.

17. Створити документ про захворювання студентів: прізвище, рік народження, діагноз захворювання, період хвороби. Вивести на екран повну інформацію про захворювання студентів. Знайти студентів з діагнозом «грип» і змінити цей діагноз на «ГРВІ», вивести на екран.

18. Сформувати список, в якому зберігаються результати сесії групи (прізвище, середній бал, порядковий номер). Вивести на екран повну інформацію про студентів. Вивести на екран прізвища студентів, які отримуватимуть стипендію.

19. У звіті зберігається інформація: найменування підприємства, прибуток підприємства за місяць, нарахування на зарплату. Вивести на екран весь звіт. Відсортувати цей звіт в порядку зменшення прибутку і вивести на екран інформацію про 3-х найбільш прибуткові підприємства

20. Створити файл «*point*», який містить параметри для 12 точок в просторі. Параметрами кожної точки є: речові координати точки, колір точки. Дані повинні вводитися з клавіатури. Створити процедуру для

можливості перегляду користувачем створеного файлу даних. Для точок які мають зелений колір, вивести дані про їх координати на екран.

21. Створити файл «*process*», який містить опису для 8 процесорів, які застосовуються в ЕОМ. Параметрами кожного процесора є: назва, (Pentium, Celeron, Duron, Athlon, 486, 486DX2) фірма виробник (AMD, IDC, Texas Instrument), частота, кількість контактів в корпусі мікросхеми. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Для найшвидшого процесора фірми AMD змінити назву на Athlon.

22. Створити файл ім'ям «*worker*», що містить опису для 15 працівників підприємства. Параметрами кожного працівника є такі дані: прізвище та ініціали працівника; назва займаної посади; рік надходження на роботу; розмір отримуваної зарплати. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Вивести на екран посаду працівника підприємства, прізвище якого введена з клавіатури.

23. Створити файл «*student*», який містить інформацію про 10 студентів групи. Про кожного студента повинна зберігатися наступна інформація: прізвище, порядковий номер, отримує стипендію чи ні, середній бал за останню сесію. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Вивести на екран прізвища і номери студентів, які отримують стипендію і мають середній бал вище 4,8.

24. Створити файл «*files*», який містить параметри для 10 файлів поточного каталогу. Параметрами кожної файлу є: ім'я файлу, розмір файлу в байтах, атрибут, дата створення. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Для «архівних» файлів знайти найбільший і замінити його дату створення на дату виконання.

25. Створити файл «*tranz*», який містить параметри для 15 типів транзисторів, які є на складі. Параметрами кожного транзистора є: назва (КТ315Г, КТ814В, КТ302, КТ853А, КТ361А), потужність, максимальний робочий струм, кількість. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Знайти транзистор з максимальною потужністю і вказати для нього кількість 200.

26. Створити файл «*lines*», який містить параметри для 12 відрізків на площині. Параметрами кожного відрізка є: координати початку відрізка (цілі), координати кінця відрізка (цілі), перетинає чи відрізок вісь OX. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Для відрізків, які перетинають вісь OX обчислити загальну довжину і вивести на екран.

27. Створити файл з ім'ям «*note*», що містить опису для 15 записів з наступними полями: прізвище, ім'я, номер телефону, дата народження (масив з трьох чисел). Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Вивести на екран дату народження користувача, прізвище якого введена з клавіатури.

28. Створити файл «*point*», який містить параметри для 12 точок в просторі. Параметрами кожної точки є: речові координати точки, колір точки. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Визначити найдальшу точку від початку координат і вивести її параметри на екран.

29. Створити файл «*student*», який містить інформацію про 10 студентів групи. Про кожного студента повинна зберігатися наступна інформація: прізвище, порядковий номер, отримує стипендію чи ні, середній бал за останню сесію. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Вивести на екран інформацію про студентів в порядку убування середнього бала.

30. Створити файл «*files*», який містить параметри для 10 файлів поточного каталогу. Параметрами кожної файлу є: ім'я файлу, розмір файлу в байтах, атрибут, дата створення. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Для «прихованих» файлів знайти найменший і замінити його дату створення на дату виконання.

31. Створити файл з ім'ям «*znak*», що містить опису 10 записів з наступними даними: прізвище, ім'я; знак зодіаку; дата народження. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Вивести на екран дані про тих, хто народився під знаком стрілець.

32. Створити файл «*tranz*», який містить параметри для 15 типів транзисторів, які є на складі. Параметрами кожного транзистора є: назва (КТ315Г, КТ814В, КТ302, КТ853А, КТ361А), потужність, максимальні робочий струм, кількість. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Вивести на екран параметри транзисторів в порядку зростання робочого струму.

33. Створити файл «*lines*», який містить параметри для 12 відрізків на площині. Параметрами кожного відрізка є: координати початку відрізка (цілі), координати кінця відрізка (цілі), перетинає чи відрізок вісь OX. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Вивести на екран відрізки, які не перетинають вісь OX.

34. Створити файл «*point*», який містить параметри для 12 точок в просторі. Параметрами кожної точки є: речові координати точки, колір точки. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Для точок, які мають жовтий колір замінити його на зелений і вивести дані на екран.

35. Створити файл «*student*», який містить інформацію про 10 студентів групи. Про кожного студента повинна зберігатися наступна інформація: прізвище, порядковий номер, отримує стипендію чи ні, середній бал за останню сесію. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Вивести на екран прізвища і номери студентів, які отримують стипендію.

36. Створити файл «*process*», який містить опису для 12 процесорів, які застосовуються в ЕОМ. Параметрами кожного процесора є: назва, (Pentium, Celeron, Duron, Athlon, 486, 486DX2) фірма виробник (AMD, IDC, Texas Instrument), частота, кількість контактів в корпусі мікросхеми. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Для процесорів фірми AMD знайти процесор з найменшою частотою.

37. Створити файл ім'ям «*worker*», що містить опису для 15 працівників підприємства. Параметрами кожного працівника є такі дані: прізвище та ініціали працівника; назва займаної посади; рік надходження на роботу; розмір отримуваної зарплати. Дані повинні вводитися з

клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Вивести на екран прізвище та ініціали працівника підприємства з найвищою зарплатою.

38. Створити файл з ім'ям «*note*», що містить опису для 15 записів з наступними полями: прізвище, ім'я, номер телефону, дата народження (масив з трьох чисел). Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Вивести на екран номер телефону користувача, прізвище якого введена з клавіатури.

39. Створити файл з ім'ям «*znak*», що містить опису 10 записів з наступними даними: прізвище, ім'я; знак зодіаку; дата народження. Дані повинні вводитися з клавіатури. Створити процедуру для можливості перегляду користувачем створеного файлу даних. Вивести на екран дані про тих, хто народився під знаком ваги.

Контрольні питання

1. Структури даних. Призначення, опис, уявлення структур в пам'яті.

2. Покажчики на структури. Доступ до полів структур даних.

3. Вкладені структури.

4. Об'єднання. Призначення, опис, подання до пам'яті. Використання.

5. Перерахування. Призначення, опис, подання до пам'яті. Використання.

6. Файли даних і робота з ними. Поняття файлу даних. Структура файлу. Операції роботи з файлом.

7. Типи файлів. Представлення даних в файлах різних типів.

8. Текстові файли. Оператори запис / читання даних в / з файлу.

9. Функція читання даних із двійкового файлу. Призначення, принцип роботи. Приклади.

10. Функція визначення кінця файлу. Призначення, синтаксис, особливості використання. Приклади.

11. Бінарні файли. Функції читання / запису даних з файлу.

12. Функції відкриття та закриття файлу.

13. Оголошення файлової змінної в програмі.

14. Функція встановлення індикатора позиції файлу.

15. Функція переміщення індикатора позиції файлу в початок файлу.

СПИСОК ЛІТЕРАТУРИ

1. Васильченков О.Г., Тверитникова О.Є., Крилова В.А. Основы алгоритмизации и программирования на С++. Учебное пособие. Харьков: НТУ «ХПІ», 2015. 228 с.

2. Тверитникова О.Є., Крилова В.А., Васильченков О.Г. Методичні вказівки з інформатики «Арифметичні та логічні основи вимірювальної техніки». Харків: НТУ “ХПІ”, 2013. 52 с.

3. Тверитникова О Є., Крылова В. А.,Методические указания по информатике «Основы программирования на С++». Часть 1. Харьков: НТУ “ХПІ”, 2013. 52 с.

4. Тверитникова О.Є., Крылова В.А., Опрышкина М.И Методические указания по информатике «Основы программирования на С++». Часть 2. Харьков: НТУ “ХПІ”, 2014. 68 с.

5. Тверитникова О Є., Крылова В. А. Методические указания по информатике «Основы программирования на С++». Часть 3. Двухмерные массивы. Харьков: НТУ «ХПІ», 2017. 52 с.

6. Програмування мовами С та С++ в середовищі Windows. Навчальний посібник. Вінниця: УНІВЕРСУМ- Вінниця”, 2003. 128 с.

7. Ткачук В.М. Програмування на С++. Лабораторний практикум Івано-Франківськ: Видавництво Прикарпатського національного університету імені Василя Стефаника, 2011. 160 с.

8. Жуковський С.С., Вакалюк Т.А. Програмування мовою С++. Структурне програмування (лабораторний практикум). Навчальний посібник для студентів фізико-математичного факультету. Житомир: Вид-во ЖДУ, 2011. 92 с.

9. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. / Ю.А. Бєлов, Т О. Карнаух, Ю. В. Коваль, А. Б. Ставровський. Київ: Видавничо-поліграфічний центр «Київський університет», 2012. 175 с.

10. Віник В.Ю. Алгоритмічні мови та основи програмування: мова С. навчальний посібник. Житомір: ЖДТУ, 2007. 328 с.

11. Дейтел Х.М. Как программировать на С++. Москва: ЗАО «Издательство БИНОМ», 2000. 1024 с.

12. Страуструп Бьерн. Язык программирования С++. Москва: Бином, 2002. 1098 с.

13. Шилдт Герберт. Справочник программиста по C/C++: пер. с англ. Москва: Издательский дом «Вильямс», 2003. 432 с.

14. Павловская Т.А., Щупак Ю.А. Структурное программирование C/C++. Практикум. Санкт-Петербург: Питер, 2007. 239 с.

15. Прокопенко Ю.В., Татарчук Д.Д., Казміренко В.А. Обчислювальна математика: Навч. посіб. Київ: ІВЦ Видавництво «Політехніка», 2003. 120 с.

16. Вирт Н. Алгоритмы и структуры данных. 2-е изд., испр. Санкт-Петербург: Невский Диалект, 2001. 352 с.

17. Пол Ирэ. Объектно-ориентированное программирование с использованием C++. Киев: НИПФ «ДиаСофт Лтд», 1995. 480 с.

18. Карпов Б., Баранова Т. C++: специальный справочник. Санкт-Петербург: Издательство «Питер», 2000. 480 с.

19. Шилдт Г. Теория и практика C++. Санкт-Петербург: «BHV-Санкт-Петербург», 1996. 416 с.

20. Седжвик Р. Фундаментальные алгоритмы на C++. Алгоритмы на графах. СПб: ООО «ДиаСофтЮП», 2002. 496 с.

ДОДАТКИ

Додаток А

Системи числення

Таблиця А.1 – Перетворення в позиційних системах числення

<table>
<tr><td colspan="4">Приклади перетворення з однієї системи числення в іншу</td></tr>
<tr><td>1</td><td>10→2

$37_{10} = 100101_2$
–37 | 2
36 | –18 | 2
1 | 18 | –9 | 2
0 | 8 | –4 | 2
1 | 4 | –2 | 2
0 | 2 | 1
0</td><td>5</td><td>2→10
5 4 3 2 1 0
$100101_2 = 1\cdot2^5 + 0\cdot2^4 + 0\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 = 37$</td></tr>
<tr><td></td><td></td><td>6</td><td>2→16
$100101_2 = 25_{16}$
2 5</td></tr>
<tr><td>2</td><td>10→8

–37 | 8
32 | 4
5

$37_{10} = 45_8$</td><td>7</td><td>8→2
$41_8 = 100101_2$
100 101</td></tr>
<tr><td></td><td></td><td>8</td><td>16→2
$25_{16} = 00100101_2$
0010 0101</td></tr>
<tr><td>3</td><td>10→16

–37 | 16
32 | 2
5

$37_{10} = 25_{16}$</td><td>9</td><td>8→10
1 0
$45_8 = 4\cdot8^1 + 5\cdot8^0 = 37_{10}$</td></tr>
<tr><td></td><td></td><td>10</td><td>16→10
1 0
$25_{16} = 2\cdot16^1 + 5\cdot16^0 = 37_{10}$</td></tr>
<tr><td>4</td><td>2→8
$100101_2 = 45_8$
4 5</td><td>11</td><td>16→8
$25_{16} = 00100101_2$
$25_{16} = 00100101_2 = 45_8$
0010 0101 0 4 5</td></tr>
<tr><td></td><td></td><td>12</td><td>8→16
$41_8 = 100101_2 = 25_{16}$
100 101 2 5</td></tr>
</table>

Таблиця А.2 – Перетворення позиційних систем числення

Десяткова система числення	Двійкова система числення	Вісімкова система числення	Шістнадцяткова система числення
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	*A*
11	1011	13	*B*
12	1100	14	*C*
13	1101	15	*D*
14	1110	16	*E*
15	1111	17	*F*

Таблиця А.3 – Степені числа 16 у шістнадцятковій системі числення

n (степінь)	0	1	2	3	4	5	6
16^n	1	16	256	4096	65536	1048576	16777216

Таблиця А.4 – Степені числа 8 у вісімковій системі числення

n (степінь)	0	1	2	3	4	5	6
8^n	1	8	64	512	4096	32768	262144

Таблиця А.3 – Додавання в вісімковій системі числення

+	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**10**
0	0	1	2	3	4	5	6	7	10
1	1	2	3	4	5	6	7	10	11
2	2	3	4	5	6	7	10	11	12
3	3	4	5	6	7	10	11	12	13
4	4	5	6	7	10	11	12	13	14
5	5	6	7	10	11	12	13	14	15
6	6	7	10	11	12	13	14	15	16
7	7	10	11	12	13	14	15	16	17
10	10	11	12	13	14	15	16	17	20

Таблиця А.4 – Множення в вісімковій системі числення

·	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**10**
0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	10
2	0	2	4	6	10	12	14	16	20
3	0	3	6	11	14	17	22	25	30
4	0	4	10	14	20	24	30	34	40
5	0	5	12	17	24	31	36	43	50
6	0	6	14	22	30	36	44	52	60
7	0	7	16	25	34	43	52	61	70
10	0	10	20	30	40	50	60	70	100

Таблиця А.5 – Ступені числа 2 у десятковій системі числення

Ступені числа 2^n		
$2^0 = 1$	$2^7 = 128$	$2^{14} = 16\,384$
$2^1 = 2$	$2^8 = 256$	$2^{15} = 32\,768$
$2^2 = 4$	$2^9 = 512$	$2^{16} = 65\,536$
$2^3 = 8$	$2^{10} = 1\,024$	$2^{17} = 131\,072$
$2^4 = 16$	$2^{11} = 2\,048$	$2^{18} = 262\,144$
$2^5 = 32$	$2^{12} = 4\,096$	$2^{19} = 524\,288$
$2^6 = 64$	$2^{13} = 8\,192$	$2^{20} = 1\,048\,576$

Додаток Б
Інтегроване середовище розробки
Microsoft Visual Studio

Інтегроване середовище розробки (Integrated Development Environment) – це програмний продукт, що поєднує текстовий редактор, компілятор, відладчик і довідкову систему. Будь–яка програма, створювана в середовищі Microsoft Visual Studio, завжди оформляється як окремий проект (project). Проект – набір взаємопов'язаних вихідних файлів, компіляція і компонування яких дозволяють створити виконувану програму. Зовнішній вигляд робочої області проекту (project workspace) представлений на рис. Б.1.

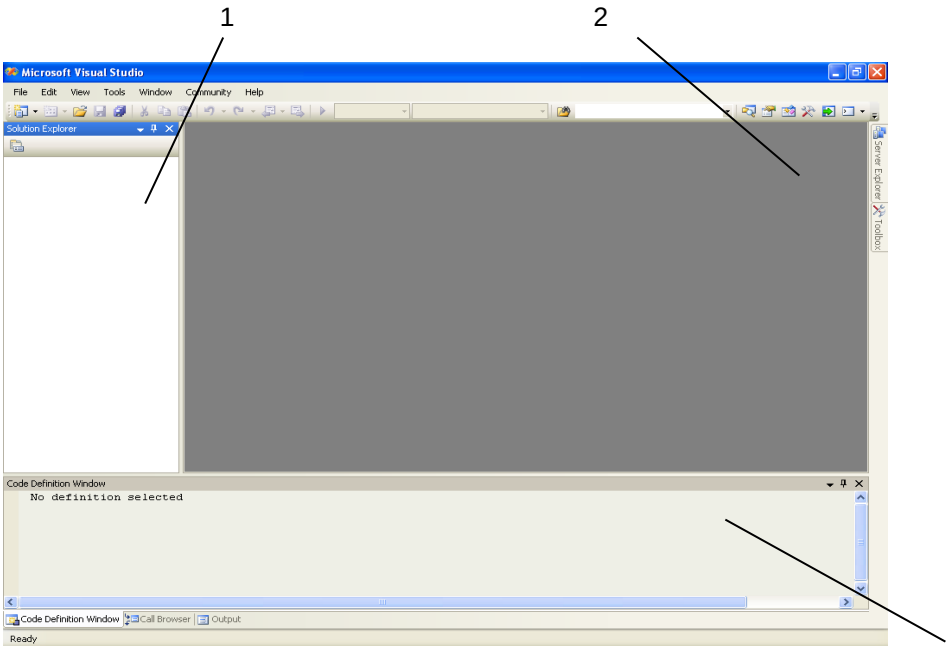


Рисунок Б.1 – Головне вікно програми Visual C ++:
1 – Вікно «Project Workspace»; 2 – Вікно Editor; 3 – Вікно Output

Робоча область проекту може містити будь–яку кількість різних проектів, згрупованих разом для узгодженої розробки: від окремого додатка до бібліотеки функцій або цілого програмного продукту.

Робочий стіл Visual C ++ включає в себе три вікна:

- Вікно Project Workspace (вікно робочої області), призначене для надання допомоги при описі і супроводі великих багатофайлових програм.
- Вікно Editor (вікно редактора) використовується для введення і перевірки вихідного коду.

- Вікно Output (вікно виведення) служить для виведення повідомлень про хід компіляції, збирання і виконання програми. Зокрема, повідомлення про виникаючі помилки з'являються саме в цьому вікні.

Інтегроване середовище Visual C ++ дозволяє будувати проекти різних типів, орієнтовані на різні сфери застосування, також передбачена робота з так званими консольними додатками. При запуску консольного застосування операційна система створює так зване консольне вікно, через яке йде весь введення–виведення програми. Зовні це нагадує роботу в режимі командного процесора операційної системи Windows або інших опнрпціонних систем в режимі командного рядка.

Створення нового проекту

Для створення проекту типу «консольний додаток» необхідно виконати наступні дії:

1. Запустити програму *MS Visual Studio*. «Пуск» – «Програми» – «Microsoft Visual Studio» – Microsoft Visual Studio.

2. Вибрати в рядку меню головного вікна команду File / New / Project.

3. У діалоговому вікні *New Project* необхідно:

- на панелі Project types вибрати тип проекту Visual C ++ / Win 32 (поле 1, рис. Б.2);
- на панелі Templates вибрати тип проекту Win32Project (поле 2, рис. Б.2);
- на текстовому полі Name ввести ім'я проекту – lab1_фамілія (поле 3, рис. Б.2);
- в текстовому полі Location вказати папку для розміщення файлів проекту – папка групи (поле 4, рис. Б.2).
- лівою кнопкою миші клацнути Ok (поле 5, рис. Б.2).

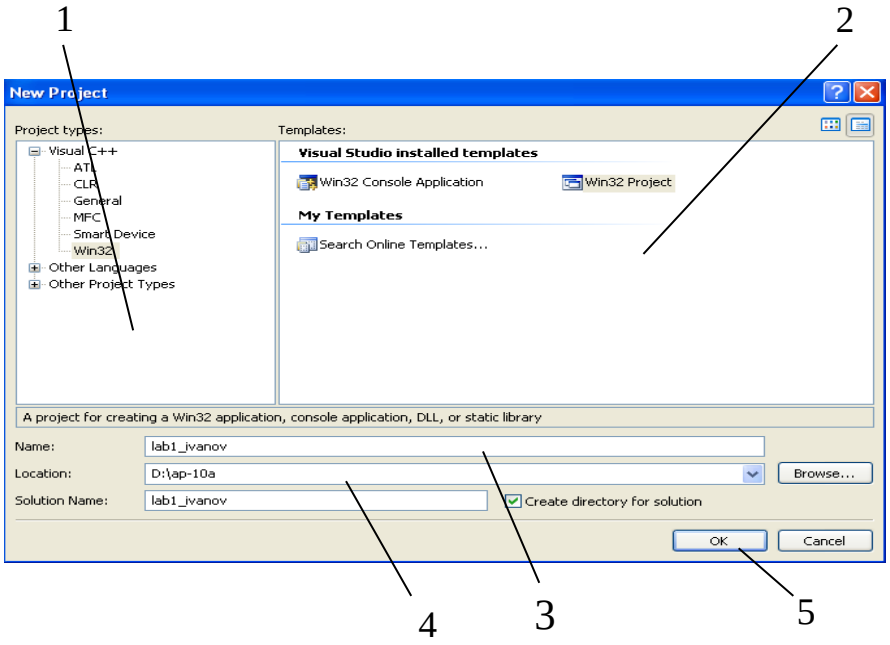


Рисунок Б.2 – Зовнішній вигляд діалогового вікна New Project

4. Після клацання запускається майстер додатків *Application Wizard*. У діалоговому вікні необхідно вибрати закладку Application Settings – додаткові установки (поле 1, рис. Б.3):

- в поле Application type вибрати необхідний підтип додатки. Для консольного застосування в пропонованому списку необхідно вибрати Console Application (поле 2, рис. Б.3);
- натиснути кнопку Finish (поле 3, рис. Б.3).

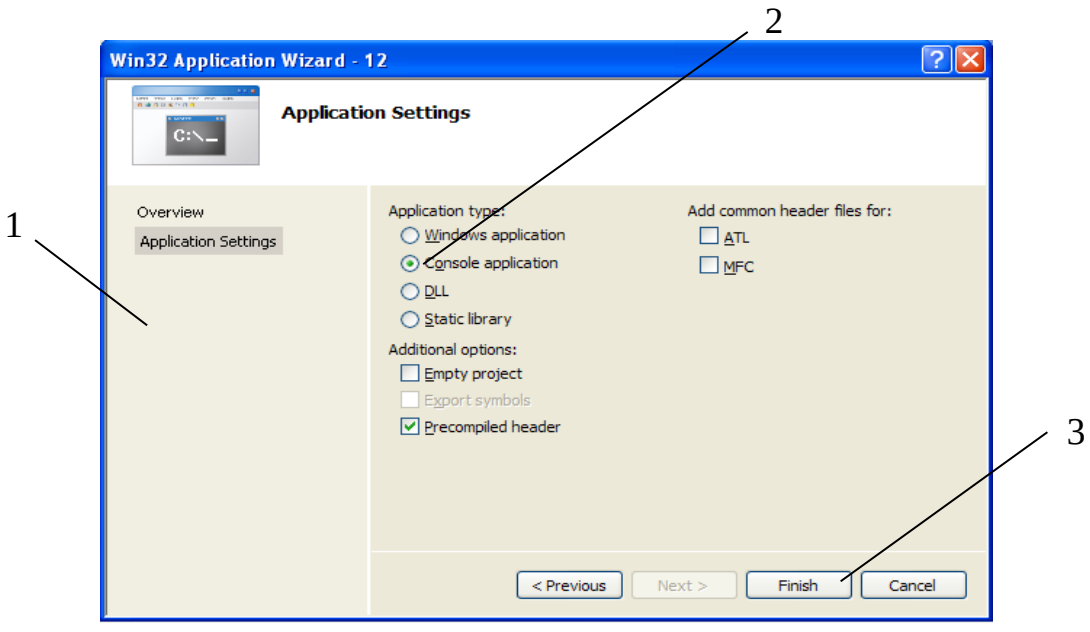


Рисунок Б.3 – Зовнішній вигляд діалогового вікна Application Wizard

Компіляція, компоновка і виконання проекту

Операції компіляція, компоновка і виконання проекту можуть бути виконані через головне меню програми за допомогою кнопок панелі інструментів або за допомогою комбінації клавіш. Для того щоб відкомпілювати створену програму, необхідно в меню вибрати *Build\Build Solution*. *Build* – компоновка проекту. Компілюються всі файли, в яких відбулися зміни з моменту останньої компонування. Після компіляції відбувається збірка (*link*) всіх об'єктних модулів, включаючи бібліотечні в результуючий виконуваний файл. Повідомлення про помилки компоновки виводиться у вікно Output. Про те, що компіляція пройшла успішно, можна переконатися по наступному рядку:

«1> lab1_ivanov – 0 error (s), 0 warning (s)
====== Build: 1 succeeded, 0 failed, 0 up–to–date, 0 skipped ======= «

Також для компіляції можна скористатися клавішею F7 або кнопкою на панелі інструментів (рис. Б.4).

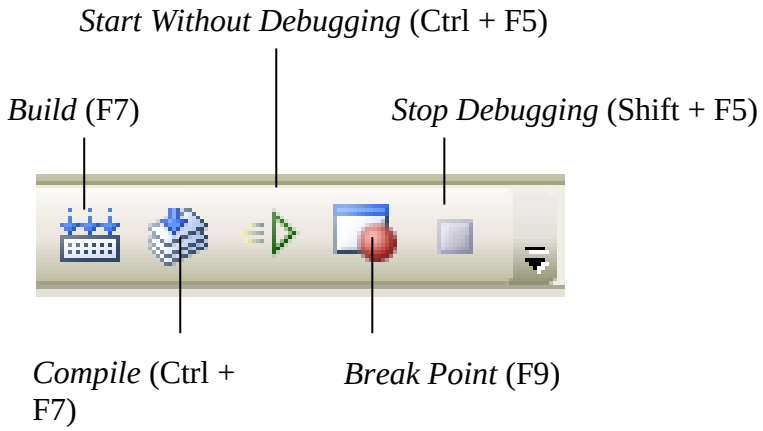


Рисунок Б.4 – Кнопки панелі інструментів для виконання проекту

Для того щоб відкомпілювати вказаний файл, необхідно в головному меню вибрати *Build*/*Compile* або скористатися комбінацією клавіш Ctrl+F7, а також іконкою на панелі інструментів (рис. Б.4).

Якщо компіляція програми успішно завершена без помилок, можна запустити програму на виконання. Для цього в меню необхідно вибрати *Debug/Start Without Debugging* або натиснути кнопку *Start Without Debugging* на панелі інструментів (рис. Б.4), а також можна

використовувати комбінацію клавіш Ctrl+F5. Після запуску на екрані з'явиться консольний додаток з результатами роботи програми.

Робота з отладчиком

В інтегрованому середовищі Visual C ++ виконання програми можна здійснювати як всю цілком, так і через підрядник, тобто програму можна виконувати послідовно, рядок за рядком – такий процес називається покроковим виконанням. Цей режим дозволяє стежити за тим, як змінюються значення різних змінних. Іноді він допомагає зрозуміти, в чому полягає проблема: якщо виявляється, що змінна приймає несподіваного значення, то це може послужити відправною точкою для виявлення помилки. Після виявлення помилки її можна виправити і виконати програму заново в отладочном режимі.

Також в отладочном режимі можлива установка в програмі точки переривання і виконання програми до заданої точки. Коли під час виконання зустрічається точка переривання, програма зупиняється, а на екрані з'являється налагоджувати код. Це дає можливість детально з'ясувати, що відбувається в програмі.

- *Установка точки переривання і виконання програми до точки переривання*

Точка переривання дозволяє зупинити виконання програми перед будь-виконуваної інструкцією (оператором), з тим, щоб продовжувати виконання програми або в покроковому режимі, або в безперервному режимі до наступної точки переривання.

Щоб задати точку переривання перед деякими оператором, необхідно встановити перед ним текстовий курсор і натиснути клавішу F9 або клацнути на панелі інструментів кнопку *Break Point* (рис. Б.4), також можна скористатися головним меню: *Debug*/*Toggle Breakpoint*. Точка переривання позначається у вигляді червоного гуртка на лівому полі вікна редагування. Повторне клацання на зазначеній кнопці знімає точку переривання. У програмі може бути кілька точок переривань.

Після установки точки переривання програма запускається в отладочном режимі за допомогою команди *Debug*/*Start Debugging* або натисканням клавіші F5. В результаті код програми виконується до рядка, на якій встановлена точка переривання. Потім програма зупиняється і відображає у вікні Editor ту частину коду, де знаходиться точка

переривання, причому жовта стрілка на лівому полі вказує на рядок, яка буде виконуватися на наступному кроці налагодження. Для того щоб завершити роботу з отладчиком, необхідно натиснути комбінацію клавіш Shift + F5.

- *Покрокове виконання програми*

Для покрокового виконання програми без заходу у функції необхідно вибрати в головному меню *Debug/Step Over* або клавішу F10. Натискаючи клавішу F10, можна виконувати один оператор за іншим. Якщо необхідно виконати покрокове виконання програми з кодом викликається функції, то треба натиснути клавішу F11 або вибрати в меню *Debug/Step Into*.

Після запуску засобів налагодження змінюється зовнішній вигляд і склад вікон інтегрованого середовища розробки. Відповідні кнопки на панелі інструментів дозволяють розставляти крапки переривання і керувати покроковим виконанням програм. Замість точок переривання можна скористатися функцією «Виконувати до курсора» (Ctrl + F10).

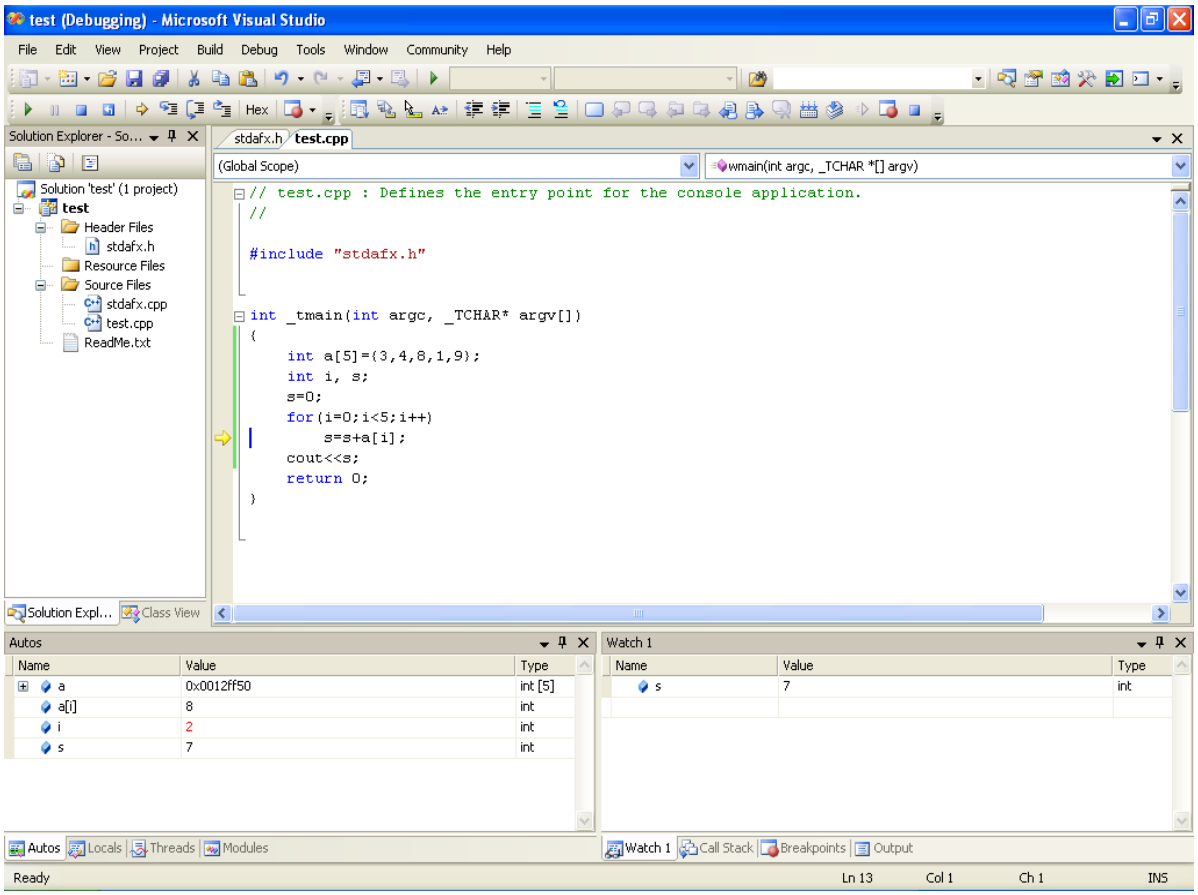


Рисунок Б.5 – Зовнішній вигляд вікон в отладочном режимі

Під час налагодження програми, для того щоб дізнатися значення змінної, необхідно затримати над нею вказівник, і поруч з ім'ям змінної на екрані з'явиться підказка зі значенням цієї змінної. Крім екранної підказки, змінна зі своїм значенням відображається у вікні *Auto*, розташованому в лівому нижньому кутку екрану. У цьому вікні наведені значення останніх змінних, з якими працював *Visual C ++*. Крім цього, у вікні *Watch*, яке знаходиться в правому нижньому кутку, можна задати ім'я змінної, за значеннями якої ви хочете поспостерігати (рис. Б.5).

Відкриття створеного раніше проекту

1 спосіб

1. Запустити на виконання *Visual C ++*
2. Вибрати в меню *File, пункт Open Workspace*
3. У діалоговому вікні необхідно знайти папку з вашим проектом, а в ній файл *імя_проекта*.dsw
4. Відкрити цей файл.

2 спосіб

1. Запустити на виконання *Visual C ++*
2. Вибрати в меню *File* і навести курсор миші на пункт *Recent Workspace*.
3. Якщо в меню зі списком останніх файлів, з якими йшла робота, ви знайдете цікавить вас файл імя_проекта.dsw, то необхідно клацнути по ньому мишею.

3спосіб

1. Чи не викликаючи *Visual C ++*, необхідно знайти папку з вашим проектом, а в ній файл *імя_проекта.dsw*.

2. Клацнути мишею на файлі *імя_проекта.dsw*

Навчальна видання

ТВЕРИТНИКОВА Олена Євгенівна
КРИЛОВА Вікторія Анатоліївна
ВАСИЛЬЧЕНКОВ Олег Георгійович

БАЗОВІ АЛГОРІТМИ ТА ОСНОВИ ПРОГРАМУВАННЯ. ТЕОРІЯ І ПРАКТИКА

Навчальний посібник
для студентів спеціальностей «Автоматизація та комп'ютерно-інтегровані технології», «Метрологія та вимірювальна техніка»
усіх форм навчання вищих навчальних закладів

Відповідний за випуск Кондрашов С.І.

Роботу до друку рекомендував О.В. Дудник

Редактор Самініні О.С.

План 2020 р., Поз. 23

Підпісано до друку. .20. формат 60×84 1/16. Папір друк. № 2.
Друк – різографія. гарнітураTimes New Roman. Розум. друк. арк.3,2.
Обл. – вид. арк. 2,7.Наклад 100 прим. Зам. № .Ціна договірна.

Видавничий центр НТУ «ХПІ». 61002, Харків, вул .. Фрунзе, 21.
Свідоцтво про реєстрацію ДК № 3657 від 24.12.2009 р.

Друкарня НТУ «ХПІ», 61002, Харків, вул. Фрунзе, 21.