МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний технічний університет
«Харківський політехнічний інститут»

МЕТОДИЧНІ ВКАЗІВКИ
до виконання самостійних робіт
з курсів: «Інформаційні технології і програмування»
«Інформаційні процеси в телекомунікаційних системах»

ОСНОВИ ПРОГРАМУВАННЯ НА МОВІ С++

для студентів спеціальностей:
151 Автоматизація та комп'ютерно-інтегровані технології
172 Телекомунікації та радіотехніка

затверджено
редакційно-видавничою
радою університету,
протокол №3 від 26.10.2022 р.

Харків
НТУ «ХПІ»
2022

Методичні вказівки для виконання самостійних робіт «Основи програмування на мові С++» для студентів спеціальностей 151 Автоматизація та комп'ютерно-інтегровані технології, 172 Телекомунікації та радіотехніка / Уклад. А.О. Зуєв, Д.А. Гапон, М.А. Денисенко - Х.: НТУ «ХПІ», 2022. - 45 с.

Укладачі: А. О. Зуєв
Д.А. Гапон
М.А. Денисенко

Рецензент О.Є. Тверитникова

Кафедра інформаційно-вимірювальних технологій і систем

ВСТУП

Розвиток сучасних технологій неможливий без використання комп'ютерної техніки та програмного забезпечення. Підготовка фахівців в області систем управління, автоматики та телекомунікаційних систем вимагає широких знань і навичок володіння обчислювальною технікою, а так само знання основ алгоритмізації та мов програмування високого рівня.

Методичні вказівки призначені для вивчення основ мови програмування С++ як на практичних і лабораторних роботах, так і для самостійного вивчення. У методичних вказівках на прикладах розглядаються основні принципи програмування та методи написання програм на мові С++, а саме: вбудовані і користувацькі типи даних, структури і масиви та їх ініціалізація і використання, оператори мови С++, логічні і математичні функції, цикли і умовні оператори, функції, передача аргументів і повернення результату роботи функції, шаблони і обробка виключних ситуацій, а також основи роботи з потоками введення-виведення.

Наведено індивідуальні завдання для виконання практичних і лабораторних робіт, а також завдання для самостійного вивчення мови програмування. Розглянуті приклади та завдання допоможуть в ефективному освоєнні програмування на мові С++.

1 ЗАГАЛЬНІ ВІДОМОСТІ

Кожна програма на мові С++ складається з двох типів файлів:

1) заголовних (з розширенням .h або .hpp), в яких зазвичай містяться описи констант, глобальних змінних, класів, структур і функцій, підключення бібліотек.

2) файлів реалізації (з розширенням .cpp), в яких міститься компільований код.

Заголовні файли підключаються директивою #include до файлів реалізації. Після директиви вказують в кутових дужках назву системної бібліотеки, або в лапках назву файлу з проекту що розробляється.

Будь-яка програма на мові С++ містить як мінімум одну функцію з назвою main (_tmain), з якої починається її виконання.

Кожен рядок програми закінчується символом крапки з комою. Код програми і опис класів та структур обертається в фігурні дужки {}. Також фігурні дужки використовуються для позначення тіл операторів, або довільних блоків коду.

Багаторядкові коментарі обмежуються символами /* на початку і */ в кінці. Однорядковий коментар починається з символів // і закінчується в кінці рядка.

1.1 Об’ява об'єктів

Кожна об’ява на мові С++ складається з двох частин: назви типу і назви об'єкта, розділених пропуском. Тип визначає те, які операції доступні для об'єкта, а також формат і розмір займаної об'єктом пам'яті.

Об’ява — це інструкція, яка вводить нове ім'я в програмі, і вказує його тип. Будь-який об'єкт повинно бути об’явлено перед його використанням. Час існування кожної об’яви за замовчуванням дорівнює часу існування того блоку в якому вона зроблена.

У мові С++ блоки обмежуються фігурними дужками {}.

Для задавання імені можна використовувати латинські літери у верхньому та нижньому регістрі, символ підкреслення _, а також цифри. При цьому ім'я має починатися з підкреслення або латинської літери, а великі та маленькі літери вважаються різними. Об’яви одного типу можуть розділятися символом коми, що гарантує їх послідовну ініціалізацію.

Наприклад: _abcd, Abcd, ABC_def, a124

Тип — визначає варіанти використання об’єкту або виразу. Типи даних діляться на вбудовані — вони є частиною мови С++, і користувацькі — вони об’являються в бібліотеках або в заголовних файлах програми.

Приклади об’яв з використанням вбудованих типів: int abcd; char c;

При об’явленні треба пам'ятати: все, що використовується (об’єкти, типи, та інше), має бути об’явлено вище за текстом програми (раніше).

1) Час існування об'єктів. Кожен об'єкт або тип існує в програмі в межах того блоку, де він об'явлений. Зазвичай, об'єкт знищується, якщо виконання програми виходить за межі блоку (за фігурну дужку }), де було зроблено його об’яву.

Об'єкти, створені поза блоком або з модифікатором **static**, існують протягом усього часу роботи програми. При цьому об'єкти з модифікатором static, об'явлені всередині блоку, створюються при першому виконанні рядка з їх об’явою.

2) Простори імен. Кожна об’ява або тип може бути наведено в певному просторі імен. Для цього потрібно визначити простір ключовим словом **namespace**, після якого вказати назву простору, і в фігурних дужках {} перерахувати об’яви і типи, після чого закінчити об’яву простору крапкою з комою. Для того, щоб використовувати тип з простору імен, необхідно використовувати оператор :: перед яким необхідно вказати ім'я простору, а потім — ім'я типу або об'єкту.

Приклад:

```
namespace work
{
        int x;
};
work::x = 10;
```

Також можна вказати простори імен, які використовуються за замовчуванням, для цього використовується конструкція **using namespace**, після якої вказується ім'я простору (в кінці крапка з комою). В цьому випадку компілятор сам спробує знайти, звідки взяти ту чи іншу конструкцію і явну вказівку з використанням оператора :: наводити непотрібно. Але вона може знадобитися, якщо в двох просторах є об’яви з однаковими іменами. *Простори імен доцільно застосовувати для логічного поділу типів і об’яв за їх призначенням.*

1.2 Вбудовані типи даних

Вбудовані типи даних в мові C++ можна умовно розділити на 3 групи: цілочисельні, дійсні і логічні.

1) Цілочисельні типи даних можуть зберігати цілі числа зі знаком і без знака. До цих типів відносяться char, short, int, long. Всі типи даних зберігають числа зі знаком, при цьому, позитивні числа будуть зберігатися в прямому коді, а негативні в додатковому. Старший розряд призначений для зберігання знака, 0 — плюс, 1 — мінус. Діапазон збережених чисел можна зрушити в позитивну область, додавши перед назвою типу приставку unsigned, в цьому випадку, в змінної такого типу можна зберігати тільки позитивні числа в прямому коді. Об’єкти цілочисельних типів зберігають точне значення числа.

Розміри цілочисельних типів даних залежать від комбінації операційної системи, процесора і компілятора. Стандарт мови гарантує тільки мінімальні розміри для кожного типу:

char – 8 біт, short – 16 біт, int – 16 біт, long – 32 біт.

Також стандартом гарантується, що розмір long буде більше або дорівнювати int, а розмір int, в свою чергу, більше або дорівнюватиме short, розмір short більше або дорівнює розміру char. При цьому розмір жодного типу даних не може бути меншим за розмір типу char (8 біт).

Для визначення розміру типів і об'єктів використовується функція **sizeof(<тип>| <об'єкт>)**. Ця функція повертає розмір переданого їй типу або об'єкта в розмірах типу char. Таким чином, sizeof(char) завжди дорівнює 1.
Типи даних меншого розміру (по розрядності) автоматично перетворюються до типів більшого розміру, при присвоюванні.

Компілятори мови C++ зазвичай вводять свої власні типи даних, наприклад, _int32, _int64 або long long, для зберігання чисел з фіксованою розрядністю.

2) Дійсні типи даних дозволяють зберігати числа з плаваючою комою одинарної або подвійної точності. До цих типів відносяться float, double, long double. Перші два типи стандартизовані, їх розміри фіксовані, для float — 32 біта, для double — 64 біта. Визначення типу long double еквівалентно типу double (але це різні типи с точки зору мови C++). Дійсні числа зберігаються в нормалізованому вигляді, зі зрушенням до старшого одиничного розряду. Число складається з трьох частин — знака, мантиси (цифр числа) і експоненти. При цьому мантиса завжди зберігається в прямому коді. Об’єкти таких типів зберігають приблизне значення числа, а не точне, як целочисельні, зберігаються тільки кілька старших розрядів. Для float це приблизно 7 десяткових цифр, а для double — 14.

3) Логічний тип даних bool може зберігати значення істина або хибність, які визначаються константами true і false. Тип bool автоматично наводиться до будь-якого цілого типу, таким чином, що true стає 1, а false – 0. Припустиме також зворотне перетворення, в цьому випадку 0 буде перетворений в false, а всі інші числа в true.

1.3 Операції з типами

Для цілих і дійсних типів припустимі наступні операції +, -, *, /. При цьому операція ділення для цілих типів повертає частку а для отримання остачі використовується операція %. Якщо в обчисленні виразу беруть участь і цілі і дійсні числа, то результат приводиться до більшого дійсного типу.

Всі оператори можуть скорочуватися, якщо обчислення йде з «накопиченням».

Наприклад: a = a + 5 *може бути записано як* a += 5.

Для цілих типів визначені операції інкремент і декремент ++, --, а також операції з битами: логічні &, |, ~, ^ і зсув >>, <<. Для завдання пріоритетів операцій використовуються ().

1) Літеральні константи використовуються для завдання початкових значень. Якщо константа містить тільки цифри і не починається з 0, то вона розглядається як константа типу int в десятковій системі числення. Якщо константа починається з 0, то вона записана в вісімковій системі, якщо починається з 0x, то в шістнадцяткової.

Наприклад: 10, 010, 0x10.

Якщо після цифр записана буква L, то це трактується як константа типу long, якщо записано U, то це константа беззнакового типу.

Наприклад: 10L, 1010UL.

Якщо крім цифр в описі константи зустрічається крапка або літера e, це константа дійсного типу. Якщо в кінці записана F, це константа типу float.

Наприклад: 10.0, 10.f, 5e-2

Якщо константа записана в одинарних лапках, вона розглядається як код символу, який знаходиться в лапках, і має тип char.

2) Об’ява змінних і констант складається з імені типу, імені змінної і значення для ініціалізації (яке може бути іншою змінною, константою або літеральною константою), розділених символом =. При цьому для змінних значення для ініціалізації необов'язкове, а для констант обов'язкове. Об’ява константи починається з ключового слова **const.**

Наприклад:
int x; // змінна цілого типу з назвою x.
const float pi = 3.14f; // константа дійсного типу з назвою pi.

3) Об’ява масивів фіксованого розміру, аналогічна об'яві змінної, також вказується тип елементів масиву і назва масиву, розділені пропуском, але після назви в квадратних дужках вказується кількість елементів в масиві - ціле позитивне число (або константа цілого типу) більше 0. Розмір масиву визначається на етапі компіляції і не може змінюватися під час роботи програми.

У масиві всі елементи однотипні (мають однаковий розмір і властивості) і розташовуються в пам'яті один за одним, це дозволяє в будь-який момент часу отримати доступ до будь-якого елементу масиву по його номеру (індексу) підсумувавши початкову адресу масиву і номер елемента, помножений на його розмір. Ці дії компілятор виконує автоматично. Доступ до елементів масиву здійснюється за допомогою оператора [] (квадратні дужки) в якому вказується індекс елемента. Нумерація індексів починається з 0, максимально допустиме значення індексу менше на одиницю ніж розмір масиву. Автоматична перевірка на вихід за межі масиву не ведеться.

Приклад: int A[10]; // масив для 10 цілих чисел типу int з назвою A

Масив може містити як змінні, так і константи, в разі констант перед об’явам пишеться ключове слово const. Для масиву констант обов'язковий список ініціалізації. Список ініціалізації для масиву пишеться в фігурних дужках, після оператора присвоювання.

Приклад: int A[5] = { 10, 5, 3, 4, 5 };

У списку ініціалізації елементів повинно бути не більше ніж зазначений розмір масиву. Якщо елементів менше, то відсутні будуть заповнені значенням за замовчуванням, для чисел значення за замовчуванням це 0.

Приклад: int A[5] = {1, 2, 3};

У масиві будуть наступні значення: 1, 2, 3, 0, 0.

У списку ініціалізації повинен бути як мінімум один елемент. Якщо список ініціалізації присутній, то вказувати розмір масиву необов'язково, він буде вирахувано автоматично, за кількістю елементів для ініціалізації, в цьому випадку квадратні дужки можна залишити порожніми.

Приклад: int A[] = {1, 2, 3};

Визначити кількість елементів в масиві можна за допомогою наступного виразу sizeof(A) / sizeof(A[0]) - розділити розмір масиву на розмір одного елемента масиву. Масив буде мати розмірність 3.

Для ініціалізації константного рядка в «стилі мови С» — масиву символів, використовується спеціальний запис символів в подвійних лапках:

const char str[] = "12345ABCD";

1.4 Приведення типів

Мова С++ є мовою зі строгою типізацією, всі типи всіх об'єктів повинні бути відомі на етапі компіляції і не можуть помінятися в процесі виконання програми. Допустимо тільки автоматичне перетворення об'єктів при ініціалізації, копіюванні, передачі і поверненні. При цьому автоматичне перетворення діє тільки в тому випадку якщо з меншого типу даних необхідно отримати більший. Але в деяких випадках виникає необхідність в ручному перетворенні (приведенні) типів.

Для ручного приведення типів від одного до іншого в С++ допускається використовувати як механізм з мови С, коли в круглих дужках вказується тип до якого потрібно привести, або оператори **static_cast** і **const_cast,** які дозволяють провести обмежену перетворення або прибрати константність з об'єкта. Для двох останніх операторів в кутових дужках вказується тип до якого наводиться, а в круглих той об'єкт який потрібно привести.

Приклад приведення типів в стилі мови С і С++:
int i = 12345;
char c = (char) i; або char c = char(i);
char c = static_cast<char>(i);

По можливості слід утримуватися від ручного приведення типів (особливо в стилі мови С), використовуючи тільки автоматичне приведення. Для цього слід ретельно продумувати типи змінних, констант і аргументів функцій та результати що повертаються.

Індивідуальні завдання

1. Створіть шаблон програми, що складається з двох файлів, заголовку і файлу реалізації.

2. Визначте змінну і константу типів int і char, з початковою ініціалізацією і без.

3. Визначте масив чисел типу float, з 5 елементів, зі списком початкової ініціалізації і без.

4. Визначте масив чисел типу int, з 7 елементів, зі списком початкової ініціалізації. Запишіть в 4-й, елемент масиву значення 0.

5. Визначте новий простір імен, в цьому просторі змінну типу double. У програмі (поза цім простором) змініть значення цієї змінної.

6. Визначте дві змінних типу int і float, надайте значення однієї змінної інший, використовуючи приведення типів в стилі мови C++.

7. Визначте змінну і константу типів double і float, з початковою ініціалізацією одним і тим же значенням і без нього.

8. Визначте масив чисел типу short, зі списком початкової ініціалізації з 5 значень. Запишіть в 1-й і 3-й, елементи масиву значення 5 і 7.

9. Визначте новий простір імен, і в цьому просторі константу типу int. У програмі запишіть значення цієї константи в змінну.

10. Визначте масив чисел типу long, зі списком початкової ініціалізації з декількох елементів. Обчисліть розмір масиву в елементах і помістіть його в змінну типу int.

11. Визначте новий простір імен, в цьому просторі змінну типу char. У програмі Ви можете активувати цю змінної рівним коду латинського символу 'Z'.

12. Визначте константний рядок в стилі мови C, надайте цій константі як початкове значення - ваше ім'я.

13. Визначте масив беззнакових чисел будь-якого типу, з 5 елементів, зі списком початкової ініціалізації.

14. Визначте новий блок коду, визначте в ньому змінну типу int таким чином, щоб вона продовжила існувати після виходу з блоку.

Контрольні питання

1. Що таке об’ява? Наведіть приклади об’яв допустимих в мові С++.
2. Що таке тип і що він визначає?
3. Які вбудовані типи даних є в мові С++?
4. Які вимоги і обмеження пред'являються до цілочисельних типів даних в мові С++?
5. Чим відрізняються цілочисельні типи даних від дійсних?
6. Для чого призначений логічний тип даних?
7. Визначення розміру типів даних в мові С++?
8. Які основні операції що можуть бути застосовані до об'єктів цілочисельних типів даних?
9. Що таке літеральні константи і які правила їх об’яви?
10. Чим відрізняються об’яви змінних і констант?
11. Що таке масив?
12. Перерахуйте особливості ініціалізації елементів масиву.
13. Як визначити кількість елементів в масиві?
14. Які особливості доступу до елементів масиву?
15. Для чого застосовується модифікатор static?
16. Чим визначається час існування об'єктів в програмі?
17. Для чого застосовується оператор кома?
18. Що таке простір імен і для чого він застосовується?
19. Які способи приведення типів є в мові С++?
20. Що таке приведення типів, для чого воно потрібне?

2 ОПЕРАТОРИ І ОПЕРАЦІЇ

2.1 Оператор присвоювання

Оператор присвоювання позначається одним знаком рівності (=), розділяє вираз на дві частини. Він обчислює праву частину виразу, і результат обчислення зберігає в лівій частині. У правій частині можуть бути будь-які вирази, виклики функцій і т. ін. У лівій частині зазвичай знаходиться змінна, в якій зберігається результат операції присвоєння.

Арифметичні операції, результат яких присвоюється якийсь змінної з накопиченням, можуть бути скорочені, шляхом перенесення знаку операції вліво щодо оператора присвоювання і видалення дубліката назви змінної в якій відбувається накопичення.

Наприклад, такі записи еквівалентні:
value = value + 5;
value += 5;

Також, знак оператора присвоювання (=) використовується для ініціалізації змінних, масивів і констант початковим значенням.

2.2 Операції порівняння і логічні операції

Ці операції призначені для перевірки двох об'єктів на нерівність і для об'єднання декількох перевірок в одну. Результатом таких операцій є логічна величина, яка може приймати два значення - істина або хибність, в термінах мови C++ - об’єкт типу bool.

Операції порівняння: на рівність і нерівність ==, !=, Порівняння на більше або менше <, >, <=, >=. Знаки операцій з двох символів записуються без пробілів між ними.

Логічні операції: логічне множення &&, логічне додавання || і логічне заперечення !. Перші дві операції бінарні, і дозволяють об'єднувати дві логічні величини, а операція заперечення унарна, і змінює значення логічного виразу на протилежне.

Приклад: a < b && !(c>= b)

2.3 Умовні оператори

Умовні оператори дозволяють організувати в програмі розгалуження на декілька варіантів виконання, який з варіантів дійсно буде виконаний — визначається умовою або значенням змінної. У мові C++ існує три види умовних операторів: умовний оператор if, тернарний оператор ? і оператор вибору switch.

1) Умовний оператор if застосовується в найпростішому випадку, коли необхідно організувати розгалуження на два варіанти. Цей оператор є еквівалентом двохпозиційного перемикача.

Позначається як if (A) {B;} else {C;}. Блок A — умова, одна або кілька операцій порівняння, які можуть бути об'єднані логічними операціями. Блок B виконується в тому випадку якщо значення A істинно, в іншому випадку виконується блок C, який відділений оператором else. При цьому else і блок C необов'язкові, і якщо їх немає, то в разі якщо

умова A помилково - програма продовжить виконання з наступного за оператором if рядки коду.

2) Тернарний оператор ? — скорочений запис умовного оператора if, зазвичай застосовується для обчислення виразів, в тому випадку якщо алгоритм обчислення залежить від умови, а результат обчислення в обох випадках поміщається в одне й теж місце (змінну).

Позначається як y = A ? B : C; — де позначення блоків таке ж як і в попередньому випадку, y — та змінна, в яку буде збережено результат обчислення, замість if пишеться ?, замість else — двокрапка, в цьому випадку друга частина обов'язкова.

Приклад, повна і скорочена запис:

```
int x; if( a < b ) { x = 25; } else { x = 30; }
int x = a < b ? 25 : 30;
```

3) Оператор вибору switch застосовується в тому випадку, якщо необхідно організувати розгалуження більш ніж на два варіанти. Він дозволяє в залежності від значення змінної вибрати і виконати один або кілька блоків коду. Такий оператор фактично є еквівалентом багатопозиційного перемикача. Керуючим елементом оператора є змінна цілого типу A, для якої задається безліч значень (констант) які вона може приймати — A1, A2, A3, ..., An. Кожному значенню відповідають блоки коду B1, B2, B3, ..., Bn, відповідно. Блок Bi виконується в тому випадку якщо змінна A прийняла відповідне значення Ai.

Позначення:

```
switch( A )
{
        case A1: { B1; break; }
        case A2: { B2; break; }
        case A3: { B3; break; }
                . . .
        case An: { Bn; break; }
        default: { C; break; }
}
```

Блок C виконується в тому випадку, якщо не знайдено жодного збігу в множині A1, A2, A3, ..., An з актуальних значенням змінної A. Частина, позначена міткою default, є необов'язковою, так само, як і else в умовному операторі.

4) Оператор break перериває виконання оператора switch і виконання програми переходить до наступного (після оператору switch) рядка. Якщо оператор break не наведено в якомусь з блоків, то виконання оператора switch продовжиться з наступного за списком варіанту і так далі, поки не зустрінеться оператор break або не закінчяться всі варіанти.

Допускається на один блок коду ставити кілька міток з варіантами.

Наприклад: case A1: case A2: case A3: { B2; break; }

В цьому випадку, якщо значення змінної A рівні A1, A2 або A3, виконається блок B2. Якщо змінна A має тип bool, то оператор вибору є еквівалентом умовного оператора if.

2.4 Оператори циклу

Цикли дозволяють виконати один і той же блок коду (тіло циклу) кілька разів поспіль. У мові C++ існує три види циклів: з передумовою **while і for** і з післяумовою **do ... while**. Одноразове виконання тіла циклу - *ітерація*.

1) Цикл з передумовою while має вигляд while (A) {D;}. Блок A — умова, поки вона дорівнює істині, виконується тіло циклу — D. Діаграма роботи циклу: A^DA^(DA^). Якщо спочатку умова хибна, то тіло циклу не виконається жодного разу. Знак ^ позначає що після попереднього блоку можливий вихід з циклу. У дужках — частина, що повторюється у разі істинності умови A.

2) Цикл з післяумовою do ... while має вигляд do {D;} while (A);. Блоки аналогічні попередньому циклу, але тіло циклу виконується до перевірки умови. Діаграма роботи циклу: DA^(DA^). У цього циклу тіло виконається як мінімум один раз.

3) Цикл з передумовою for має вигляд for (B; A; C) {D;}. Універсальний цикл, з найширшими можливостями, може замінити будь-який цикл. Блок A — умова циклу, може бути відсутнім. Блок B — блок початкової ініціалізації, виконується один раз перед початком циклу, може бути відсутнім. Блок C — виконується після виконання тіла циклу D, перед умовою, також може бути відсутнім.

Блоки поділяються крапкою з комою (які повинні бути завжди), самі блоки можуть бути наявні в будь-яких комбінаціях. Діаграма роботи циклу: BA^DCA^DC. Якщо спочатку умова хибна, то як і в циклі while, тіло циклу не виконається жодного разу.

Якщо в for відсутні блоки B і C, він є повним еквівалентом циклу while.
Цикл for доцільно використовувати в тому випадку якщо заздалегідь відомо скільки разів потрібно повторити тіло циклу, наприклад для обробки елементів масивів.

Типовий приклад використання виглядає таким чином:

```
for( int i = 0; i < N; ++i ) { D; }
```

Всередині циклу, в блоці B задана змінна циклу i, яка показує номер ітерації, N — задає кількість повторів тіла циклу. У блоці B і C може застосовуватися оператор кома для послідовної ініціалізації або зміни змінних циклу. Змінні об'явлені в блоці B є локальними змінними циклу і час їх існування дорівнює часу роботи циклу.

4) Оператори управління циклом break, continue — дозволяють управляти циклом з тіла циклу (блоку D). Оператор break перериває виконання циклу і програма переходить до виконання наступного рядка за мажами циклу. Оператор continue переходить до наступної ітерації циклу, для циклів while і do ... while до перевірки умови (блоку A), а для for до блоку C, якщо він є, або до блоку A, якщо блоку C немає.

Індивідуальні завдання

1. Визначте змінну будь якого целочисельного типу, збільште її значення на одиницю трьома різними способами.

2. Визначте вираз $x = \overline{a \barwedge b \wedge \bar{a} \vee c}$ за допомогою логічних операторів.

3. Визначте вираз $x = \overline{\bar{a} \wedge b \vee c < d + 5}$ за допомогою умовних та логічних операторів.

4. Визначте вираз $y = \begin{cases} x^2, якщо\ x < -5 \\ \sin(x), якщо\ -5 \le x < 5 \\ \sqrt[3]{x+2}+3, якщо\ 5 \le x \end{cases}$ із застосуванням умовного оператору*.

5. Визначте вираз $y = \begin{cases} x^3+5, якщо\ a < 0 \\ \sin^a(x), якщо\ 0 \le a < 10 \\ \sqrt[0,5]{2^x + a}, якщо\ 10 \le a \end{cases}$ за допомогою тернарного оператору*.

6. Визначте вираз $y = \begin{cases} x^3, якщо\ a = 0\ та\ a = 2 \\ x^2, якщо\ a = 5 \\ x, якщо\ a = -3 \end{cases}$ застосував оператор вибору*.

7. Перетворіть наступний фрагмент програми:

```
int x = 0;
if( a == 3 ) { x = 2; }
else
{
        if( a == 0 ) { x = 1; }
        else
                { x = 3; }
}
```

змінивши умовний оператор на оператор вибору.

8. Перепишіть наведений нижче фрагмент програми з використанням оператора continue:

```
for( int i = 0; i < N; ++i )
{
        if( A[i] < 0 )
        A[i] *= A[i];
}
```

таким чином, щоб фрагмент програми працював так само.

9. Перепишіть наведений нижче фрагмент програми без використання оператора break:

```
for( int i = 0; i < N; ++i )
{
        if( A[i] > 0 )
                break;
}
```

таким чином, щоб фрагмент програми працював так само.

10. Напишіть програму, яка б заповнювала масив з N цілих чисел зростаючої послідовністю що складається з непарних чисел 1, 3, 5, 7, ...

11. Напишіть програму, яка підраховує кількість негативних чисел в масиві з N елементів, з використанням циклів for і while.

12. Напишіть програму, яка замінює негативні числа в масиві з N елементів значенням 0, з використанням циклів while і do .. while.

13. Напишіть програму, яка підраховує суму невід'ємних елементів масиву з N елементів.

14. Напишіть програму, яка підраховує добуток елементів масиву дійсних чисел з N елементів.

** Для роботи з математичними і тригонометричними функціями необхідно підключити бібліотеку math.h. Тригонометричні функції для типу double визначені без префікса: sin, cos, tan, pow(x, y) - зведення x в ступінь y, sqrt - квадратний корінь. Для типу float необхідно використовувати префікс f, наприклад sinf, sqrtf і т.ін.*

Контрольні питання

1. Як працює оператор присвоєння і для чого він потрібен в програмі?
2. Як працює скорочення запису арифметичних операцій?
3. Для чого застосовуються операції порівняння і логічні операції?
4. Для чого застосовуються умовні оператори?
5. Як замінити умовний оператор if за допомогою тернарного оператора?
6. Як замінити тернарний оператор умовним оператором if?
7. Для чого застосовується оператор вибору?
8. Які варіанти виконання блоків допустимі для оператора вибору?
9. Для чого застосовується оператор break всередині оператора вибору?
10. Для чого потрібні оператори циклу?
11. Які оператори циклу застосовуються в мові C++? У чому між ними різниця?
12. У чому різниця між циклом з після- та передумовою?
13. Як замінити цикл з післяумовою на цикл з передумовою?
14. Які особливості запису і роботи циклу for?
15. Для чого застосовуються оператори break і continue всередині циклів?
16. Для чого застосовується оператор кома в циклі for?
17. Які частини циклу for потрібно прибрати, щоб він став функціонально еквівалентний циклу while?
18. У чому полягає різниця між циклом while і do..while?
19. Який цикл більш універсальний for або while і чому?

3 ФУНКЦІЇ, ПЕРЕДАЧА АРГУМЕНТІВ І ПОВЕРНЕННЯ РЕЗУЛЬТАТІВ

Функції визначають межі коду що виконується. Весь код, який повинен бути виконаний, має бути зосереджено всередині функцій. Об’ява функції аналогічна об’яві змінної (тип та назва), за яким слідує список аргументів який поміщено в круглі дужки. Кожен аргумент являє собою пару: тип та назва, аргументи розділяються комою.

Приклад об’яви: int func1(int p1, int p2);

Якщо аргументів немає, то записуються тільки круглі дужки. Визначення закінчується крапкою з комою і зазвичай поміщається до файлу заголовку.

Приклад об’яви функції без аргументів: int func2();

Якщо функція не повертає значення безпосередньо, то замість типу вказується ключове слово **void**. Така функція називається **процедурою**.

Приклад об’яви процедури: void proc1(int p1, int p2);

Компілятор визначає **функції як різні**, якщо вони відрізняються назвою або кількістю аргументів або типом хоч одного з аргументів.

На кожну об’яву має бути відповідна реалізація функції, яка відрізняється тим, що список параметрів не закінчується крапкою з комою, а закінчується блоком коду — реалізацією функції, яка розміщена у фігурних дужках. При цьому назва, кількість і типи аргументів у опису та реалізації повинні збігатися. Об’ява функції та реалізація можуть співпадати.

Безпосередній результат роботи функції повертається за допомогою оператора **return**, за яким слідує об'єкт або вираз, значення якого буде повернуто, і крапка з комою. При цьому тип об'єкта або виразу повинен співпадати з типом функції або автоматично наводитися до нього.

Для процедур оператор return не обов'язковий, і застосовується для дострокового виходу з процедури в скороченій формі — **return;**

Приклад об’яви і реалізації функції підсумовування двох цілих чисел:

```
int add( int a, int b );
int add( int a, int b ) { return a + b; }
```

Значення аргументів a і b буде підсумовано і сума буде повернута з функції.

3.1 Передача аргументів до функції

Існує п'ять основних варіантів передачі аргументів до функції (процедури):

1) За значенням. В цьому випадку передається значення аргументу, що передбачає копіювання при виконанні функції. Такий варіант передачі позначається як пара: тип, значення. І по суті такий аргумент є локальною змінною функції, яка при виклику ініціалізується переданим значенням і час її існування обмежено часом виконання функції.

Приклад передачі аргументів за значенням:

```
int func_by_value( float x, int y, bool z );
```

Так як при виконанні функції з такими аргументами відбувається копіювання даних, доцільно передавати за значенням тільки вбудовані типи даних (або дані невеликого розміру).

2) За посиланням. В цьому випадку передається не значення, а адреса (посилання) на те місце де знаходиться об'єкт. Для таких аргументів праворуч, після типу аргументу, дописується знак **&**.

Приклад передачі аргументів за посиланням:

```
int func_by_ref( float& x, int& y, bool& z );
```

В цьому випадку копіювання не проводиться, і такий спосіб передачі може бути застосовний до об'єктів будь-якого виду і розміру, без втрат продуктивності. Треба пам'ятати, що зміна значення таких аргументів всередині функції, веде до зміни тих об'єктів, з якими пов'язані посилання.

3) За константної посиланням. Аналогічно до попереднього варіанту, але, перед типом аргументу додається ключове слово **const**, яке захищає об'єкт, з яким пов'язане посилання, від модифікації. Для передачі чистих аргументів цей варіант кращий за попередній.

Приклад передачі аргументів за константним посиланням:

```
int func_by_cref( const float& x, const int& y, const bool& z );
```

4) По покажчику. У цьому випадку, як і для посилання, передається не значення, а покажчик (адреса) об'єкту. Для таких аргументів праворуч, після типу аргументу, дописується знак *.

Приклад передачі аргументів за показчиком:

```
int func_by_ptr( float* x, int* y, bool* z );
```

Для доступу до об'єктів, що передані за показчиком, потрібно використовувати операцію розіменування, яка позначається символом * зліва від назви покажчика.

Приклад доступу до об'єктів через розіменування покажчика:

```
*x = 1.f; *z = true; int i = *y;
```

У цьому випадку також копіювання не проводиться, і такий спосіб передачі, може бути застосовний до об'єктів будь-якого типу і розміру, без втрат продуктивності. Як і для посилань, зміна значення таких аргументів всередині функції, веде до зміни тих об'єктів, з якими пов'язані покажчики.

Зазвичай передача через покажчики використовується для масивів. Фактично **масив і покажчик** в мові C++ є синонімами, і ім'я масиву є покажчиком на його перший елемент. Таким чином, передавши цей покажчик можна отримати доступ до всіх елементів масиву за допомогою операції [ ] зсередини функції або процедури.

5) За константним покажчиком. Аналогічно до попереднього варіанту, але, перед типом аргументу додається ключове слово **const**, яке захищає об'єкт, з яким пов'язаний покажчик, від модифікації. Для передачі масивів, з яких дані тільки зчитуються, цей варіант кращий за попередній.

Приклад передачі аргументів по константним покажчиком:

```
int func_by_cptr( const float* x, const int* y, const bool* z );
```

Зверніть увагу, що для масиву в такому випадку розмір не передається і його необхідно передати окремим аргументом за значенням.

3.2 Повернення результатів з функції

Існує три основні варіанти повернення результатів з функції, два з яких можна застосувати й для процедур:

1) Безпосередньо, такий варіант найкращий, але істотним обмеженням є те, що повертається тільки один об'єкт. Також цей варіант неприпустимий для процедури. Повернення значення проводиться за допомогою оператора **return**, в будь-якому місці функції. Всі шляхи виконання коду для функції повинні закінчуватися оператором return, який повертає об'єкт з типом, що збігається або автоматично приводиться до типу вказаному в об’яві функції.

Приклад безпосереднього повернення результату:

```
float func_max( float a, float b ) { return a < b ? b: a; }
```

2) За посиланням. З попереднього розділу нам відомо, що модифікація значення для об'єкта переданого за посиланням призводить до того що змінюється значення об'єкта пов'язаного з посиланням зовні функції. Це дозволяє повертати результати з функції, змінюючи (присвоюючи нові) значення для аргументів переданих за посиланням. Такий спосіб повернення результатів також підходить і для процедур.

Приклад повернення результату за посиланням:

```
void proc_max_ref_ret( float a, float b, float& r ) { r = a < b ? b : a; }
```

В об'єкті на який посилається r буде збережено результат роботи цієї процедури, максимум з аргументів a і b.

3) По покажчику. Працює аналогічно до попереднього метода, але застосовується для повернення масивів.

Приклад:

```
void proc_max_ptr_ret( float* r, int N, float v )
{ for( int i = 0; i < N; ++i ) r[i] = v; }
```

Масив з N елементів, на який вказує r, буде заповнений значенням аргументу v.

3.3 Виклик функцій і процедур

Для того щоб код функції було виконано, її опису і реалізації недостатньо. Компілятор гарантує виконання тільки однієї функції з умовною назвою main (точки входу програми). Відповідно, інші функції і процедури повинні викликатися або з main або з інших функцій і процедур які в свою чергу викликаються з main.

Виклик функції складається з назви функції, і значень аргументів які в неї передаються, поміщених в круглі дужки, і закінчується крапкою з комою. **Типи даних не вказуються.**

Приклад: add(5, 10);

Значення аргументів передаються по порядку їх перерахування, для аргументів за посиланням і за значенням можна передавати безпосередні значення, константи і константні об'єкти. А для посилань і покажчиків тільки об'єкти або адреси (знак & зліва від назви об'єкта або ім'я масиву).

Приклад:

```
float m; bool b; int A[10];
func_by_ptr( &m, A, &b );
```

Значення, що безпосередньо повертається з функції, доцільно зберігати або передавати в іншу функцію. Для збереження, зазвичай, присвоюють значення, що повертається, якійсь змінній або константі такого ж типу, як і тип функції.

Приклад:

```
int res = 0; res = add( 5, 10 );
const int res = add( 5, 10 );
```

У мові C++ функції можна перевизначати з одним і тим самим іменем, але різною кількістю або типом аргументів (тип, що повертає функція, не враховується), в такому випадку компілятор вважає їх різними. Також для аргументів функції можна задати значення за замовчуванням, які будуть використані, якщо при її виклику не було передано відповідні аргументи. Для цього, після імені аргументу (в об’яві функції або в реалізації, якщо вони співпадають), треба вказати значення через знак =.

Приклад:

```
int add( int a, int b = 10 );
int add( int a, int b ) { return a + b; }
int x = add( 5 );
```

При такому скороченому виклику, аргумент b буде дорівнювати 10, і після виклику функції, змінна x дорівнюватиме 5 + 10 = 15.

3.4 Рекурсія і цикли

Рекурсією в програмуванні називається виклик функції з самої себе. Рекурсія називається **кінцевою**, в тому випадку, якщо вона закінчується викликом самої себе, в такому випадку компілятор може побудувати більш ефективний код, з точки зору економії пам'яті не дублюючи аргументи. По суті рекурсія, як і цикли, дозволяє повторити один і той же код кілька разів, але з різними початковими даними. Таким чином, цикли і рекурсія взаємозамінні.

Розглянемо функцію з циклом, який обчислює добуток елементів масиву більших ніж 0:

```
float prod_gt0( const float* A, int N )
{
    float prod = 1.f;
    for( int i = 0; i < N; ++i )
    {
        if( A[i] > 0.f )
        {
            prod *= A[i];
        }
    }
    return prod;
}
```

Замінимо цикл рекурсією, для цього необхідно:

1) Прибрати оператор циклу (тільки сам оператор!).
2) Винести змінну циклу в список аргументів.
3) Перенести умову виходу в початок функції, змінивши її на протилежну.
4) Додати рекурсивний виклик зі зміненим значенням змінної циклу.

Рекурсивна реалізація функції буде такою:

```
float prod_gt0_rec( const float* A, int N, int i = 0 )
{
    if( i >= N ) return 1.f;
    float prod = A[i] > 0.f ? A[i] : 1.f;
    return prod * prod_gt0_rec( A, N, i + 1 );
}
```

Виклик функції виглядає однаково як для рекурсивної так і для нерекурсивної версії:

```
float res = prod_gt0_rec( A, N );
```

Для переходу від рекурсії до циклу треба виконати наведені вище кроки у зворотному порядку.

3.5 Обробка виняткових ситуацій

У мові С++ існують спеціальні оператори для обробки виняткових ситуацій (виключень), які можуть виникнути в програмі через:

- нестачу пам'яті;
- помилки, пов'язані з апаратним забезпеченням;
- відсутність даних;
- обмеження доступу (робота з файлами);
- помилки в використаних бібліотеках і іншому ПЗ.

У деяких випадках механізм обробки виключень можна використовувати для повернення результатів з функцій, але це може бути недоцільно з міркувань продуктивності.

Основне призначення **механізму обробки виключень** — виявити виняткову ситуацію і передати її в те місце (або місця), де вона може бути коректно оброблена. Найчастіше такі ситуації виникають всередині функцій, які самостійно не можуть їх обробити, це може зробити тільки функція вищого рівня.

Для виявлення виняткових ситуацій, блок коду зазначається ключовим словом **try**, за яким у фігурних дужках слідують оператори, в яких може виникнути виняткова ситуація.

Після цього блоку (після закриваючої фігурної дужки) в цій же функції або в функції вище за ієрархією, повинен слідувати блок (після виклику цієї функції) що починається оператором **catch**, після якого в круглих дужках слідує тип виключень, які будуть оброблятися. Якщо потрібно обробити всі виняткові ситуації, в круглих дужках пишеться три крапки. Кожне виключення має свій тип, і може бути повторно передано з блоку catch за допомогою оператора **throw**, після якого вказується значення, яке описує суть (код, номер або інші дані) та тип переданого виключення. Залежно від його типу, воно потрапить в ті блоки catch, які розташовані далі по тексту програми і в заголовку яких вказаний такий же тип.

Оператор **throw** так само дозволяє створити виняткову ситуацію прямо в програмі. Цей механізм може бути застосовано для передачі результатів з функцій.

Приклад обробки виняткових ситуацій:

```
try
{
        Process();
}
catch( int ex_int ) { }
catch( … ) { }
```

В наведеному коді, перевіряється виникнення виключення в функції Process() за допомогою блоку try. Якщо виключення виникло і має тип int, то виконання програми перейде в блок catch(int ex_int), а якщо виключення матиме інший будь який, але відмінний від int, тип, то в блок catch(...).

Приклад створення виняткової ситуації:

```
try
{
        int x = add( a, b );
        if( x > 5 ) throw 0;
        if( x < -5 ) throw 1;
}
catch( int res ) {}
```

Функція add додає два числа, які задані в змінних a і b, якщо результат більше за 5, то генерується виключення типу int з кодом 0, а якщо результат менше -5, то з кодом 1.

За замовчуванням типом літеральних констант без префіксів, що складаються тільки з цифр, є тип int.

У блоці catch виключення буде оброблено (якщо воно виникло), а аргумент int res буде містити його код, 0 або 1, залежно від того що сталося. Таким же чином можна передавати результати різних типів з функцій і процедур.
Більш складний приклад, з подальшою передачею виключення:

```
try
{
        int x = add( a, b );
        if( x > 5 ) throw 0;
        if( x < -5 ) throw 1;
}
catch( int res )
{
        if( res > 0 ) throw (long)-1;
 }
```

У цьому випадку частина виключень з кодами менше або рівними 0 буде оброблена на місці, а частина (з кодами більше за 0) буде передана далі і вже буде мати код -1 і тип long.

Індивідуальні завдання

1. Створити функцію, приймаючу в якості аргументів масив дійсних чисел, яка повертає суму чисел послідовності зведену в квадрат в якості результату.

2. Створити функцію що обчислює факторіал числа N.

3. Написати функцію яка обчислює факторіал числа N за допомогою рекурсії.

4. Написати функцію, яка б заміняла в масиві символів всі символи 'a' на символ '1'.

5. Створити функцію приймаючу в якості аргументів два дійсні числа a та b, яка повертає різницю цих чисел в аргумент c, а суму чисел в аргумент d.

6. Написати функцію, яка обчислює всі цифри цілого числа в двійковій системі числення.

7. Створити рекурсивну функцію, яка б підраховувала середнє арифметичне елементів масиву з N дійсних чисел.

8. Створити функцію приймаючу в якості аргументів беззнакові цілі числа a та b, і повертала остачу від ділення цих чисел плюс 5 в якості результату. Перевірити дільник на 0 і видати повідомлення про помилку за допомогою виключення.

9. Створити функцію приймаючу в якості аргументів два дійсних числа x та y, що повертає різницю цих чисел мінус 3 в аргумент e. Якщо число негативне - викликати виключення.

10. Замінити цикл в наведеній нижче функції на рекурсію:

```
int sum_range( const int* A, int N, int L, int R )
{
        int s = 0, i = 0;
        while( i < N && A[i] >= L && A[i] <= R ) { s += A[i]; }
        return s;
}
```

11. Написати функцію, яка підраховує логічну суму елементів масиву з логічних величин.

12. Написати функцію, яка змінює порядок проходження елементів масиву на зворотний, не використовуючи додаткової пам'яті більш ніж на 1 елемент.

13. Замінити рекурсію в наведеній нижче функції на цикл for.

```
int count_rec( const double* A, int N, int i )
{
        if( --i < 0 ) return 0;
        return ( A[i] > 0 ) + count_rec( A, N, i );
}
```

14. Написати функцію, яка перемішує вміст двох масивів символів, беручи значення з першого, а потім другого масиву по черзі, результат повинен бути поміщений в третій масив вдвічі більшої розмірності.

Контрольні питання

1. Як виглядає об’ява функції?
2. Чим відрізняється функція від процедури?
3. Яка різниця між реалізацією і об’явою функції?
4. Для чого використовується оператор return в процедурах і функціях?
5. Які основні варіанти передачі аргументів до функції?
6. Які особливості передачі аргументу за посиланням?
7. Для чого використовується передача аргументу за покажчиком?
8. Які особливості передачі аргументу за значенням?
9. Які існують варіанти повернення результатів з функції?
10. Які існують варіанти повернення результатів з процедури?
11. Чим відрізняється передача аргументу за значенням і за константним посиланням?
12. Для чого потрібен виклик функцій?
13. Як зберегти результат що повертається функцією?
14. Чим обмежений час існування аргументів функції переданих за значенням?
15. У якому випадку компілятор вважає функції різними?
16. Що таке аргументи функції за замовчуванням і як їх визначити?
17. Для чого призначена обробка виняткових ситуацій і як вона працює?
18. Як в програмі створити виняткову ситуацію?
19. Що таке рекурсія і для чого вона застосовується?
20. Як замінити цикл рекурсією?
21. Як замінити рекурсію циклом?
22. Як зберегти значення локальної змінної функції до наступного виклику?
23. Чим відрізняється передача аргументів за покажчиком і за значенням?
24. Чим відрізняється виклик функції від її об’яви?

4 СТРУКТУРИ ТА КОРИСТУВАЦЬКІ ТИПИ ДАНИХ

Мова С++ має широкі можливості для об’яви різноманітних, користувацьких типів даних, починаючи від найпростіших масивів, закінчуючи складними структурами і класами. Для користувацьких типів можна задати будь-яку поведінку та структуру, а також зробити їх такими що не відрізняються від вбудованих типів даних.

4.1 Структури і класи

У мові С++ структура і клас є синонімами і відрізняються тільки доступом за замовчуванням до полів і методів. *Надалі будуть розглядатися структури, але все написане про них справедливо і для класів.*

Структура це тип даних, який об'єднує об'єкти різних типів (**поля**), які розташовані в пам'яті послідовно, та функції, які їх обробляють (**методи**).

Для об’яви структури (або класу) використовується ключове слово struct (class), після якого слідує назва структури, і в фігурних дужках опис полів (тип і назва) і методів (функції і процедури) структури, яке закінчується крапкою з комою. Таке визначення додає новий користувацький тип даних в програму, а також простір імен з такою ж назвою як у структури.

Наприклад:

```
struct s_temp
{
        int i; char c;
};
```

Цим кодом додається новий тип, структура з назвою s_temp, яка містить два поля: і типу int та с типу char. Після додавання нового типу, його можна використовувати як вбудований тип даних, для об’яви будь якого об'єкта:

Наприклад: s_temp S;

У програмі буде визначена змінна S користувацького типу s_temp.
Так як типи даних полів структури і, як наслідок, їх розміри можуть бути різні (*більш докладно про розташування елементів структур і масивів в пам'яті див. у додатку А*), то доступ за номером для структури недоцільний. І для доступу до полів структури використовується **оператор точка** у об'єкта структури, після якої необхідно записати ім'я поля.

Приклад: S.i = 10; char cc = S.c;

Структури можна об'єднувати в масиви, також масиви можуть бути полями структур. При цьому доступ до елементів йде в порядку ієрархії об’яв.

Приклад:

```
struct s_complex
{
        char c;
        s_temp temp[10];
};

s_complex C[10];
char cc = C[0].c;
C[3].temp[5].i = 10;
```

У структурі можна задавати **конструктори** — методи, які ініціалізують поля початковими значеннями; оператори, а також регулярні методи, які виконують маніпуляції з полями структури. Назва конструктора має збігатися з назвою структури, після аргументів у конструктора, через двокрапку йде **список ініціалізації**, в якому перераховані поля і в круглих дужках ті аргументи або значення якими вони мають бути ініціалізовані.

Приклад структури з конструктором:

```
struct s_temp_ctr
{
        int i;
        char c;
        s_temp_ctr( int i_, char c_ ): i( i_ ), c( c_ ) {}
};
s_temp_ctr S( 10, ‘A’ );
```

Поля i та c структури S будуть ініціалізовані значеннями: 10 і кодом символу 'A'.
Існує спеціальна версія конструктора, **конструктор копіювання**, який використовується для створення копій об'єктів. Такий конструктор завжди приймає один аргумент — константне посилання на об'єкт такого ж типу, і виглядає наступним чином:

```
s_temp_ctr( const s_temp_ctr& to_copy );
```

Якщо такий конструктор не об'явлено, то компілятор створює конструктор копіювання за замовчуванням, який здійснює побітове копіювання об'єкта.

Для користувацьких типів даних також можна визначати оператори за допомогою ключового слова **operator**. Приклад оператора порівняння на менше та методу accum, для накопичення значень в полі структури i:

```
struct s_temp_met
{
        int i;
        bool operator<( const s_temp& t ) const { return i < t.i; }
        int accum( int k ) { i += k; return i; }
};
```

Для тих методів і операторів, які не передбачають зміну полів структури, доцільно писати ключове слово **const** після списку параметрів.

4.2 Об'єднання

Ці конструкції мови дуже схожі на структури, але у об'єднань всі поля починаються з одної й тієї ж адреси у пам'яті, а не послідовно як у структури. Об'єднання додається ключовим словом **union**.

Приклад:

```
union u_temp
{
        int i;
        char c;
};
```

Якщо записати в поле і якесь значення, то прочитавши значення з поля c, можна отримати один байт (молодший на процесорах Intel) записаного в і значення.

Приклад: u_temp U; U.i = 0x2520; char ch = U.c;

В змінної ch буде значення 0x20. Це справедливо за умови, що розмір типу char дорівнює 8 біт, і використовується порядок зберігання слів, починаючи з молодшого байта, як у процесорів Intel.

Об'єднання застосовуються якщо необхідно отримати доступ до формату вбудованих типів даних, або для специфічного перетворення типів даних. Об'єднання можуть бути полями структур, елементами масивів і навпаки.

4.3 Перерахування

Перерахування (перелічувальний тип) це **набір логічно пов'язаних констант**. За замовчуванням такий набір має тип int. Позначається ключовим словом **enum**. У фігурних дужках перераховуються імена констант (вони всі мають один тип). Перша константа має значення 0, наступна на одиницю більше і так далі. Поточне значення можна змінити, використовуючи знак = і вказавши інше значення.

Приклад констант що описують місяці року:

```
enum e_months
{
        jan = 1,
        feb,
        mar = 5,
        apr,
        ...
        dec,
};
```

Скрізь де в програмі будуть використані імена з цього перерахування, компілятор підставить відповідні їм константи. Якщо перерахування має назву (можна визначати і без назви), то в програму додається новий тип даних з такою назвою.

Застосування перерахувань доцільно при об'яві множини логічно пов'язаних констант.

4.4 Шаблони

Для об’яви шаблону застосовується ключове слово **template**, за яким в кутових дужках вказані імена параметрів шаблону, після чого наведено тип та ім'я шаблону. Для параметрів також вказується, що це таке: тип (typename) або значення (наприклад, int). Тип самого шаблону може бути класом (class), структурою або функцією.

Наприклад, найпростіший шаблонний клас:

```
template <typename T> class A { void process( T& t ) {}; };
```

Шаблонна функція:

```
template <class T> const T& max( const T& a, const T& b )
{ return ( b < a ) ? a : b; }
```

Для завдання параметризованого типу використовується ключове слово **typename**, яке є синонімом слова class. Таким чином, шаблон дозволяє змінювати не тільки значення аргументів, а ще й типи. Шаблони повинні бути об’явлені в заголовкових файлах.

Якщо необхідно, щоб для якогось типу шаблон працював інакше ніж для інших, застосовується спеціалізація, яка виглядає так само, як об’ява, але параметри шаблону в кутових дужках не вказуються, а замість цього задаються конкретні аргументи шаблону після його імені:

```
template<> class A<int> {};
```

Після цього можна описати методи і поля спеціальної реалізації для int. Ці методи і поля можуть бути такими, яких немає в первісному шаблонному класі. Така спеціалізація називається **повною**.

Таким чином, в програму додається новий шаблонний клас, який може мати свої окремі, не пов'язані з вихідним шаблоном параметри, які обов'язково повинні використовуватися.

```
template<typename W, typename S> class A<int> {};
template<typename W, typename S> class A<int>
{
        void process2( W& w, S& s ) {}
};
```

Перша об’ява не буде компілюватися! При цьому новий клас можна спеціалізувати через додаткові параметри, і така спеціалізація називається частковою.

```
template<typename W, typename S> class A< B<W, S> > {};
template<typename S> class A<int, S> {};
```

4.5 Директива typedef

Об’ява шаблонів та інших користувацьких типів іноді виглядає досить громіздко. Для того щоб зробити програму більш зрозумілою, доцільно в таких випадках давати скорочені — більш прості назви типів. Для цих цілей служить директива **typedef** — вона дозволяє задати для існуючого типу даних нове (зазвичай коротше) ім'я. При цьому новий тип даних не вводиться, і обидва імені можуть бути використані спільно. Після директиви вказується «старе» ім'я, а потім, через пробіл, нове, а в кінці крапка з комою.

Приклад:

```
typedef unsigned long t_ulong;
typedef A<unsigned int> t_auint;
```

Імена t_ulong і t_auint можуть бути використані замість unsigned long і A<unsigned int> відповідно, в будь-якому місці програми.

Індивідуальні завдання

1. Визначте структуру для зберігання таких даних: назва виробу, колір, вага і ціна.

2. Визначте шаблонну функцію для зведення значення довільного типу даних в квадрат.

3. Визначте структуру, яка може зберігати одночасно значення одного з типів: int, float, double або bool використовуючи мінімально можливу кількість пам'яті, а також номер, який однозначно вказує на тип що зберігається.

4. Визначте клас для роботи з комплексними числами, об'явить для нього конструктор, оператор порівняння на менше за дійсною частиною, а також оператор додавання.

5. Визначте структуру для зберігання таких даних: 5-ть оцінок, прізвище, ім'я, номер групи. Визначте для такої структури конструктор копіювання і оператор присвоювання.

6. Визначте перелічувальний тип, який би об'єднував римські цифри від 1 до 20 включно.

7. Визначте структуру для зберігання таких даних: координати на площині і назва.

8. Визначте шаблонну функцію для обчислення квадратного кореня для довільного типу даних.

9. Визначте структуру, що складається з двох полів a та b. Напишіть функцію, в яку передається ця структура як аргумент, та в поле a записується значення поля b.

10. Визначте перелічувальний тип даних з константами для угруповання кольорів: синій, червоний, зелений, коричневий, білий. В якому константа білого кольору повинна мати значення 0x200.

11. Визначте об'єднання, яке б дозволяло отримувати доступ до даних типу float як до байтів. Відомо, що тип char має розмір 8 біт, а тип float 32 біта.

12. Визначте структуру для зберігання вектора координат в 3-х мірному просторі. Визначте для цієї структури оператор +, для поелементного додавання двох таких векторів.

Контрольні питання

1. Що таке структура?
2. Чим відрізняється клас від структури?
3. Що таке конструктор, як він об'являється і для чого потрібен?
4. Що таке методи, як вони визначаються?
5. Які операції з полями структури і класу не може виконати метод, об'явлений з ключовим словом const?
6. Як здійснюється доступ до полів структури?
7. Що таке об'єднання?
8. Чим об'єднання відрізняються від структур?
9. Що таке перерахування?
10. Для чого застосовуються шаблони?
11. Які види спеціалізації шаблонів бувають і чим вони відрізняються?
12. Для чого застосовується директива typedef?
13. Для чого застосовуються об'єднання?
14. Чим структура відрізняється від масиву?
15. Що таке конструктор копіювання і для чого він застосовується?
16. Який розмір буде у структури, в якій не визначено жодного поля?
17. Що таке конструктор копіювання і як він визначається?
18. Що таке часткова спеціалізація шаблону?

5. ВВЕДЕННЯ-ВИВЕДЕННЯ

Для введення-виведення даних різного типу в мові C++ використовуються так звані потоки введення-виведення. У мові визначені два стандартних потоки **std::cout** і **std::cin**, які призначені для виведення на екран (консоль) і введення з терміналу (клавіатури) відповідно. Для роботи з потоками необхідно підключити бібліотеку iostream. Для виведення використовується потік типу ostream, а для введення — istream.

Для виведення в потік використовується оператор виведення **<<**, а для введення оператор **>>**.

Наприклад: std :: cout << 10 << "ABCD";

Виведе на консоль рядок 10ABCD.

int x; std :: cin >> x;

Чекатиме введення користувачем цілого числа, введення відбудеться після натискання кнопки Enter.

Для роботи із строковими даними доцільно застосовувати спеціалізований строковий тип даних **std::string**, для чого потрібно підключити бібліотеку string.

Наприклад, введення рядка довільної довжини з клавіатури буде виглядати так:
std::string s;
cin >> s;

Для отримання рядка у вигляді покажчика на масив символів (const char*) призначений метод c_str(). Для строкового типу даних визначено оператори введення-виведення в потоки, а також оператор '+' для складання рядків. Конструктор std::string дозволяє створити рядок з масиву символів.

Для управління форматом виведення в потік використовуються так звані маніпулятори введення-виведення, це спеціальні об'єкти-функції, які змінюють стан потоку при своєму виклику. Для роботи з маніпуляторами необхідно підключити бібліотеку **iomanip**. Виклик відбувається за допомогою оператора виведення **<<**, як і для звичайного тексту.

Наприклад: std::cout << std::endl;

Цей код здійснює переклад рядка та очищення буфера потоку, що призведе до відправлення даних що були буферізовані — на екран або збереженню в файл.

Основні маніпулятори:

1) std::setw(int n) — задає розмір колонки в n символів, діє тільки на наступне поле;

2) std::fixed, std::scientific — перемикають режим виведення чисел з плаваючою комою між форматом з фіксованою комою і «науковим представленням»;

3) std::setprecision(int p) — задає точність (кількість символів після коми) для дійсних чисел;

4) std::right, std::left — задає вирівнювання в колонці по правому або лівому краю відповідно;

5) std::setfill(char c) — задає символ заповнювач для порожніх клітинок колонки, за замовчуванням використовується пробіл.

6) std::setbase(int x) — задає систему обчислення в якій будуть виводитися цілі числа, для 8-ми, 10-ти і 16-ти річних систем є окремі маніпулятори std::oct, std::dec і std::hex;

7) std::showbase, std::noshowbase — включає і відключає показ префікса системи обчислення для цілих чисел.

Приклад виведення форматованої таблиці:

```
#include <iostream>
#include <iomanip>
using namespace std;
struct s_human
{
        char    name[30];
        float  weight;
};
cout << fixed << setprecision( 2 ) << dec << setfill( ‘ ‘ );
for( int i = 0; i < N; ++i )
{
        cout << right << setw( 3 ) << ( i + 1 ) << “. ”;
        cout << left << setw( 30 ) << A[i].name << “ “;
        cout << right << setw( 6 ) << A[i].weight << endl;
}
```

Даний фрагмент програми виводить масив структур s_human на екран, у вигляді таблиці: номер елемента, назва і числова характеристика.

Для користувацьких типів даних можна визначити оператори введення і виведення (як окремі функції). Такий оператор повертає посилання на потік, а в якості аргументів приймає посилання на потік і на користувацький тип.

Наприклад:

```
struct s_data
{
        int x;
        float y;
};
ostream& operator<<( ostream& s, const s_data& d )
{
        s << x << y;
        return s;
}
```

```
istream& operator<<( istream& s, s_data& d )
{
        s >> x >> y;
        return s;
}
```

5.1 Робота з файловими потоками

Загалом, принцип роботи з файловими потоками не відрізняється від роботи зі стандартними потоками cin та cout, але все ж таки має деякі технічні особливості. Для роботи з файловими потоками необхідно підключити бібліотеку **fstream**, а також створити об'єкт потоку. Для цього потрібно визначити змінну типу fstream і в аргументах конструктора задати шлях до файлу і прапори, які визначають бажаний режим роботи.

Наприклад:

```
std::fstream f( "test.txt", std::ios_base::out | std::ios_base::trunc );
```

У цьому випадку файл test.txt буде відкритий для запису (прапорець out) і якщо він існував раніше, то його вміст буде видалено (прапорець trunc).

Прапорці, які задають режим роботи, знаходяться в просторі імен **std::ios_base** і об'єднуються за допомогою оператора побітового «або»: |.

За замовчуванням файл відкривається як текстовий, для того щоб відкрити його як двійковий, необхідно вказати прапорець **binary**. Для відкриття на читання треба вказати прапорець in, для дозапису в кінець файлу — прапорець app.

Для того щоб перевірити чи успішно відкрито файл, об'єкт файлового потоку має метод is_open(), що повертає значення true, якщо файл відкритий вдало і можлива подальша робота з ним. Після закінчення роботи з файлом необхідно викликати метод close(). Але він так само викликається автоматично, коли змінна файлового потоку вийде з зони видимості.

Для читання неформатованих двійкових даних з файлу призначений метод read(), який приймає два аргументи — покажчик на буфер, куди повинні бути поміщені прочитані дані, і кількість елементів типу char, які необхідно прочитати з файлу. Метод повертає актуальну кількість прочитаних елементів.

Для запису неформатованих двійкових даних призначений метод write(), який, як і read(), приймає два аргументи — покажчик на константний буфер, який є джерелом даних для запису до файлу, і кількість елементів типу char, які потрібно записати в файл. Метод повертає актуальну кількість записаних елементів.

Для перевірки, чи можна ще зчитувати дані з файлу, застосовується метод eof(), який повертає true в тому випадку, якщо більше не має даних для зчитування — досягнута кінцівка файлу.

Приклад зчитування цілих чисел з файлу з подальшим їх виведенням на екран:

```
#include <fstream>
using namespace std;
fstream f( “input.bin”, ios_base::in | ios_base::binary );
if( f.is_open() )
```

```
{
    while( !f.eof() )
    {
        int x;
        if( !f.read( (char*) &x, sizeof( x ) ) ) throw “Error”;
        cout << x << endl;
    }
    f.close();
}
```

Приклад запису масиву дійсних чисел до текстового файлу:

```
#include <fstream>
using namespace std;

double A[N];

fstream f( “output.txt”, ios_base::out | ios_base::trunc );
if( f.is_open() )
{
    f << fixed << setprecision( 3 ) << setfill( ‘ ‘ );
    for( int i = 0; i < N; ++i )
    {
        f << A[i] << endl;
    }
    f.close();
}
```

Індивідуальні завдання

1. Напишіть функцію, яка виводить зведені в квадрат значення числової послідовності 1, 2, 3, 4 ... M, на екран. Число M вводить користувач з клавіатури.

2. Напишіть функцію, яка виводить зведені в квадрат значення числової послідовності 1, 3, 5, 7, 9 ... M, в текстовий файл. Число M вводить користувач з клавіатури.

3. Напишіть функцію, яка зберігає в текстовий файл введені користувачем (з клавіатури) дані у вигляді форматованої таблиці: номер запису (формується автоматично починаючи з 1), прізвище, зріст, вага.

4. Напишіть функцію, яка підраховує кількість символів 'A' в текстовому файлі.

5. Створіть функцію, яка видаляє вміст файлу. Файл вводиться користувачем з клавіатури.

6. Визначте клас для роботи з комплексними числами, об'явить для нього оператори введення-виведення.

7. Реалізуйте функцію, яка виводить на екран масив структур, який передається через аргументи функції, у вигляді таблиці. Структура містить: номер телефону, прізвище, ім'я.

8. Напишіть функцію, яка підраховує кількість символів 'a' та 'b' в текстовому файлі. Результати підрахунку необхідно вивести на екран.

9. Визначте функцію, яка створює новий порожній файл. Данні до файлу та його ім’я вводяться користувачем з клавіатури.

10. Реалізуйте функцію, яка створює новий файл і записує в нього рядок "ABCDEF". Ім’я файлу вводиться користувачем з клавіатури і передається як аргумент функції.

11. Напишіть функцію, яка підраховує кількість входжень цифр до текстового файла. Результати підрахунку необхідно вивести на екран у вигляді таблиці: цифра, кількість входжень.

12. Реалізуйте функцію, яка зберігає в текстовий файл масив структур, який передається через аргументи функції, у вигляді таблиці. Структура містить: номер телефону, прізвище, ім'я, рейтинг (у вигляді дійсного числа).

13. Напишіть функцію, яка зберігає в текстовий файл значення функції sin(x), для x від 0 до π, з кроком 0,1.

14. Напишіть функцію, яка виводить на екран значення функцій sin(x) та cos (x), для x від -π до π, з кроком 0,15.

Контрольні питання

1. Як здійснюється введення-виведення даних?
2. Чим відрізняється виведення даних до файлу від виведення даних в стандартний потік std::cout?
3. Що таке маніпулятори введення-виведення?
4. Як задати кількість цифр після коми для виведених на екран дійсних чисел?
5. Як задати вирівнювання в колонці таблиці при форматованому виведення в файл?
6. Як записати 10 байт в двійковий файловий потік?
7. Як зчитати 100 байт з двійкового файлового потоку?
8. Як зчитати будь-яку кількість елементів будь-якого типу з файлового потоку?
9. Чим відрізняється форматований вивід даних в текстовий потік, від запису в бінарний потік?
10. Як визначаються оператори введення-виведення для користувацьких типів даних?
11. Як зберегти структуру даних в текстовий файл?
12. Як зчитати структуру даних з текстового файлу?
13. Як зчитати масив із двійкового і текстового файлів?
14. Як зберегти масив структур в двійковий файл?
15. Як зберегти масив структур в текстовий файл?

ДОДАТОК А

Особливості розташування елементів структур в пам'яті

Адреси об'єктів вбудованих типів даних в пам'яті вирівнюються кратно їх розмірам. Тобто, якщо розмір типу int становить 4 байта, то, об'єкт такого типу буде розташовано за адресою кратною 4, для double (розміром 8 байт) за адресою кратною 8 і т. ін. Така поведінка задана за замовчуванням в налаштуваннях компілятора, і пов'язана з особливостями доступу до ОЗП в сучасних мікропроцесорах — доступ до даних що вирівняні проводиться швидше, ніж до тих які не вирівняні. Структура даних вирівнюється кратно за розміром найбільшого поля яке входить в цю структуру.

Зміну характеру вирівнювання може бути проведено в настройках проекту (для всього проекту), або директивою #pragma pack, для певних структур даних. Але в більшості випадків така зміна недоцільна, і може привести до істотного уповільнення роботи програми.

Приклад установки вирівнювання на 1 байт для структури:

```
#pragma pack( push, 1 )
struct s_unalign
{
    char c;
    int i;
};
#pragma pack( pop )
```

Перша директива зберігає значення вирівнювання на стеку компілятора (push), а потім встановлює вирівнювання на 1 байт. А друга відновлює раніше збережене значення (pop) зі стека. Застосовувати таке строге завдання вирівнювання доцільно тільки в разі забезпечення сумісності з будь-яким існуючим форматом зберігання даних або інтерфейсом обміну даними між програмами.

Розглянемо особливості зберігання в пам'яті структур даних на прикладі структури:

```
struct s_align
{
    char c;
    int x[2];
    double d;
};
```

Максимальний розмір поля цієї структури дорівнює 8, це поле типу double (64 біта, 8 байт). Це значить, що структура в пам'яті буде розташована за адресою кратною 8 байтам. На рисунку А.1 показано розташування полів вирівняної та невирівняної структури.

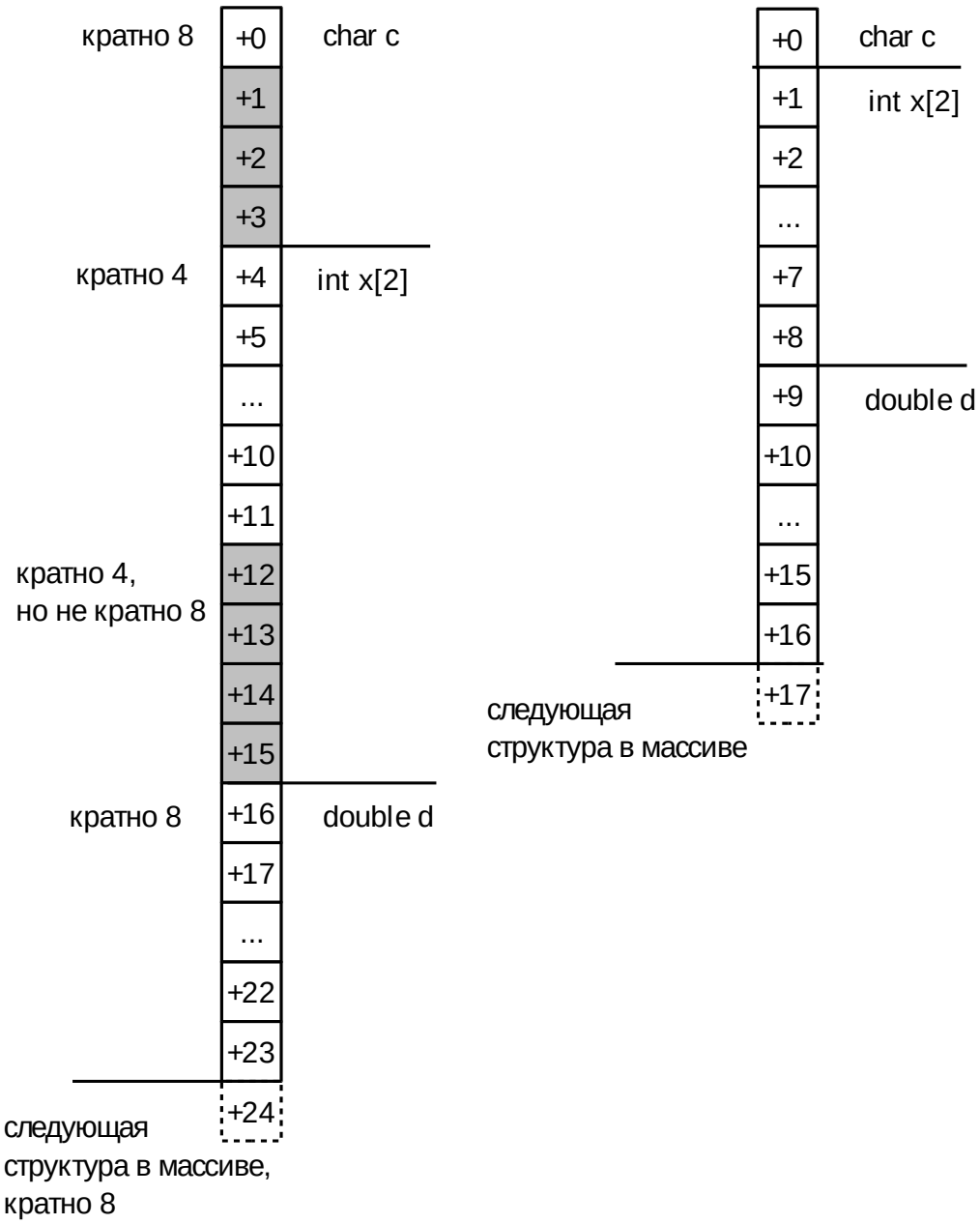

Рисунок А.1 – Розташування полів вирівняної і невирівняної структури в пам'яті.

Зверніть увагу, що для вирівняної структури реальний розмір в загальному випадку не збігається з сумою розмірів окремих полів. А між полями різного розміру, можуть виникати «дірки», які часто заповнені різними випадковими даними. Це слід враховувати при збереженні даних в двійковому форматі, а також розрахунку різних метрик даних, який проводиться на побітовому рівні, наприклад, контрольних сум. Також такі «сміттєві» дані можуть відігравати негативну роль при стисненні даних.

Для того щоб зменшити втрати пам'яті, пов'язані з вирівнюванням, доцільно групувати поля даних за типами та розташовувати поля більшого розміру на початку структури, а поля меншого в кінці.

Як видно з малюнку А.1, вирівняна структура займає 24 байта в пам'яті, а невирівняна — тільки 17, що дорівнює сумі розмірів полів структури: 1 (char) + 2 * 4 (int) + 8 (double).

ДОДАТОК Б

Індивідуальні завдання підвищеної складності

1. Перетворіть наступний фрагмент програми:

```
int x = 7;
if( a > 5 ) { x = 3; }
else
    if( a < 0 ) { x = 5; }
    else
        { x = 0; }
```

таким чином, щоб умовний оператор було замінено на тернарний.

2. Замініть в наведеному нижче фрагменті оператор циклу while на оператор do … while:

```
while (a + b> 5) {a + = 2; --b; }
```

таким чином, щоб фрагмент програми працював аналогічно, при будь-яких початкових значеннях a і b.

3. Замініть у наведеному нижче фрагменті оператор циклу for на оператор do ... while:

```
for( ; a < b; ++a )
{
    A[a] = A[a + 1];
    if( a + 1 == b - 1 ) break;
}
```

таким чином, щоб фрагмент програми працював аналогічно, при будь-яких початкових значеннях a і b.

4. Напишіть функцію, яка знаходить мінімум і максимум в масиві з N елементів.

5. Визначте шаблонний клас, який зберігає стек фіксованого розміру (в статичному масиві) для довільних елементів, для якого визначені операції додавання і видалення елементів, отримання розміру стека і верхнього елементу.

6. Напишіть програму, яка підраховує кількість унікальних слів що зустрічаються в текстовому файлі. Роздільником слова є символ пробіл.

7. Напишіть програму, яка обчислює числа Фіббоначі до заданого з номером M і зберігає їх в текстовий файл у вигляді пар: номер, значення. Кожне число Фіббоначі утворюється шляхом складання двох попередніх. При цьому відомо, що перші два числа (з номерами 0 і 1) рівні 0 і 1. Програму потрібно реалізувати за допомогою рекурсії і циклу. Номер M ввести з клавіатури.

8. Реалізуйте функцію, яка виводить на екран і в файл всі прості числа менше заданого M. Значення M ввести з клавіатури.

9. Заповнення і виведення одновимірного масиву. Програма повинна заповнити масив, і після введення останнього елемента масиву, виконати відображення елементів масиву на екрані. Заповнювати масив повинен користувач.

10. Додайте відсутні фрагменти програми в місцях зазначених підкресленням:

```
#include <_________>
using namespace std;
___ main ()
{
```

```
    const int N = 10; int A[N]_
    for( int i = _; i <_; ___ ) cin >> _________;
    for( ___ i = 0; ___; ___ ) cout << _________;
    return 0;
}
```

11. Напишіть функцію, яка видалить з масиву елементи що дублюються і поверне нову довжину масиву.

12. Реалізуйте функцію, яка підрахує кількість непарних чисел в масиві цілих чисел і замінить їх на -1.

13. У текстовому файлі, що містить текст програми на мові С++, необхідно перевірити відповідність фігурних дужок { та } що відкриваються і закриваються. Результат перевірки (відповідність) вивести на екран.

14. Написати функцію, яка по дійсному x, обчислює корінь кубічний за такою ітераційною формулою:
$y_{i+1}=0{,}5\left(y_i+3x/\left(2y_i^2+x/y_i\right)\right)$, де $y_0=x$. Ітерування потрібно закінчити якщо $\left|y_{i+1}-y_i\right|<10^{-5}$. Значення x ввести з клавіатури і передати як аргумент, результат вивести на екран.

15. Написати функцію, яка буде шукати в рядку найбільшу кількість розташованих підряд однакових букв. Рядок вводиться з клавіатури і передається як аргумент функції, результат роботи функції потрібно вивести на екран.

Список рекомендованої літератури

1. Stroustrup, Bjarne. The C++ programming language / Bjarne Stroustrup.— Fourth edition. Published by Pearson Education, Inc. 2013. 1360 p. ISBN 978-0-321-56384-2.

2. Грицюк Ю., Рак Т. Програмування мовою С++. Навчальний посібник. / Юрій Грицюк, Тарас Рак. Львів. Вид-во ЛДУ БЖД 2011. 290 с. ISBN 978-966-3466-85-9.

3. Grimes R. Beginning C++ Programming. Richard Grimes. Packt Publishing Ltd. Birmingham, UK. 2017. 499 p. ISBN 978-1-78712-494-3.

4. Татарчук Д.Д., Діденко Ю.В. Програмування мовами С та С++: навч. посіб. / Д.Д. Татарчук, Ю.В. Діденко. – К.: 2012.–112 с.

5. Bancila M. The Modern C++ Challenge. Marius Bancila. Packt Publishing Ltd. Birmingham, UK. 2018. 309 p. ISBN 978-1-78899-386-9.

6. Бублик В.В. Об’єктно-орієнтоване програмування: [Підручник] / В.В. Бублик. – К.: ІТ книга, 2015. – 624 с. ISBN 978-966-97182-1-1.

ЗМІСТ

Навчальне видання

ЗУЄВ Андрій Олександрович
ГАПОН Дмитро Анатолійович
ДЕНИСЕНКО Микола Анатолійович

ОСНОВИ ПРОГРАМУВАННЯ НА МОВІ С++

Методичні вказівки
до виконання самостійних робіт

з курсу «Інформаційні технології та програмування»

для студентів спеціальностей «Автоматизація та комп'ютерно-інтегровані технології», «Телекомунікація та радіотехніка» усіх форм навчання вищих навчальних закладів

Відповідальний за випуск Зуєв А.О.
Роботу до друку рекомендував Дудник О.В.

План 2022 р., Поз.342
Підписано до друку 05.11.2022. формат 60×84 1/16. Папір друк. № 2.
Друк – різографія. гарнітура Times New Roman. Розум. друк. арк.3,2.
Обл. – вид. арк. 2,7.Наклад 100 прим. Зам. № . Ціна договірна.

Самостійне видання