МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання практичних робіт
з дисципліни «Програмування»
для студентів денної та заочної форм навчання
за спеціальністю 123 «Комп'ютерна інженерія»

Частина 2

Затверджено
редакційно-видавничою
радою університету,
протокол №3 від 06.10.2021 р.

Харків
НТУ «ХПІ»
2021

Методичні вказівки до виконання практичних робіт з дисципліни «Програмування» для студентів денної та заочної форм навчання за спеціальністю 123 «Комп'ютерна інженерія». Частина 2 / уклад. : Онищенко В. В., Статкус А. В., Носик А. М., Корольова Я. Ю. – Харків: НТУ «ХПІ», 2021. – 168 с.

Укладачі: В. В. Онищенко
А. В. Статкус
А. М. Носик
Я. Ю. Корольова

Рецензент В. В. Усик

Кафедра мультимедійних інформаційних технологій і систем

ЗМІСТ

Вступ …… 4

Практична робота №1

Робота зі структурами …… 5

Практична робота №2

Робота з бітами …… 31

Практична робота №3

Динамічне виділення пам’яті та багатофайлова компіляція …….. 44

Практична робота №4

Робота зі зв’язними списками ……..... 55

Практична робота №5

Робота зі стеком і чергою ……... 71

Практична робота №6

Реалізація деку на основі масиву …….. 97

Практична робота №7

Робота з бінарними деревами …… 100

Список джерел інформації ……. 124

Додатки…… 125

Додаток 1 Варіанти завдань до практичної роботи №1 …….. 125

Додаток 2 Варіанти завдань до практичної роботи №2 …….. 141

Додаток 3 Приклад програми для роботи зі списком …….. 150

Додаток 4 Варіанти завдань до практичної роботи №7 …….. 155

ВСТУП

Метою вивчення навчальної дисципліни «Програмування» є отримання навичок побудови основних обчислювальних алгоритмів, вивчення основ мови програмування високого рівня *C* та розробки комп'ютерних *C*-програм, ознайомлення з методами розв'язання обчислювальних задач та формування вмінь з їх практичного використання.

Часто вибір правильного представлення даних не обмежується простим вибором типу. Завжди слід подумати про те, які операції доведеться виконувати. Тобто потрібно обрати спосіб зберігання даних і визначити, які операції є припустимими для обраного типу даних. Незважаючи на те, що у мові *C* визначені допустимі операції для її фундаментальних типів, при проєктуванні схеми представлення даних може знадобитися самостійно визначити допустимі операції. Мовою *C* це можна зробити шляхом розробки функцій, які представляють бажані операції. Тобто, проєктування типу даних складається з визначення способу зберігання даних і розробки функцій для керування даними.

Представлені методичні вказівки до виконання практичних робіт призначені для надання допомоги студентам під час опрацювання питань, пов'язаних з роботою зі структурами та їх полями, отримання навичок роботи з окремими бітами, використання побітових операцій та операцій зсуву, організації багатофайлової компіляції. Вони стануть у нагоді при відпрацюванні основних операцій для роботи з такими формами організації даних як: зв'язний список, стек, черга, дек, двійкове дерево пошуку.

Кожну практичну роботу слід оформлювати в робочому зошиті. Для зарахування практичної роботи необхідно показати її оформлення, надати коди *C*-програм та продемонструвати результати їх виконання на комп'ютері.

Практична робота№1
РОБОТА ЗІ СТРУКТУРАМИ

Мета роботи: отримати навички роботи зі структурами та їх полями; навчитися використовувати структури в якості параметрів функцій та в якості результату, що повертається; відпрацювати навички з наповнення структури інформацією з текстового файлу, її обробки, перетворення та подальшого зберігання у файлі.

Теоретична частина

Розглянемо приклади виконання завдання для роботи зі структурами [1].

Приклад №1. В директорії з текстом вашої *C*-програми створіть текстовий файл **`input.txt`**, що містить наступну інформацію про працівників:

```
================================================================================
Name               Position                    Office          Age   Start date
================================================================================
Gloria Little      Systems Administrator       New York        59    10.04.2019
Haley Kennedy      Senior Marketing Designer   London          43    18.12.2012
Hermione Butler    Regional Director           London          47    21.03.2011
Herrod Chandler    Sales Assistant             San Francisco   59    06.08.2012
Hope Fuentes       Secretary                   San Francisco   41    12.02.2010
Howard Hatfield    Office Manager              San Francisco   51    16.12.2018
Jackson Bradshaw   Director                    New York        65    26.09.2008
Jena Gaines        Office Manager              London          30    19.12.2018
Jenette Caldwell   Development Lead            New York        30    03.09.2011
Jennifer Acosta    Junior Javascript Developer Edinburgh       43    01.02.2013
================================================================================
```

Примітка: Для визначення місця розміщення файлу **`input.txt`**, необхідно після створення нового проєкту натиснути комбінацію клавіш `Ctrl+F9`. Перш ніж заносити до файлу інформацію необхідно в Microsoft Word переконатися, що вона не містить нічого крім необхідних символів і пробілів. Наприкінці кожного рядка повинен міститися символ переходу на новий рядок.

Перелік завдань, які необхідно вирішити:

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **`input.txt`**, та створіть масив структур визначеного типу. Зверніть увагу на те, що поле `Start date` являє собою вкладену структуру.

2) Перенесіть інформацію з файлу **input.txt** до масиву структур і виведіть її на екран за допомогою окремої функції. Зверніть увагу на те, що під час виводу першим слід додати стовпець з номерами рядків (нумерація починається з 1), а стовпець `Start date` необхідно розмістити після стовпця `Name`.

Текст програми буде мати такий вигляд:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#define N     20          // Ім'я та прізвище працівника
#define N1    29          // Посада
#define N2    16          // Місцезнаходження офісу
#define KOL   10          // Кількість працівників
#define DATE  11          // Кількість символів для зберігання дати
struct Data               // Дата початку роботи
{
    int    day;           // День
    int    month;         // Місяць
    int    year;          // Рік
};
struct Company
{
    char   name[N];       // Ім'я та прізвище працівника
    char   position[N1]; // Посада
    char   office[N2];    // Місцезнаходження офісу
    int    age;           // Рік працівника
    struct Data date;     // Дата початку роботи
} com[KOL];
int main(void)
{
    FILE   *f1;
    char   mas[DATE];     // Дата початку роботи
    char   mas1[83];      // Для непотрібної символьної інформації
    char   day[3];        // Число у символьному вигляді
    char   month[3];      // Місяць у символьному вигляді
    char   year[5];       // Рік у символьному вигляді
    SetConsoleOutputCP(1251);
    // Відкриваємо файл для читання
    if((f1 = fopen("input.txt", "r")) == NULL)
    {
        fprintf(stderr, "Неможливо відкрити файл input.txt\n");
        exit(EXIT_FAILURE);
    }
  else
  {
    // Зчитування трьох непотрібних рядків
    for(int i = 0; i < 3; i++)
```

```
  fgets(mas1, 83, f1);
for(int i = 0; i < KOL; i++)
{
  int j;
  fgets(com[i].name, N, f1);
  fgets(mas1, 2, f1);   // Зчитуємо непотрібні символи
  // Знищуємо непотрібні пробіли (крапки показують межі рядка)
  // Наприклад, було: .Gloria Little      .
  j = N-2;                          // Стаємо наприкінці рядка
  while(com[i].name[j] == ' ') j--; // Знаходимо кінець слова
  com[i].name[j+1] = '\x0';         // Фіксуємо кінець рядка
  //           стало: .Gloria Little.
  fgets(com[i].position, N1, f1);
  fgets(mas1, 2, f1);   // Зчитуємо непотрібні символи
  j = N1-2;
  while(com[i].position[j] == ' ') j--;
  com[i].position[j+1] = '\x0';
  fgets(com[i].office, N2, f1);
  fgets(mas1, 2, f1);   // Зчитуємо непотрібні символи
  j = N2-2;
  while(com[i].office[j] == ' ') j--;
  com[i].office[j+1] = '\x0';
  fscanf(f1, "%d", &com[i].age);
  fscanf(f1, "%s", mas);
  // Підготовка інформації про день, місяць і рік
  // у символьному вигляді
  day[0]   = mas[0]; day[1]   = mas[1]; day[2]   = '\0';
  month[0] = mas[3]; month[1] = mas[4]; month[2] = '\0';
  year[0]  = mas[6]; year[1]  = mas[7];
  year[2]  = mas[8]; year[3]  = mas[9];
  year[4]  = '\0';
  // Перетворюємо символьну інформацію про день, місяць і рік
  // на числову форму за допомогою функції atoi()
  com[i].date.day   = atoi(day);
  com[i].date.month = atoi(month);
  com[i].date.year  = atoi(year);
  fgets(mas1, 2, f1);  // Зчитуємо непотрібні символи
}
printf("\n                                    Інформація про "
       "працівників       \n");
printf("============================================"
       "============================================\n");
printf(" N Name                 Start date   Position "
       "                    Office          Age\n");
printf("============================================"
       "============================================\n");
for(int i = 0; i < KOL; i++)
{
  printf("%2d ", i+1);
  printf("%-20s", com[i].name);
  printf("%02d.%02d.%4d   ", com[i].date.day,
         com[i].date.month, com[i].date.year);
```

```
      printf("%-29s", com[i].position);
      printf("%-15s", com[i].office);
      printf("%3d    \n", com[i].age);
    }
    printf("========================================="
           "=========================================\n");
  }
  // Перевіряємо успішність закриття файлу
  if(fclose(f1) != 0)
    fprintf(stderr, "Помилка під час закриття файлу\n");
  return 0;
}
```

Результат виконання програми наведено на рис. 1.1.

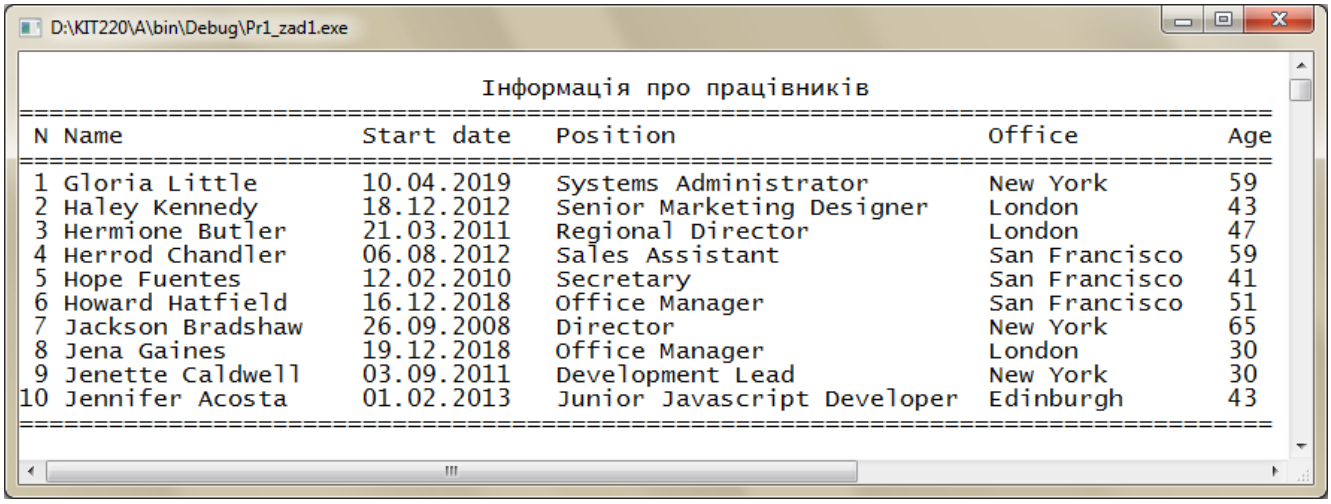


Рисунок 1.1 – Результат виводу масиву структур

Приклад №2. Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього наступну інформацію у вигляді таблиці:

```
================================================================================
Name               Position                    Office          Age   Start date
================================================================================
Gloria Little      Systems Administrator       New York        59    10.04.2019
Haley Kennedy      Senior Marketing Designer   London          43    18.12.2012
Hermione Butler    Regional Director           London          47    21.03.2011
Herrod Chandler    Sales Assistant             San Francisco   59    06.08.2012
Hope Fuentes       Secretary                   San Francisco   41    12.02.2010
Howard Hatfield    Office Manager              San Francisco   51    16.12.2018
Jackson Bradshaw   Director                    New York        65    26.09.2008
Jena Gaines        Office Manager              London          30    19.12.2018
Jenette Caldwell   Development Lead            New York        30    03.09.2011
Jennifer Acosta    Junior Javascript Developer Edinburgh       43    01.02.2013
================================================================================
```

Перелік завдань, які необхідно вирішити:

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **input.txt**, та створіть масив структур визначеного типу. Зверніть увагу на те, що поле `Start date` являє собою вкладену структуру.

2) Перенесіть інформацію з файлу **input.txt** до масиву структур і виведіть її на екран за допомогою окремої функції. Зверніть увагу на те, що під час виводу першим слід додати стовпець з номерами рядків (нумерація починається з 1), а стовпець `Start date` необхідно розмістити після стовпця `Position`.

3) Напишіть функцію, яка б передавала до функції `main()` в якості результату та виводила на екран сумарний вік осіб, що знаходяться в `San Francisco`.

4) Напишіть функцію, яка б видаляла інформацію про будь-якого працівника. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Результат видалення рядка виведіть на екран.

5) Напишіть функцію, яка б додавала інформацію про нового працівника в заданому рядку. Номер рядка та інформацію про працівника слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за збільшенням дати початку роботи в компанії (`Start date`). Результат сортування занесіть до файлу **output.txt**, а також виведіть його на екран.

Перший варіант програми для реалізації завдання з прикладу №2:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#define N     20        // Ім'я та прізвище працівника
#define N1    29        // Посада
#define N2    16        // Місцезнаходження офісу
#define KOL   10        // Кількість працівників
#define DATE  11        // Кількість символів для зберігання дати
struct Data             // Дата початку роботи
{
    int    day;         // День
```

```
    int    month;          // Місяць
    int    year;           // Рік
};

struct Company
{
    char   name[N];        // Ім'я та прізвище працівника
    char   position[N1];   // Посада
    char   office[N2];     // Місцезнаходження офісу
    int    age;            // Рік працівника
    struct Data date;      // Дата початку роботи
} com[KOL];

int sum_vik(void);

int main(void)
{
    FILE   *f1;

    char   mas[DATE]; // Дата початку роботи у символьному вигляді
    char   mas1[80];  // Для непотрібної символьної інформації
    char   day[3];    // Число у символьному вигляді
    char   month[3];  // Місяць у символьному вигляді
    char   year[5];   // Рік у символьному вигляді
    int    d;         // Номер рядка, який треба видалити/додати

    SetConsoleOutputCP(1251);

    if((f1 = fopen("input.txt", "r")) == NULL)
    {
        fprintf(stderr, "Неможливо відкрити файл input.txt\n");
        exit(EXIT_FAILURE);
    }
    else
    {
        // Зчитування трьох непотрібних рядків
        for(int i = 0; i < 3; i++)
            fgets(mas1, 83, f1);

        for(int i = 0; i < KOL; i++)
        {
            int j;
            fgets(com[i].name, N, f1);
            j = N-2;
            while(com[i].name[j] == ' ') j--;
            com[i].name[j+1] = '\x0';
            fgets(mas1, 2, f1);   // Зчитуємо непотрібні символи

            fgets(com[i].position, N1, f1);
            j = N1-2;
            while(com[i].position[j] == ' ') j--;
            com[i].position[j+1] = '\x0';
            fgets(mas1, 2, f1);   // Зчитуємо непотрібні символи

            fgets(com[i].office, N2, f1);
```

```
          j = N2-2;
      while(com[i].office[j] == ' ') j--;
      com[i].office[j+1] = '\x0';
      fgets(mas1, 2, f1);   // Зчитуємо непотрібні символи
      fscanf(f1, "%d", &com[i].age);
      fscanf(f1, "%s", mas);

      // Підготовка інформації про день, місяць і рік
      // у символьному вигляді
      day[0]   = mas[0]; day[1]   = mas[1];
      day[2]   = '\0';
      month[0] = mas[3]; month[1] = mas[4];
      month[2] = '\0';
      year[0]  = mas[6]; year[1]  = mas[7];
      year[2]  = mas[8]; year[3]  = mas[9];
      year[4]  = '\0';

      // Перетворюємо символьну інформацію про день, місяць і рік
      // на числову форму за допомогою функції atoi()
      com[i].date.day   = atoi(day);
      com[i].date.month = atoi(month);
      com[i].date.year  = atoi(year);
      fgets(mas1, 2, f1);  // Зчитуємо непотрібні символи
    }
    printf("\n            Інформація про працівників         \n");
    printf("================================================\n");
    printf(" N Name    Start date   Position   Office     Age\n");
    printf("================================================\n");
    for(int i = 0; i < KOL; i++)
    {
      printf("%2d ", i+1);
      printf("%-20s", com[i].name);
      printf("%02d.%02d.%4d   ", com[i].date.day,
             com[i].date.month, com[i].date.year);
      printf("%-29s", com[i].position);
      printf("%-15s", com[i].office);
      printf("%3d    \n", com[i].age);
    }
    printf("================================================\n");

    int sum;
    sum = sum_vik();      // Визначення сумарного віку осіб,
                          // що знаходяться в San Francisco
    printf("\nСумарний вік осіб, що знаходяться в San "
           "Francisco дорівнює %d.\n", sum);
}
if(fclose(f1) != 0)
  fprintf(stderr, "Помилка під час закриття файлу\n");

printf("\nВведіть номер рядка, який треба видалити: ");
scanf("%d", &d);
for(int i = d-1; i < KOL; i++)
{
```

```
  for(int k = 0; k < N; k++)
    com[i].name[k]     = com[i+1].name[k];
  for(int k = 0; k < N1; k++)
    com[i].position[k] = com[i+1].position[k];
  for(int k = 0; k < N2; k++)
    com[i].office[k]   = com[i+1].office[k];

  com[i].age        = com[i+1].age;
  com[i].date.day   = com[i+1].date.day;
  com[i].date.month = com[i+1].date.month;
  com[i].date.year  = com[i+1].date.year;
}

printf("\n\n   Інформація про працівників після видалення  \n");
printf("=================================================\n");
printf(" N Name      Start date   Position   Office     Age\n");
printf("=================================================\n");
for(int i = 0; i < KOL-1; i++)
{
  printf("%2d ", i+1);
  printf("%-20s", com[i].name);
  printf("%02d.%02d.%4d   ", com[i].date.day,
         com[i].date.month, com[i].date.year);
  printf("%-29s", com[i].position);
  printf("%-15s", com[i].office);
  printf("%3d\n", com[i].age);
}
printf("=================================================\n");

int    day1;
int    month1;
int    year1;
char   name1[N];
char   position1[N1];
char   office1[N2];
int    age1;

printf("\nВведіть номер рядка, який треба додати: ");
scanf("%d", &d);
printf("\nВведіть інформацію про працівника \n");
printf("================================");

fflush(stdin);
printf("\nName        ");
fgets(name1, N, stdin);
for(int j = 0; j < N; j++)
  if(name1[j] == '\n') name1[j] = '\0';

printf("Position    ");
fgets(position1, N1, stdin);
for(int j = 0; j < N1; j++)
  if(position1[j] == '\n') position1[j] = '\0';

printf("Office      ");
```

```
fgets(office1, N2, stdin);
for(int j = 0; j < N2; j++)
  if(office1[j] == '\n') office1[j] = '\0';

printf("Age         ");
scanf("%d", &age1);
printf("Day         ");
scanf("%d", &day1);
printf("Month       ");
scanf("%d", &month1);
printf("Year        ");
scanf("%d", &year1);
printf("================================\n");

for(int i = KOL-1 ; i >= d; i--)
{
  for(int k = 0; k < N; k++)
    com[i].name[k]     = com[i-1].name[k];
  for(int k = 0; k < N1; k++)
    com[i].position[k] = com[i-1].position[k];
  for(int k = 0; k < N2; k++)
    com[i].office[k]   = com[i-1].office[k];

  com[i].age        = com[i-1].age;
  com[i].date.day   = com[i-1].date.day;
  com[i].date.month = com[i-1].date.month;
  com[i].date.year  = com[i-1].date.year;
}

for(int k = 0; k < N; k++)
  com[d-1].name[k]     = name1[k];
for(int k = 0; k < N1; k++)
  com[d-1].position[k] = position1[k];
for(int k = 0; k < N2; k++)
  com[d-1].office[k]   = office1[k];

com[d-1].age        = age1;
com[d-1].date.day   = day1;
com[d-1].date.month = month1;
com[d-1].date.year  = year1;
printf("\n\n   Інформація про працівників після додавання  \n");
printf("================================================\n");
printf(" N Name      Start date   Position   Office     Age\n");
printf("================================================\n");
for(int i = 0; i < KOL; i++)
{
  printf("%2d ", i+1);
  printf("%-20s", com[i].name);
  printf("%02d.%02d.%4d   ", com[i].date.day,
         com[i].date.month, com[i].date.year);
  printf("%-29s", com[i].position);
  printf("%-15s", com[i].office);
  printf("%3d\n", com[i].age);
}
```

```
printf("=================================================\n");
//=============================================================
//  Сортування за збільшенням року початку роботи
//=============================================================
// rab — для переставлення елементів структури
struct Company rab;
int k;

for(int i = KOL-1; i > 0; i--)
{
  for(int j = 0; j < i; j++)
  {
    if(com[j].date.year > com[j+1].date.year)
    {
      for(k = 0; k < N; k++)
        rab.name[k] = com[j].name[k];
      rab.age          = com[j].age;
      rab.date.day     = com[j].date.day;
      rab.date.month   = com[j].date.month;
      rab.date.year    = com[j].date.year;
      for(k = 0; k < N1; k++)
        rab.position[k] = com[j].position[k];
      for(k = 0; k < N2; k++)
        rab.office[k] = com[j].office[k];

      for(k = 0; k < N; k++)
        com[j].name[k] = com[j+1].name[k];
      com[j].age        = com[j+1].age;
      com[j].date.day   = com[j+1].date.day;
      com[j].date.month = com[j+1].date.month;
      com[j].date.year  = com[j+1].date.year;
      for(k = 0; k < N1; k++)
        com[j].position[k] = com[j+1].position[k];
      for(k = 0; k < N2; k++)
        com[j].office[k] = com[j+1].office[k];
      for(k = 0; k < N; k++)
        com[j + 1].name[k] = rab.name[k];
      com[j+1].age        = rab.age;
      com[j+1].date.day   = rab.date.day;
      com[j+1].date.month = rab.date.month;
      com[j+1].date.year  = rab.date.year;
      for(k = 0; k < N1; k++)
        com[j+1].position[k] = rab.position[k];
      for(k = 0; k < N2; k++)
        com[j+1].office[k] = rab.office[k];
    }
  }
}
printf("\n\n Відсортована інформація за збільшенням року, "
       "з якого почалася робота      \n");
printf("=================================================\n");
printf(" N Name      Start date   Position   Office     Age\n");
printf("=================================================\n");
```

```
  for(int i = 0; i < KOL; i++)
  {
    printf("%2d ", i+1);
    printf("%-20s", com[i].name);
    printf("%02d.%02d.%4d   ", com[i].date.day,
           com[i].date.month, com[i].date.year);
    printf("%-29s", com[i].position);
    printf("%-15s", com[i].office);
    printf("%3d\n", com[i].age);
  }
  printf("================================================\n");

  if((f1 = fopen("output.txt", "w")) == NULL)
  {
    fprintf(stderr, "Неможливо відкрити файл input.txt\n");
    exit(EXIT_FAILURE);
  }
  else
  {
    fprintf(f1, "  Відсортована інформація за збільшенням року, "
                "з якого почалася робота        \n");
    printf("================================================\n");

    for(int i = 0; i < KOL; i++)
    {
      fprintf(f1, "%2d ", i+1);
      fprintf(f1, "%-20s", com[i].name);
      fprintf(f1, "%02d.%02d.%4d   ", com[i].date.day,
              com[i].date.month, com[i].date.year);
      fprintf(f1, "%-29s", com[i].position);
      fprintf(f1, "%-15s", com[i].office);
      fprintf(f1, "%3d\n", com[i].age);
    }
    fprintf(f1, "================================================\n");
  }
  if(fclose(f1) != 0)
    fprintf(stderr, "Помилка під час закриття файлу\n");
  return 0;
}

int sum_vik()
{
  int sum = 0;

  for(int i = 0; i < KOL; i++)
  {
    if(strcmp(com[i].office, "San Francisco") == 0)
      sum += com[i].age;
  }
  return sum;
}
```

Фрагмент результатів роботи програми з обчислення сумарного віку осіб, що знаходяться в `San Francisco`, та видалення зазначеного рядка, номер якого вводився з клавіатури, наведено на рис. 1.2.

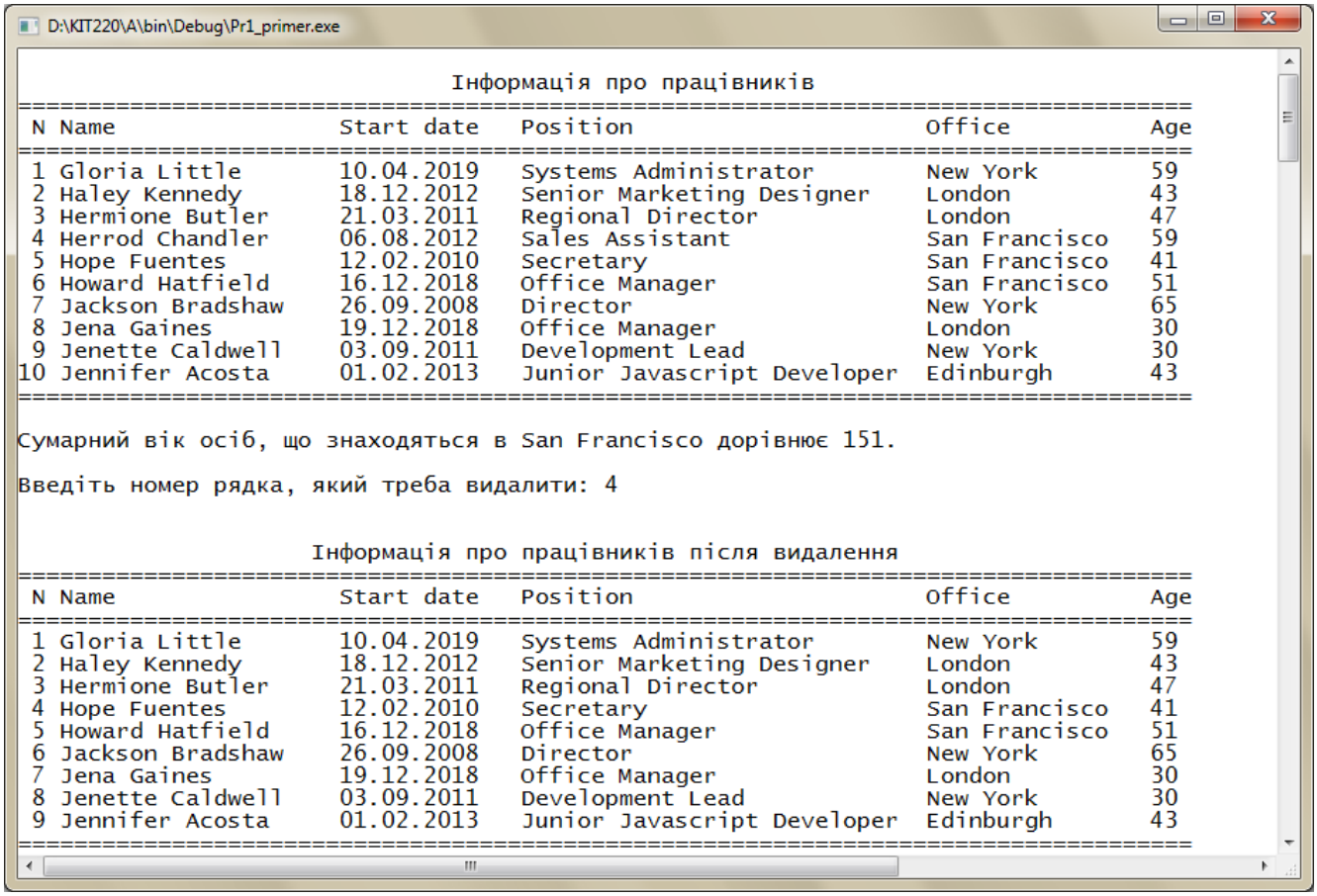


Рисунок 1.2 – Результати роботи програми (перша частина)

Фрагмент результатів роботи програми з вводу інформації про нового працівника, та додавання її в зазначеному рядку, номер якого вводився з клавіатури, наведено на рис. 1.3.

Остання таблиця на рис. 1.3 є результатом сортування інформації за збільшенням років, з яких розпочали роботу працівники в своїх компаніях.

Слід зазначити, що ця таблиця також дублюється в файлі **`output.txt`** (рис. 1.4).

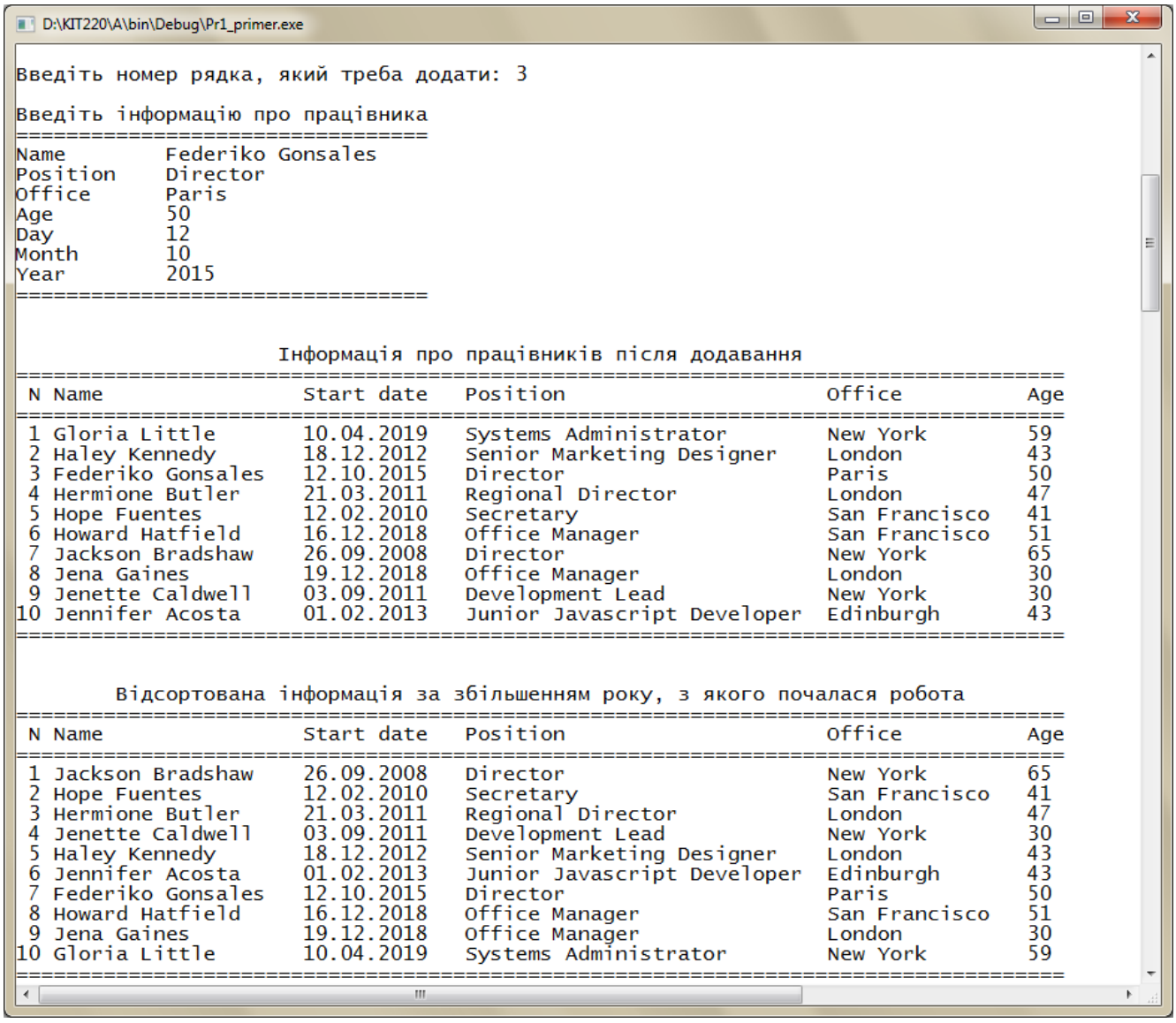


Рисунок 1.3 – Результати роботи програми (друга частина)

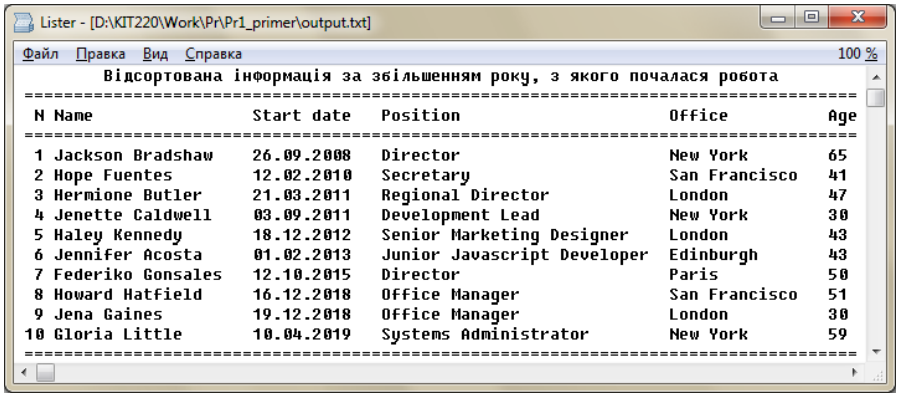


Рисунок 1.4 – Результати роботи програми (файл **`output.txt`**)

Другий варіант програми для реалізації завдання з прикладу №2:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define N     20          // Ім'я та прізвище працівника
#define N1    29          // Посада
#define N2    16          // Місцезнаходження офісу
#define KOL   10          // Кількість працівників
#define DATE  11          // Кількість символів для зберігання дати

struct Data               // Дата початку роботи
{
    int    day;           // День
    int    month;         // Місяць
    int    year;          // Рік
};

struct Company
{
    char   name[N];       // Ім'я та прізвище працівника
    char   position[N1];  // Посада
    char   office[N2];    // Місцезнаходження офісу
    int    age;           // Рік працівника
    struct Data date;     // Дата початку роботи
} com[KOL];

int sum_vik(void);
void print_struct(int line, FILE *f);

int main(void)
{
    FILE   *f1;
    char   mas[DATE];     // Дата початку роботи
    char   mas1[83];      // Для непотрібної символьної інформації
    char   day[3];        // Число у символьному вигляді
    char   month[3];      // Місяць у символьному вигляді
    char   year[5];       // Рік у символьному вигляді
    int    nomer;         // Номер рядка, що додається/видаляється

    SetConsoleOutputCP(1251);
    time_t rawtime;
    struct tm *timeinfo;
    // Визначення поточного часу
    time(&rawtime);
    timeinfo = localtime(&rawtime);
    printf("%s\n", asctime(timeinfo));
    if((f1 = fopen("input.txt", "r")) == NULL)
    {
        fprintf(stderr, "Неможливо відкрити файл input.txt\n");
    exit(EXIT_FAILURE);
  }
```

```
else
{
  // Зчитування трьох непотрібних рядків
  for(int i = 0; i < 3; i++)
    fgets(mas1, 83, f1);
  for(int i = 0; i < KOL; i++)
  {
    int j;
    fgets(com[i].name, N, f1);
    j = N-2;
    while(com[i].name[j] == ' ') j--;
    com[i].name[j+1] = '\x0';
    fgets(mas1, 2, f1);   // Зчитуємо непотрібні символи
    fgets(com[i].position, N1, f1);
    j = N1-2;
    while(com[i].position[j] == ' ') j--;
    com[i].position[j+1] = '\x0';
    fgets(mas1, 2, f1);   // Зчитуємо непотрібні символи
    fgets(com[i].office, N2, f1);
    j = N2-2;
    while(com[i].office[j] == ' ') j--;
    com[i].office[j+1] = '\x0';
    fgets(mas1, 2, f1);   // Зчитуємо непотрібні символи
    fscanf(f1, "%d", &com[i].age);
    fscanf(f1, "%s", mas);
    // Підготовка інформації про день, місяць і рік
    // у символьному вигляді
    day[0]   = mas[0]; day[1]   = mas[1]; day[2]   = '\0';
    month[0] = mas[3]; month[1] = mas[4]; month[2] = '\0';
    year[0]  = mas[6]; year[1]  = mas[7];
    year[2]  = mas[8]; year[3]  = mas[9];
    year[4]  = '\0';
    // Перетворюємо символьну інформацію про день, місяць і рік
    // на числову форму за допомогою функції atoi()
    com[i].date.day   = atoi(day);
    com[i].date.month = atoi(month);
    com[i].date.year  = atoi(year);
    fgets(mas1, 2, f1);  // Зчитуємо непотрібні символи
  }
  printf("\n            Інформація про працівників         \n");
  print_struct(KOL, stdout);
  int sum;
  sum = sum_vik();  //  Визначення сумарного віку осіб,
                    //  що знаходяться в San Francisco
  printf("\nСумарний вік осіб, що знаходяться в San "
         "Francisco дорівнює %d.\n", sum);
}
if(fclose(f1) != 0)
  fprintf(stderr, "Помилка під час закриття файлу\n");
printf("\nВведіть номер рядка, який треба видалити: ");
scanf("%d", &nomer);
for(int i = nomer-1; i < KOL; i++)
```

```
  com[i] = com[i+1];
printf("\n\n   Інформація про працівників після видалення  \n");
print_struct(KOL-1, stdout);
struct Company inf, *pstr;
pstr = &com[0];
printf("\nВведіть номер рядка, в який треба додати "
       "інформацію: ");
scanf("%d", &nomer);
printf("\nВведіть інформацію про працівника \n");
printf("================================");
fflush(stdin);
printf("\nName        ");
fgets(inf.name, N, stdin);
for(int j = 0; j < N; j++)
  if(inf.name[j] == '\n') inf.name[j] = '\0';
printf("Position    ");
fgets(inf.position, N1, stdin);

for(int j = 0; j < N1; j++)
  if(inf.position[j] == '\n') inf.position[j] = '\0';
printf("Office      ");
fgets(inf.office, N2, stdin);

for(int j = 0; j < N2; j++)
  if(inf.office[j] == '\n') inf.office[j] = '\0';
printf("Age         ");
scanf("%d", &inf.age);
printf("Day         ");
scanf("%d", &inf.date.day);
printf("Month       ");
scanf("%d", &inf.date.month);
printf("Year        ");
scanf("%d", &inf.date.year);
printf("================================\n");

for(int i = KOL-1 ; i >= nomer; i--)
  pstr[i] = pstr[i-1];
pstr[nomer-1] = inf;
printf("\n\n  Інформація про працівників після додавання   \n");
print_struct(KOL, stdout);

//============================================================
//  Сортування за збільшенням дати початку роботи
//============================================================
// rab – для переставлення елементів структури
struct Company rab;
for(int i = KOL-1; i > 0; i--)
{
  for(int j = 0; j < i; j++)
  {
    if(com[j].date.year >= com[j+1].date.year)
    {
```

```
        if(com[j].date.year == com[j+1].date.year)
        {
          if(com[j].date.month >= com[j+1].date.month)
          {
            if(com[j].date.month == com[j+1].date.month)
            {
              if(com[j].date.day > com[j+1].date.day)
              {
                rab      = com[j];
                com[j]   = com[j+1];
                com[j+1] = rab;
              }
            }
            else
            {
              rab      = com[j];
              com[j]   = com[j+1];
              com[j+1] = rab;
            }
          }
        }
        else
        {
          rab      = com[j];
          com[j]   = com[j+1];
          com[j+1] = rab;
        }
      }
    }
  }
  printf("\n\n  Відсортована інформація за збільшенням дати, "
         "з якої почалася робота      \n");
  print_struct(KOL, stdout);
  if((f1 = fopen("output.txt", "a")) == NULL)
  {
    fprintf(stderr, "Неможливо відкрити файл input.txt\n");
    exit(EXIT_FAILURE);
  }
  else
  {
    time(&rawtime);
    timeinfo = localtime(&rawtime);
    fprintf(f1,"\n%s\n", asctime(timeinfo));
    fprintf(f1,"\n\n Відсортована інформація за збільшенням дати,"
               " з якої почалася робота        \n");
    print_struct(KOL, f1);
  }
  if(fclose(f1) != 0)
    fprintf(stderr, "Помилка під час закриття файлу\n");
  return 0;
}
```

```
int sum_vik()
{
  int sum = 0;
  for(int i = 0; i < KOL; i++)
  {
    if(strcmp(com[i].office, "San Francisco") == 0)
      sum += com[i].age;
  }
  return sum;
}

void print_struct(int line, FILE *f)
{
  fprintf(f, "================================================\n");
  fprintf(f, " N Name    Start date   Position    Office  Age\n");
  fprintf(f, "================================================\n");
  for(int i = 0; i < line; i++)
  {
    fprintf(f, "%2d ", i+1);
    fprintf(f, "%-20s", com[i].name);
    fprintf(f, "%02d.%02d.%4d   ", com[i].date.day,
            com[i].date.month, com[i].date.year);
    fprintf(f, "%-29s", com[i].position);
    fprintf(f, "%-15s", com[i].office);
    fprintf(f, "%3d\n", com[i].age);
  }
  fprintf(f, "================================================\n");
}
```

Фрагмент результатів роботи програми з обчислення сумарного віку осіб, що знаходяться в `San Francisco`, та видалення зазначеного рядка, номер якого вводився з клавіатури, наведено на рис. 1.5. Зверніть увагу, що на початку виводу інформації позначається поточний час запуску програми.

Фрагмент результатів роботи програми з введення інформації про нового працівника, та додавання її в зазначеному рядку, номер якого вводився з клавіатури, наведено на рис. 1.6.

Остання таблиця на рис. 1.6 є результатом сортування інформації за збільшенням дат, з яких розпочали роботу працівники в своїх компаніях. Ця інформація дублюється в файлі **`output.txt`** (рис. 1.7). На початку файлу також позначається поточний час запуску програми. Якщо програму запустити ще, то інформація буде додаватися наприкінці файлу.

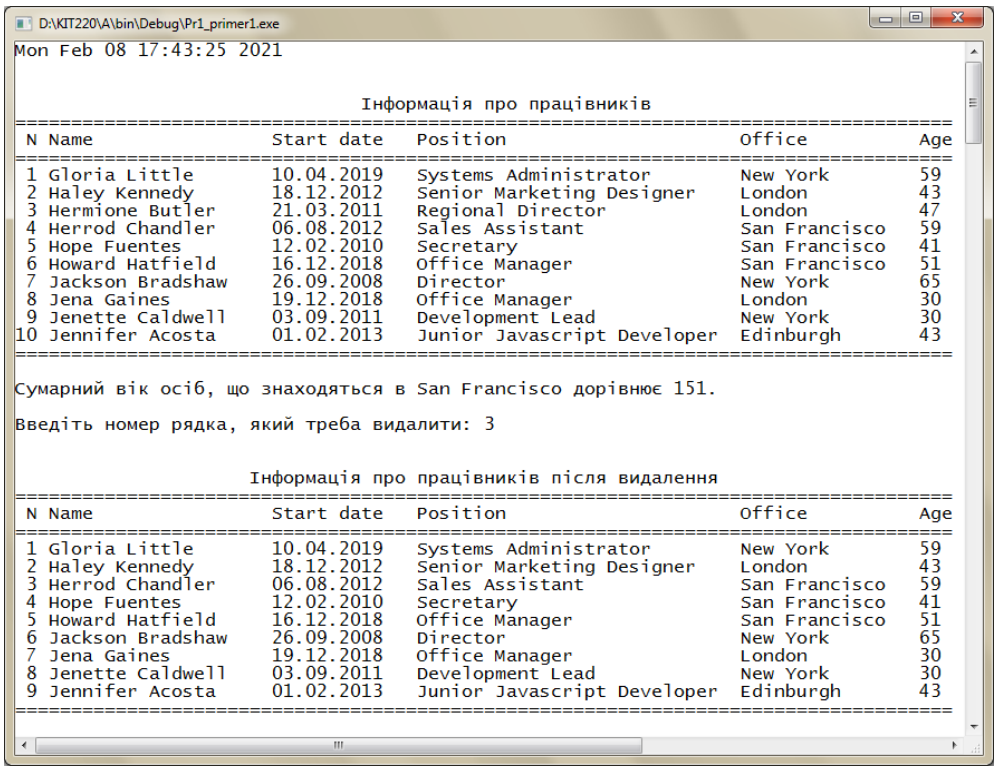


Рисунок 1.5 – Результати роботи програми (перша частина)

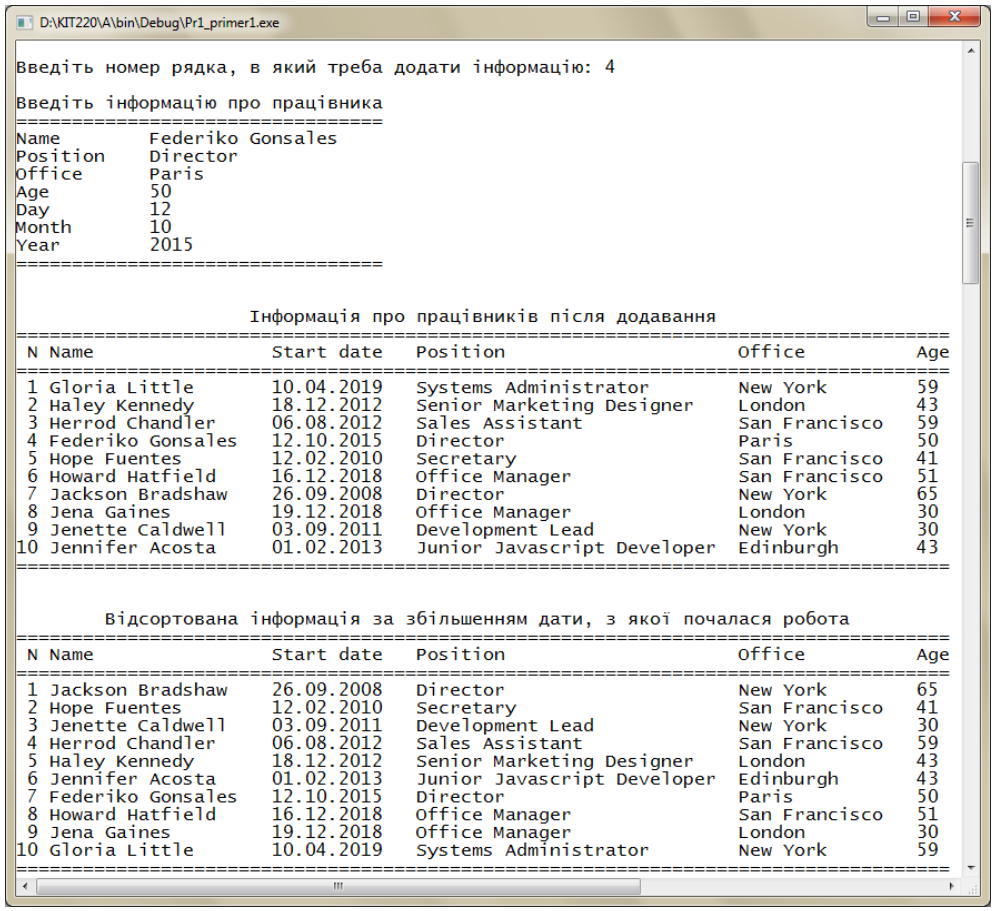


Рисунок 1.6 – Результати роботи програми (друга частина)

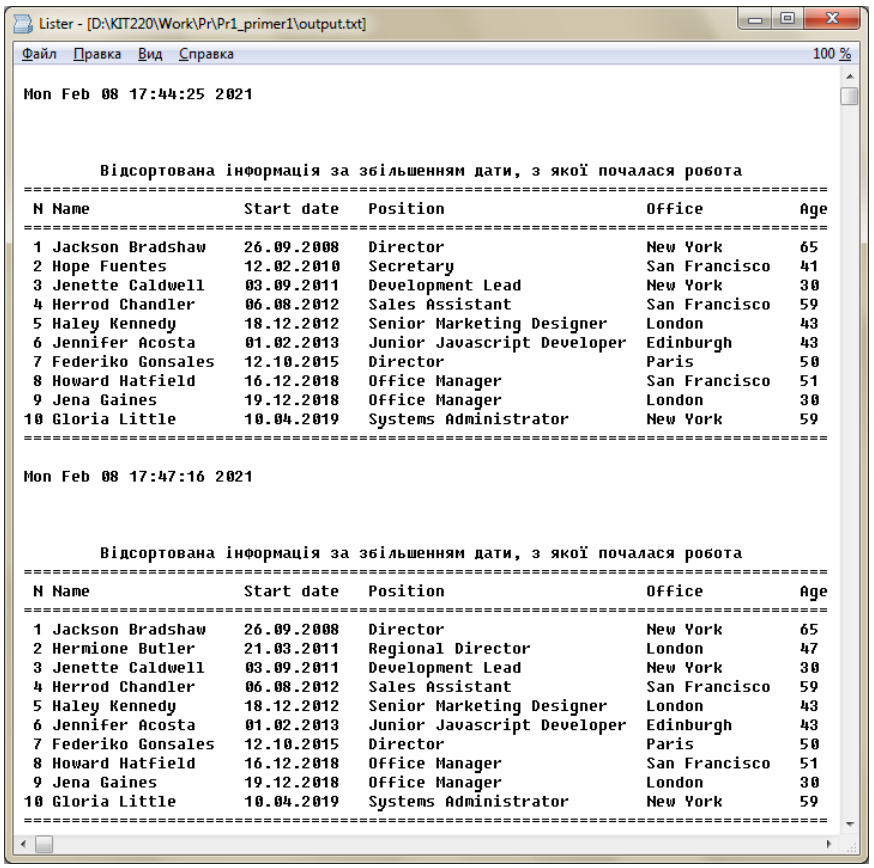


Рисунок 1.7 – Результати роботи програми (файл **output.txt**)

Третій варіант програми для реалізації завдання з прикладу №2:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define N     20          // Ім'я та прізвище працівника
#define N1    29          // Посада
#define N2    16          // Місцезнаходження офісу
#define KOL   10          // Кількість працівників
#define DATE  11          // Кількість символів для зберігання дати

struct Data               // Дата початку роботи
{
    int    day;           // День
    int    month;         // Місяць
    int    year;          // Рік
};

struct Company
{
    char   name[N];       // Ім'я та прізвище працівника
    char   position[N1]; // Посада
```

```
    char   office[N2];   // Місцезнаходження офісу
    int    age;          // Рік працівника
    struct Data date;    // Дата початку роботи
};

int sum_vik(struct Company *);
void print_struct(struct Company *str, int line, FILE *f);
void swap(struct Company *str, int j);

int main(void)
{
    FILE  *f1;
    char  mas[DATE];     // Дата початку роботи
    char  mas1[83];      // Для непотрібної символьної інформації
    char  day[3];        // Число у символьному вигляді
    char  month[3];      // Місяць у символьному вигляді
    char  year[5];       // Рік у символьному вигляді
    int   nomer;         // Номер рядка, що додається/видаляється

    struct Company com[KOL];

    SetConsoleOutputCP(1251);

    time_t rawtime;
    struct tm *timeinfo;

    time(&rawtime);
    timeinfo = localtime(&rawtime);
    printf("%s\n", asctime(timeinfo));

    if((f1 = fopen("input.txt", "r")) == NULL)
    {
        fprintf(stderr, "Неможливо відкрити файл input.txt\n");
        exit(EXIT_FAILURE);
    }
    else
    {
        // Зчитування трьох непотрібних рядків
        for(int i = 0; i < 3; i++)
            fgets(mas1, 83, f1);

        for(int i = 0; i < KOL; i++)
        {
            int j;
            fgets(com[i].name, N, f1);
            j = N-2;
            while(com[i].name[j] == ' ') j--;
            com[i].name[j+1] = '\x0';
            fgets(mas1, 2, f1);   // Зчитуємо непотрібні символи

            fgets(com[i].position, N1, f1);
            j = N1-2;
            while(com[i].position[j] == ' ') j--;
            com[i].position[j+1] = '\x0';
            fgets(mas1, 2, f1);   // Зчитуємо непотрібні символи
```

```
    fgets(com[i].office, N2, f1);
    j = N2-2;
    while(com[i].office[j] == ' ') j--;
    com[i].office[j+1] = '\x0';
    fgets(mas1, 2, f1);   // Зчитуємо непотрібні символи
    fscanf(f1, "%d", &com[i].age);
    fscanf(f1, "%s", mas);

    // Підготовка інформації про день, місяць і рік
    // у символьному вигляді
    day[0]   = mas[0]; day[1]   = mas[1]; day[2]   = '\0';
    month[0] = mas[3]; month[1] = mas[4]; month[2] = '\0';
    year[0]  = mas[6]; year[1]  = mas[7];
    year[2]  = mas[8]; year[3]  = mas[9]; year[4]  = '\0';

    // Перетворюємо символьну інформацію про день, місяць і рік
    // на числову форму за допомогою функції atoi()
    com[i].date.day   = atoi(day);
    com[i].date.month = atoi(month);
    com[i].date.year  = atoi(year);
    fgets(mas1, 2, f1);   // Зчитуємо непотрібні символи
  }
  printf("\n           Інформація про працівників          \n");
  print_struct(com, KOL, stdout);

  int sum;
  sum = sum_vik(com);     // Викликаємо функцію визначення
                          // сумарного віку осіб,
                          // що знаходяться в Лондоні
  printf("\nСумарний вік осіб, що знаходяться в Лондоні "
         "дорівнює %d.\n", sum);
}
if(fclose(f1) != 0)
  fprintf(stderr, "Помилка під час закриття файлу\n");

printf("\nВведіть номер рядка, який треба видалити: ");
scanf("%d", &nomer);
for(int i = nomer-1; i < KOL; i++)
  com[i] = com[i+1];

printf("\n\n  Інформація про працівників після видалення   \n");
print_struct(com, KOL-1, stdout);

struct Company inf, *pstr;
pstr = &com[0];

printf("\nВведіть номер рядка, в який треба додати "
       "інформацію: ");
scanf("%d", &nomer);
printf("\nВведіть інформацію про працівника \n");
printf("================================");

fflush(stdin);
printf("\nName        ");
fgets(inf.name, N, stdin);
```

```
for(int j = 0; j < N; j++)
  if(inf.name[j] == '\n') inf.name[j] = '\0';

printf("Position    ");
fgets(inf.position, N1, stdin);
for(int j = 0; j < N1; j++)
  if(inf.position[j] == '\n') inf.position[j] = '\0';
printf("Office      ");
fgets(inf.office, N2, stdin);

for(int j = 0; j < N2; j++)
  if(inf.office[j] == '\n') inf.office[j] = '\0';
printf("Age         ");
scanf("%d", &inf.age);
printf("Day         ");
scanf("%d", &inf.date.day);
printf("Month       ");
scanf("%d", &inf.date.month);
printf("Year        ");
scanf("%d", &inf.date.year);
printf("================================\n");
for(int i = KOL-1 ; i >= nomer; i--)
  pstr[i] = pstr[i-1];
pstr[nomer-1] = inf;
printf("\n\n   Інформація про працівників після додавання  \n");
print_struct(com, KOL, stdout);

//============================================================
//  Сортування за збільшенням дати початку роботи
//============================================================
for(int i = KOL-1; i > 0; i--)
{
  for(int j = 0; j < i; j++)
  {
    if(com[j].date.year >= com[j+1].date.year)
    {
      if(com[j].date.year == com[j+1].date.year)
      {
        if(com[j].date.month >= com[j+1].date.month)
        {
          if(com[j].date.month == com[j+1].date.month)
          {
            if(com[j].date.day > com[j+1].date.day)
              swap(com, j);
          }
          else swap(com, j);
        }
      }
      else swap(com, j);
    }
  }
}
```

```
  printf("\n\n      Відсортована інформація за збільшенням дати, "
         "з якої почалася робота      \n");
  print_struct(com, KOL, stdout);

  if((f1 = fopen("output.txt", "a")) == NULL)
  {
    fprintf(stderr, "Неможливо відкрити файл input.txt\n");
    exit(EXIT_FAILURE);
  }
  else
  {
    time(&rawtime);
    timeinfo = localtime(&rawtime);
    fprintf(f1, "\n%s\n", asctime(timeinfo));

    fprintf(f1, "\n\n     Відсортована інформація за збільшенням "
                "дати, з якої почалася робота        \n");
    print_struct(com, KOL, f1);
  }
  if(fclose(f1) != 0)
    fprintf(stderr, "Помилка під час закриття файлу\n");
  return 0;
}

int sum_vik(struct Company *str)
{
  int sum = 0;

  for(int i = 0; i < KOL; i++)
  {
    if(strcmp((str+i)->office, "London") == 0)
      sum += (str+i)->age;
  }
  return sum;
}

void print_struct(struct Company *str, int line, FILE *f)
{
  fprintf(f, "================================================\n");
  fprintf(f, " N Name     Start date   Position   Office  Age\n");
  fprintf(f, "================================================\n");
  for(int i = 0; i < line; i++)
  {
    fprintf(f, "%2d ", i+1);
    fprintf(f, "%-20s", (str+i)->name);
    fprintf(f, "%02d.%02d.%4d   ", (str+i)->date.day,
            (str+i)->date.month, (str+i)->date.year);
    fprintf(f, "%-29s", (str+i)->position);
    fprintf(f, "%-15s", (str+i)->office);
    fprintf(f, "%3d\n", (str+i)->age);
  }
  fprintf(f, "================================================\n");
}
```

```
void swap(struct Company *str, int j)
{
    // rab – для переставлення елементів структури
    struct Company rab;
    rab         = *(str+j);
    *(str+j)    = *(str+j+1);
    *(str+j+1)  = rab;
}
```

Фрагмент результатів роботи програми з обчислення сумарного віку осіб, що знаходяться в Лондоні, та видалення зазначеного рядка, номер якого вводився з клавіатури, наведено на рис. 1.8. Зверніть увагу, що на початку виводу інформації позначається поточний час запуску програми.

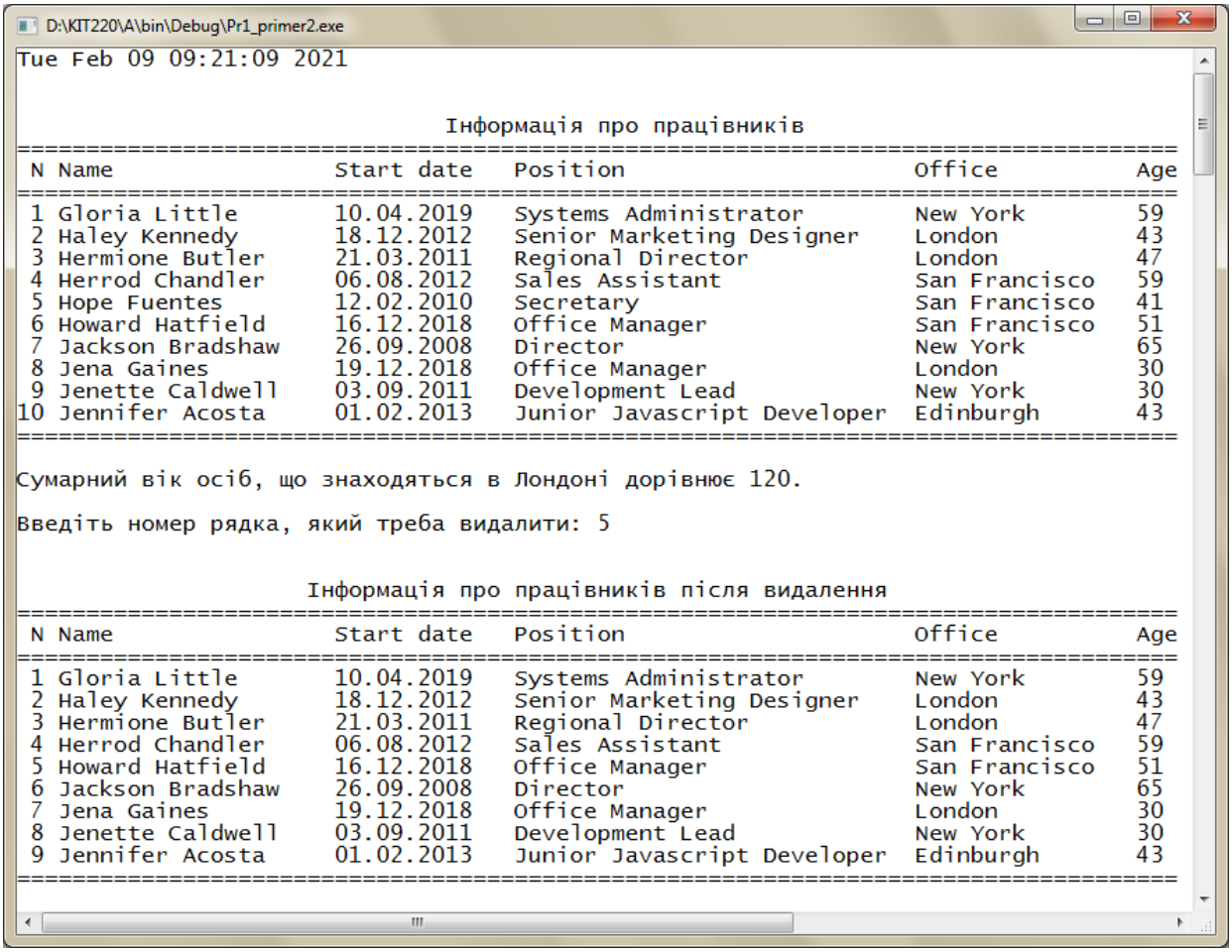


Рисунок 1.8 – Результати роботи програми (перша частина)

Фрагмент результатів роботи програми з введення інформації про нового працівника, та додавання її в зазначеному рядку, номер якого вводився з клавіатури, наведено на рис. 1.9.

Остання таблиця на рис. 1.9 є результатом сортування інформації за збільшенням дат, з яких розпочали роботу працівники в своїх компаніях.

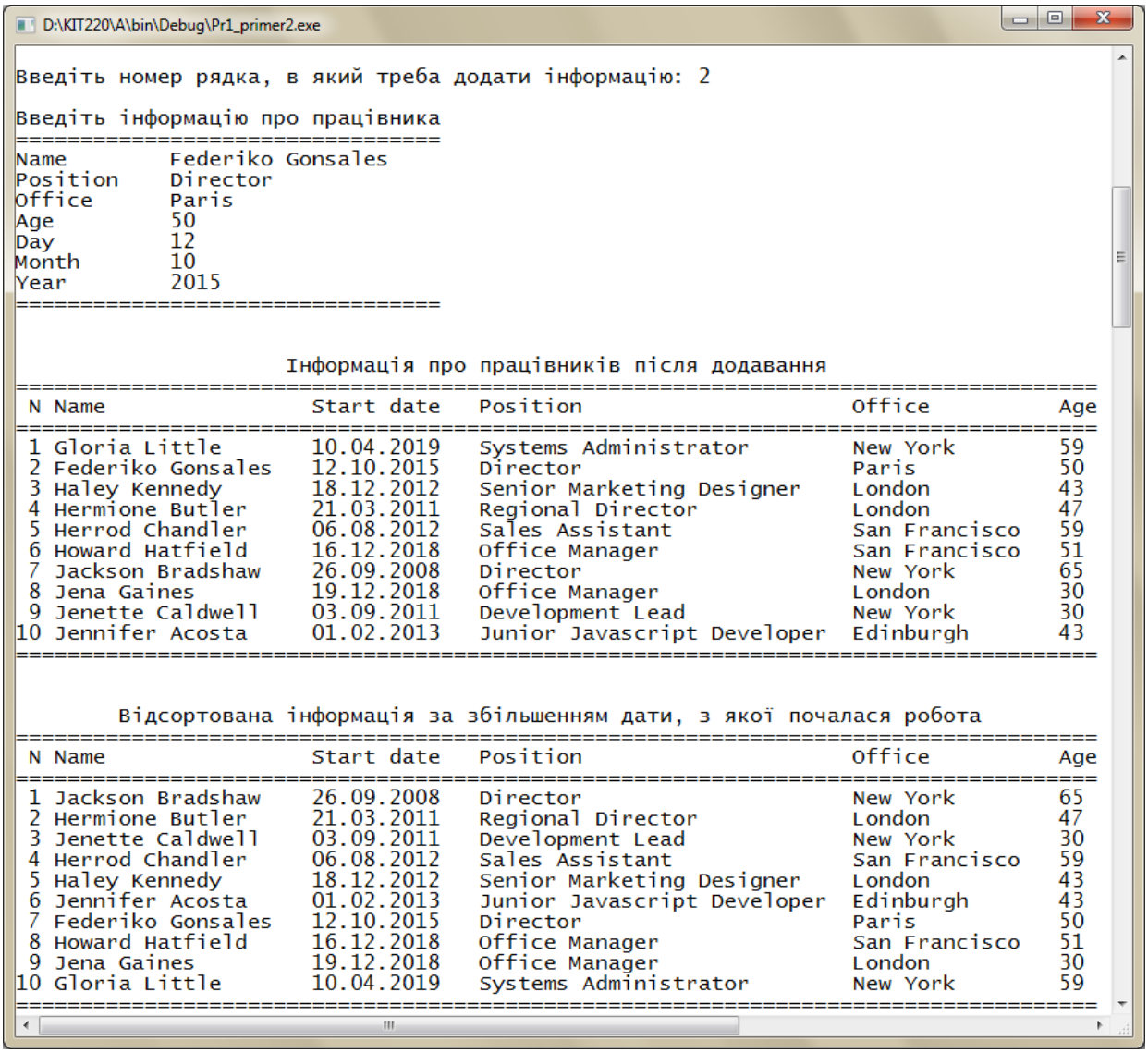


Рисунок 1.9 – Результати роботи програми (друга частина)

Слід зазначити, що ця таблиця також дублюється в файлі **`output.txt`** (рис. 1.10). На початку файлу також позначається поточний час запуску програми. Файл **`output.txt`** – відкритий для додавання, тому після наступного запуску програми, інформація буде додаватися до вже існуючої.

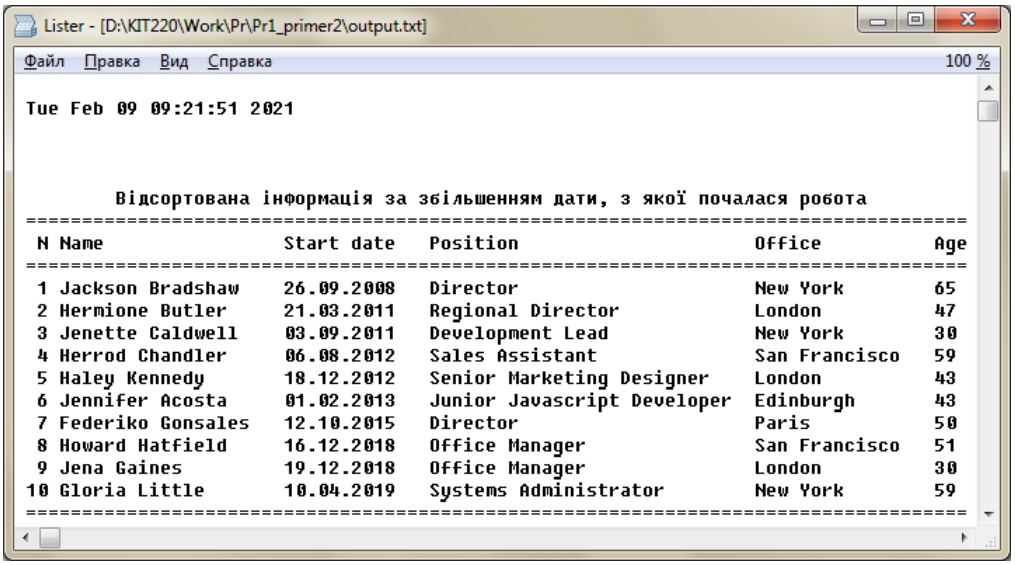


Рисунок 1.10 – Результати роботи програми (файл **`output.txt`**)

Завдання до практичної роботи:

1. Опрацюйте матеріал лекції «Структури», а також матеріал лабораторної роботи №1 «Дослідження основних можливостей використання структур в програмах на *C*», що присвячені питанням створення структур та роботі з ними.

2. Виконайте всі пункти завдання, які відповідають вашому варіанту (див. додаток 1).

Перш ніж виконувати завдання, необхідно створити файл **`input.txt`** та занести до нього необхідну інформацію.

Практична робота №2
РОБОТА З БІТАМИ

Мета роботи: отримати навички роботи з окремими бітами; навчитися використовувати порозрядні операції: доповнення до одиниці або побітове заперечення `(~)`, побітова операція "І" `(&)`, побітова операція "АБО" `(|)`, побітова операція "виключне АБО" `(^)`; відпрацювати навички роботи з побітовими операціями зсуву: зсув вліво `(<<)`, зсув вправо `(>>)`, циклічний зсув.

Теоретична частина

Порозрядні операції дозволяють конструювати вирази, в яких обробка операндів виконується на бітовому рівні (порозрядно) [1-5].

Розглянемо можливості роботи з бітами для наступних операцій:

~ – порозрядне заперечення (доповнення або інвертування бітів);

& – порозрядне "І" (порозрядна кон'юнкція);

¦ – порозрядне "АБО" (порозрядна диз'юнкція);

^ – порозрядна виключна диз'юнкція;

>> – зсув вправо послідовності бітів;

<< – зсув вліво послідовності бітів.

2.1. Побітові операції з цілими числами

2.1.1. Порозрядне заперечення

Операція порозрядного заперечення (доповнення або інвертування бітів) позначається символом `'~'` та є унарною (одномісною), тобто діє на один операнд, який повинен бути цілого типу. Значення операнда у вигляді внутрішнього бітового представлення обробляється таким чином, що формується значення тієї ж самої довжини (того самого типу), що й операнд. У бітовому представленні результату містяться 1 в усіх розрядах, де у операнда стоїть 0, і 0 в тих розрядах, де у операнда стоїть 1.

Приклад №1

Розглянемо варіант застосування операції порозрядного заперечення (~) для змінної типу **`unsigned char`** в програмі на *C*.

Текст програми має такий вигляд:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    unsigned char ch;
```

```
    SetConsoleOutputCP(1251);
    srand(time(NULL));
    ch = rand() % 128 + 128;
    printf("Початкове значення змінної ch:        %3d\n", ch);
    ch = ~ch;
    printf("Значення змінної ch після операції ~: %3d\n", ch);
    return 0;
}
```

Результат роботи програми для унарної операції (~) наведено на рис. 2.1.

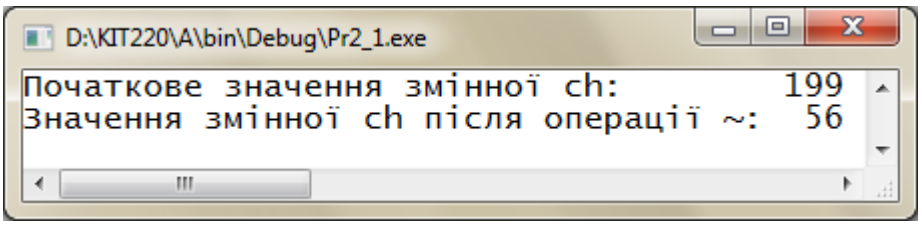


Рисунок 2.1 – Результат виконання програми для демонстрації операції порозрядного заперечення (~)

Для пояснення отриманого результату скористаємося калькулятором у режимі «Програміст». Число 199_{10} у двійковій системі числення буде представлено числом 11000111_2, а число 56_{10} – 0011100_2. Тобто, можна побачити, що отриманий результат є наслідком інвертування кожного розряду числа 199 на протилежне значення.

2.1.2. Порозрядна кон’юнкція

Порозрядна кон’юнкція (порозрядне "І") може бути застосована до цілочислових операндів. У бітовому представленні результату тільки ті біти дорівнюють 1, яким відповідають одиничні біти обох операндів.

Приклад №2

Розглянемо варіант застосування операції порозрядної кон’юнкції (&) для змінної типу **`unsigned char`** в програмі на *C*.

Текст програми має такий вигляд:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <time.h>
```

```
int main(void)
{
    unsigned char a, b, c;

    SetConsoleOutputCP(1251);

    srand(time(NULL));
    a = rand() % 128 + 128;
    printf("Значення змінної a:    %3d\n", a);
    b = rand() % 128 + 128;
    printf("Значення змінної b:    %3d\n", b);

    c = a & b;

    printf("============================\n");
    printf("Результат операції &:  %3d\n", c);
    printf("============================\n\n");
    return 0;
}
```

Результат роботи програми для бінарної операції (&) наведено на рис. 2.2.

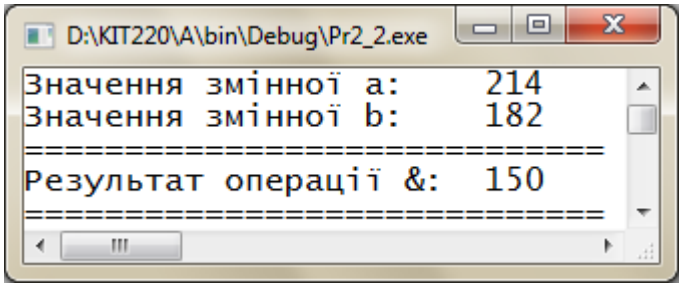


Рисунок 2.2 – Результат виконання програми для демонстрації бінарної операції порозрядної кон'юнкції (&)

Число 214_{10} у двійковій системі числення представлено числом 11010110_2, а число 182_{10} – 10110110_2. Якщо застосувати до операндів **a** і **b** операцію (&), то отримаємо ситуацію, яка наведена на рис. 2.3.

$$\begin{array}{rl} & 214_{10} = 1101\ 0110_2 \\ \& & 182_{10} = 1011\ 0110_2 \\ \hline & 150_{10} = 1001\ 0110_2 \end{array}$$

Рисунок 2.3 – Результат виконання бінарної побітової операції **'&'**

2.1.3. Порозрядна диз’юнкція

Порозрядна диз’юнкція (порозрядне "АБО") може бути застосована до цілочислових операндів. Вона дозволяє отримати 1 в усіх випадках, крім ситуації, коли розряди обох операндів дорівнюють 0.

Приклад №3

Розглянемо варіант застосування операції порозрядної диз’юнкції (¦) для змінної типу **`unsigned char`** в програмі на *C*.

Текст програми має такий вигляд:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    unsigned char a, b, c;

    SetConsoleOutputCP(1251);

    srand(time(NULL));
    a = rand() % 128 + 128;
    printf("Значення змінної a:    %3d\n", a);
    b = rand() % 128 + 128;
    printf("Значення змінної b:    %3d\n", b);

    c = a | b;

    printf("============================\n");
    printf("Результат операції ¦:  %3d\n", c);
    printf("============================\n\n");
    return 0;
}
```

Результат роботи програми для бінарної операції (¦) наведено на рис. 2.4.

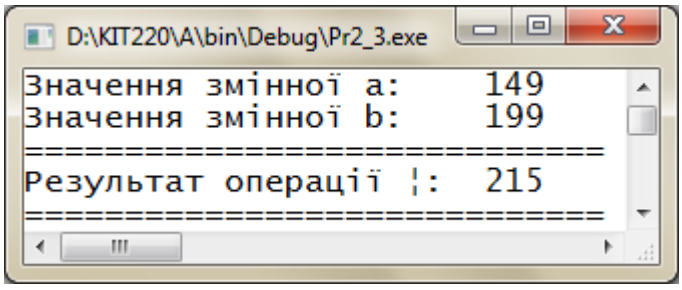


Рисунок 2.4 – Результат виконання програми для демонстрації бінарної операції порозрядної диз’юнкції (¦)

Число 149_{10} у двійковій системі числення представлено числом 10010101_2, а число 199_{10} – 11000111_2. Якщо застосувати до операндів **a** і **b** операцію (¦), то отримаємо ситуацію, яка наведена на рис. 2.5.

$149_{10} = 1001\ 0101_2$
$199_{10} = 1100\ 0111_2$

$215_{10} = 1101\ 0111_2$

Рисунок 2.5 – Результат виконання бінарної побітової операції (¦)

2.1.4. Порозрядна виключна диз'юнкція

Порозрядна виключна диз'юнкція (^) може бути застосована до цілих операндів. Результат формується при порозрядній обробці бітових кодів операндів. У тих розрядах, де обидва операнди мають однакові двійкові значення (1 і 1 або 0 і 0), результат приймає значення 0. У тих розрядах, де біти операндів не збігаються, – результат дорівнює 1.

Приклад №4

Розглянемо варіант застосування операції порозрядної виключної диз'юнкції (^) для змінної типу **unsigned char** в програмі на *C*.

Текст програми має такий вигляд:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <time.h>
int main(void)
{
    unsigned char a, b, c;
    SetConsoleOutputCP(1251);
    srand(time(NULL));
    a = rand() % 128 + 128;
    printf("Значення змінної a:    %3d\n", a);
    b = rand() % 128 + 128;
    printf("Значення змінної b:    %3d\n", b);
    c = a ^ b;
    printf("============================\n");
    printf("Результат операції ^:  %3d\n", c);
    printf("============================\n\n");
    return 0;
}
```

Результат роботи програми для бінарної операції (^) наведено на рис. 2.6.

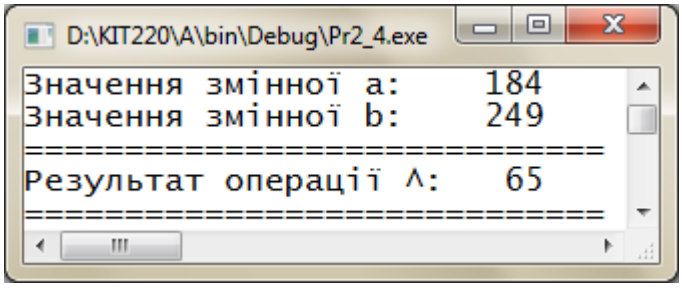


Рисунок 2.6 – Результат виконання програми для демонстрації бінарної операції порозрядної виключної диз'юнкції (^)

Число 184_{10} у двійковій системі числення представлено числом 10111000_2, а число 249_{10} – 11111001_2. Якщо застосувати до операндів **a** і **b** операцію (^), то отримаємо ситуацію, яка наведена на рис. 2.7.

$$
\begin{array}{rl}
 & 184_{10} = 1011\ 1000_2 \\
\wedge & 249_{10} = 1111\ 1001_2 \\
\hline
 & 65_{10} = 0100\ 0001_2
\end{array}
$$

Рисунок 2.7 – Результат виконання бінарної побітової операції (^)

2.2. Операції зсуву

2.2.1. Операція зсуву вліво

Операція зсуву вліво (<<) повинна мати цілочислові операнди. Над двійковим представленням значення лівого операнда виконується зсув бітів вліво. Правий операнд визначає величину порозрядного зсуву в бітах.

При зсуві вліво на **b** позицій двійкове представлення лівого операнда зсувається, а розряди, які звільняються праворуч, заповнюються нулями. Такий зсув еквівалентний множенню значення операнда на 2^b.

Приклад №5

Розглянемо варіант застосування в *C*-програмі операції порозрядного зсуву для змінної типу **unsigned char** на декілька бітів вліво.

Текст програми має такий вигляд:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    unsigned char a;  // Операнд, для якого треба виконати зсув
    unsigned char b;  // Кількість розрядів, на яку треба
                      // здійснити зсув
    unsigned char c;  // Результат виконання операції зсуву
                      // вліво
    SetConsoleOutputCP(1251);
    srand(time(NULL));
    a = rand() % 128 + 128;
    printf("Значення змінної a:     %3d\n", a);
    b = rand() % 7 + 1;
    printf("Значення змінної b:     %3d\n", b);
    c = a << b;
    printf("============================\n");
    printf("Результат операції <<:  %3d\n", c);
    printf("============================\n\n");
    return 0;
}
```

Результат роботи програми для демонстрації бінарної операції зсуву вліво (<<) наведено на рис. 2.8.

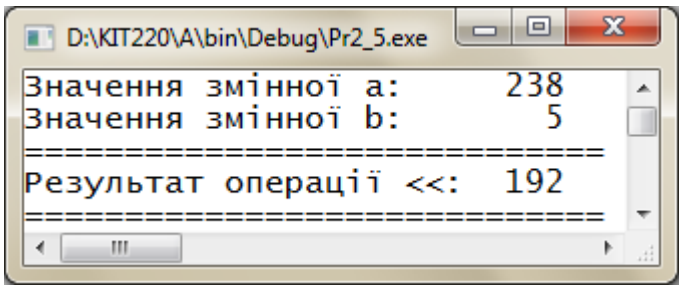


Рисунок 2.8 – Результат виконання програми для демонстрації бінарної операції зсуву вліво (<<) на 5 розрядів числа 238_{10}

Число 238_{10} у двійковій системі числення представлено числом 11101110_2. Якщо застосувати до операнда **a** операцію зсуву вліво (<<) на **b** позицій (**b** = 5), то отримаємо ситуацію, яка наведена на рис. 2.9.

238_{10} = 1110 1110_2 << 5_{10}

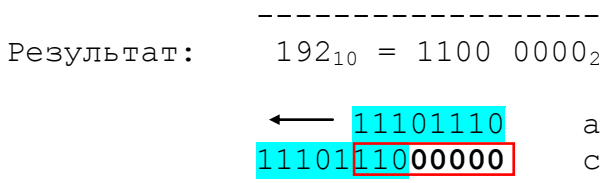


Рисунок 2.9 – Результат виконання бінарної побітової операції зсуву вліво (<<)

2.2.2. Операція зсуву вправо

Операція зсуву вправо (>>) повинна мати цілочислові операнди. Над двійковим представленням значення лівого операнда виконується зсув бітів вправо. Правий операнд визначає величину порозрядного зсуву.

Зсув вправо на **b** позицій відбувається дещо складніше. Тут слід зазначити дві особливості. Перша – це зникнення молодших розрядів, що виходять за розрядну сітку. Друга – відсутність стандарту на правило заповнення лівих розрядів, які звільняються. У стандарті мови *C* сказано, що коли лівий операнд є цілим значенням з від'ємним знаком, то при зсуві вправо заповнення лівих розрядів, які звільняються, визначається реалізацією. Тут можливі два варіанти: звільнені розряди заповнюються значеннями знакового розряду (арифметичний зсув вправо) або звільнені зліва розряди заповнюються нулями (логічний зсув вправо).

При додатному лівому операнді зсув вправо (>>) на **b** позицій еквівалентний зменшенню значення лівого операнда в 2^b разів з відкиданням дробової частини результату.

Приклад №6

Розглянемо варіант застосування операції порозрядного зсуву для змінної типу **unsigned char** на декілька бітів вправо в програмі на С.

Текст програми має такий вигляд:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <time.h>
int main(void)
{
```

```
    unsigned char a;  // Операнд, для якого треба виконати зсув
    unsigned char b;  // Кількість розрядів, на яку треба
                      // здійснити зсув
    unsigned char c;  // Результат виконання операції зсуву
                      // вправо
    SetConsoleOutputCP(1251);
    srand(time(NULL));
    a = rand() % 128 + 128;
    printf("Значення змінної a:     %3d\n", a);
    b = rand() % 7 + 1;
    printf("Значення змінної b:     %3d\n", b);
    c = a >> b;
    printf("============================\n");
    printf("Результат операції >>:  %3d\n", c);
    printf("============================\n\n");
    return 0;
}
```

Результат роботи програми для бінарної операції зсуву вправо (>>) наведено на рис. 2.10.

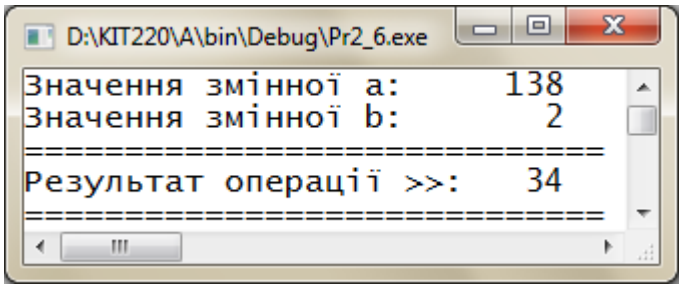


Рисунок 2.10 – Результат виконання програми для демонстрації бінарної операції зсуву вправо (>>) на 2 розряди числа 138_{10}

Число 138_{10} у двійковій системі числення представлено числом 10001010_2. Якщо застосувати до операнда **a** операцію зсуву вправо (>>) на **b** позицій (**b** = 2), то отримаємо ситуацію, яка наведена на рис. 2.11.

$138_{10} = 1000\ 1010_2 >> 2_{10}$

Результат: $34_{10} = 0010\ 0010_2$

→ 10001010 a

00100010 10 c

Рисунок 2.11 – Результат виконання бінарної побітової операції зсуву (>>)

2.2.3. Операції циклічного зсуву

В операціях циклічного зсуву зміщення бітів може виконуватися як вліво так і вправо. Кругова операція лівого зсуву робить зміщення кожного біту в **n**-бітовому операнді на **k** позицій вліво. Крайні ліві **k**-бітів видаляються зліва і стають крайніми правими бітами. Кругова операція правого зсуву робить зміщення кожного біту в **n**-бітовому операнді на **k** позицій вправо. Крайні праві **k**-бітів справа видаляються та стають крайніми лівими бітами. На рис. 2.12 наведені обидва види циклічного зсуву для **n = 8** і **k = 3**.

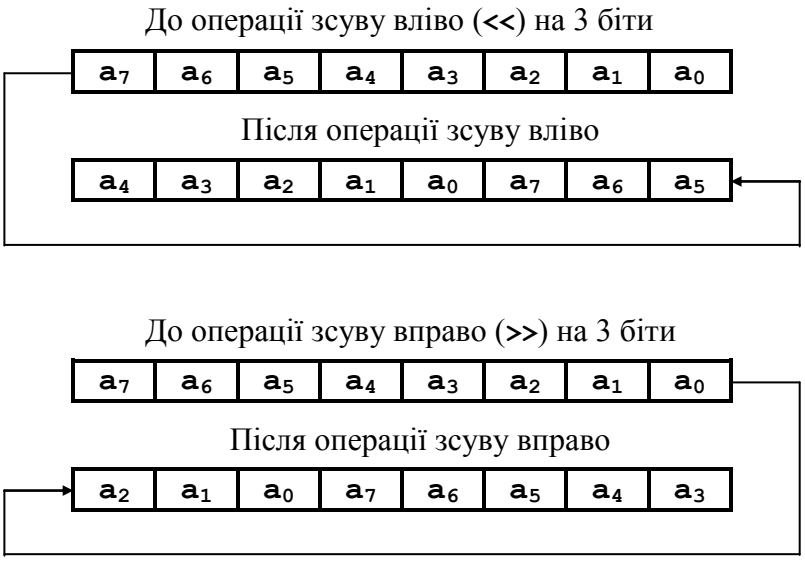


Рисунок 2.12 – Операції циклічного зсуву на 3 біти вліво (**<<**) та вправо (**>>**)

Приклад №7

Розглянемо варіант застосування операцій циклічного зсуву для змінної **a** типу **unsigned char** на **k** бітів вліво та вправо.

Якщо проаналізувати рис. 2.12, то можна побачити, що алгоритм для операції циклічного зсуву вліво буде складатися з наступних кроків:

```
1) занести до змінної a значення типу unsigned char;
2) ввести значення k;
3) b = a << k;        // біти, що зміщуються в межах байту
4) c = a >> (8 - k); // біти, які піддаються циклічному зсуву
5) b = b | c;         // об'єднуємо біти з пунктів 3 і 4
6) здійснити вивід змінної b на екран.
```

Алгоритм для операції циклічного зсуву вправо буде складатися з наступних кроків:

```
1) занести до змінної a значення типу unsigned char;
2) ввести значення k;
3) b = a >> k;        // біти, що зміщуються в межах байту
4) c = a << (8 - k); // біти, які піддаються циклічному зсуву
5) b = b | c;         // об'єднуємо біти з пунктів 3 і 4
6) здійснити вивід змінної b на екран.
```

Текст *C*-програми для реалізації операцій циклічного зсуву вправо та вліво має такий вигляд:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <time.h>
#define  N  8

void print_byte(unsigned char b);

int main(void)
{
    unsigned char a;  // Операнд, для якого треба виконати зсув
    unsigned char b;  // Операнд, що буде зберігати зсув
    unsigned char c;  // Додатковий операнд
    unsigned char k;  // Кількість розрядів зсуву
    SetConsoleOutputCP(1251);
    srand(time(NULL));
    printf("Циклічний зсув вліво\n");
    a = rand() % 128 + 128;
    k = rand() % 7 + 1;
    b = a << k;
    c = a >> (8-k);
    b = b | c;
    printf("====================\n");
    printf("%u << %u = %u\n", a, k, b);
    printf("====================\n");
    printf("Змінна a:  ");
    print_byte(a);
    printf("Результат: ");
    print_byte(b);
    printf("====================\n\n");
    printf("Циклічний зсув вправо\n");
    a = rand() % 128 + 128;
    k = rand() % 7 + 1;
    b = a >> k;
    c = a << (8-k);
    b = b | c;
    printf("====================\n");
    printf("%u >> %u = %u\n", a, k, b);
    printf("====================\n");
```

```
    printf("Змінна a:  ");
    print_byte(a);
    printf("Результат: ");
    print_byte(b);
    printf("====================\n\n");
    return 0;
}
void print_byte(unsigned char b)
{
    char bin_str[N+1];
    unsigned char c;
    int i, j;
    j = N-1;
    for(i = 0; i < N; i++)
    {
        c = (b >> i) & 1;
        bin_str[j] = (c == 0) ? '0' : '1';
        j--;
    }
    bin_str[N] = '\x0';
    for(i = 1; i < N+1; i++)
    {
        printf("%c", bin_str[i-1]);
        if(i % 4 == 0) printf("%c", ' ');
    }
    printf("\n");
}
```

Результати роботи програми наведені на рис. 2.13 та рис. 2.14.

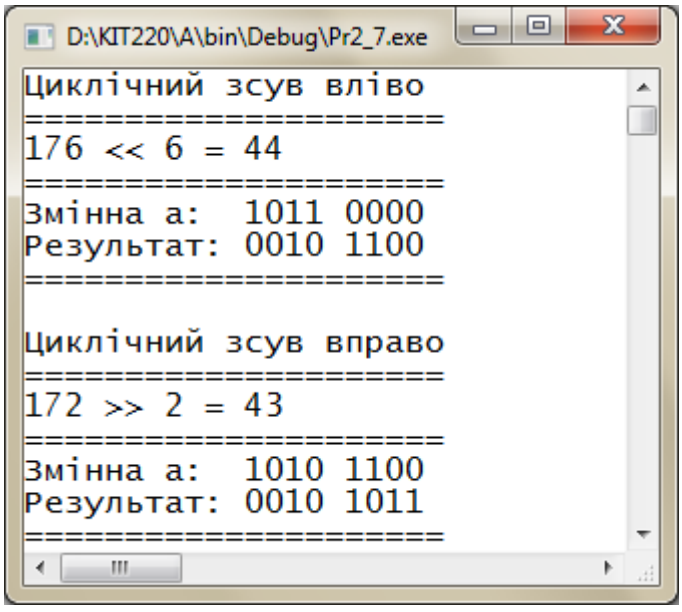


Рисунок 2.13 – Результат роботи програми для демонстрації циклічного зсуву вліво та вправо (перший запуск)

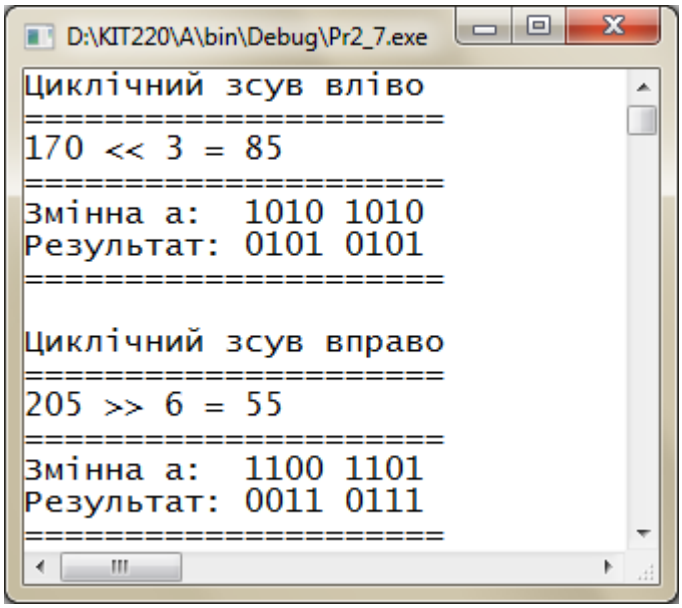


Рисунок 2.14 – Результат роботи програми для демонстрації циклічного зсуву вліво та вправо (другий запуск)

Завдання до практичної роботи:

1. Опрацюйте матеріал лекції «Маніпулювання бітами», а також теоретичну частину даної практичної роботи.

2. Виконайте завдання, яке відповідає вашому варіанту (див. додаток 2). Під час написання програми намагайтеся оптимально застосовувати бітові операції мови *C*.

Практична робота №3
ДИНАМІЧНЕ ВИДІЛЕННЯ ПАМ'ЯТІ ТА БАГАТОФАЙЛОВА КОМПІЛЯЦІЯ

Мета роботи: отримати навички роботи з організації багатофайлової компіляції; навчитися використовувати специфікатор класу зберігання **`extern`**; відпрацювати навички роботи з функціями виділення пам'яті: `malloc()`, `calloc()` і `realloc()`.

Теоретична частина

Будь-яку програму, текст якої займає сотні рядків, можна зберігати в одному файлі. Навіть якщо допустити, що файл з програмою може бути будь-якого розміру, модифіковувати та налагоджувати програму, розмір якої становить тисячі рядків, а на екрані видно тільки близько 20 рядків, дуже незручно. Крім цього, при збільшенні довжини програми значно збільшується і час її компіляції. Коли розмір програми досягає певної межі, доводиться поділяти її на частини, кожна з яких зберігається в окремому файлі [1].

Це ж саме можна сказати і про випадок, коли одну програму розробляє група людей. Кожен файл повинен включати одну або декілька функцій.

Переваги такого підходу полягають в:

– розробці та налагодженні різних частин програми, які можна доручити різним людям;

– спрощенні та зручності налагоджування та модифікації програми;

– прискоренні роботи, оскільки при запуску програми можна компілювати лише ті файли, які змінювалися, для інших можна використовувати вже існуючі **obj**-файли.

C-файлів може бути багато, а компіляція – досить довгий процес, тому це дає значний виграш у часі в процесі роботи. Ця послідовність дій (створення спочатку об’єктних файлів, а потім самої програми) є типовою для багатьох компіляторів і сильно економить час при розробці великих програм, тому що обробляються тільки тексти *C*-програм зі змінами. Потім об’єктні файли надходять до редактору зв’язків, і на виході отримуємо готову програму.

Існує два підходи до багатофайлової розробки програм на мові *C*:

1) Підключати потрібні *C*-файли за допомогою директиви

```
#include "*.c"
```

до головного *C*-файлу.

2) Використовувати проєкти для об’єднання кількох С-файлів в одну програму.

Перший підхід є не дуже вигідним, оскільки препроцесор просто створює один великий файл, який весь компілюється.

Другий підхід застосовується в багатьох інтегрованих середовищах розробки (в тому числі і в Code::Blocks). Цей підхід дозволяє використовувати всі переваги багатофайлової розробки програм.

Проєкт – це спеціальний файл, в який записуються імена файлів, які IDE слід об'єднувати в один виконуваний **exe**-файл. Ім'я файлу, що виконується, збігається в такому випадку з ім'ям проєкту, а не з ім'ям *C*-файлу, як це було при однофайловій програмі.

Розглянемо на прикладі процес багатофайлової компіляції. Візьмемо одну з програм, яка була розглянута на лекції «Можливості структур», і шляхом її модифікації спробуємо розмістити текст програми у двох файлах.

Початковий текст програми має такий вигляд:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>

struct Flex
{
    size_t count;
    double average;
    double scores[];   // член з типом гнучкого масиву
} *pf1, *pf2;
void showFlex(const struct Flex *p);

int main(void)
{
    int n = 5;
    int i;
    int tot = 0;
    SetConsoleOutputCP(1251);
    // виділення пам'яті для структури та масиву
    pf1 = malloc(sizeof(struct Flex) + n * sizeof(double));
    pf1->count = n;
    for(i = 0; i < n; i++)
    {
        pf1->scores[i] = 20.0 - i;
        tot += pf1->scores[i];
    }
    pf1->average = tot / n;
    showFlex(pf1);
    n = 9;
```

```
    tot = 0;
    pf2 = malloc(sizeof(struct Flex) + n * sizeof(double));
    pf2->count = n;
    for(i = 0; i < n; i++)
    {
        pf2->scores[i] = 20.0 - i / 2.0;
        tot += pf2->scores[i];
    }
    pf2->average = tot / n;
    showFlex(pf2);
    free(pf1);
    free(pf2);
    return 0;
}

void showFlex(const struct Flex *p)
{
    int i;

    printf("Рейтинги: ");
    for(i = 0; i < p->count; i++)
        printf("%g ", p->scores[i]);
    printf("\nСереднє значення: %g\n", p->average);
}
```

Для виконання цієї програми необхідно створити проєкт, що містить єдиний файл **`main.c`** і провести його компіляцію.

Результат роботи програми наведено на рис. 3.1.

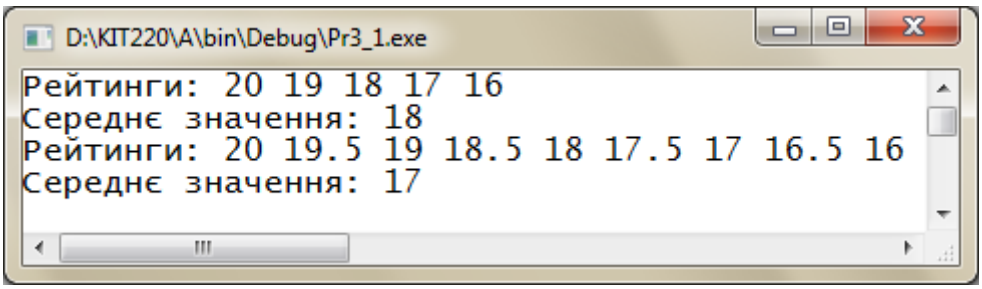


Рисунок 3.1 – Результат виконання програми, текст якої міститься в одному файлі

Спробуємо створити ще один файл **`ShowFlex.c`**, в якому розмістимо функцію `showFlex()`, а з файлу **`main.c`** її вилучимо. Для цього обираємо пункт меню `File` → `New` → `File...` (рис. 3.2) і у вікні «New from template» (рис. 3.3) обираємо «C/C++ source» та натискаємо на кнопку «Go».

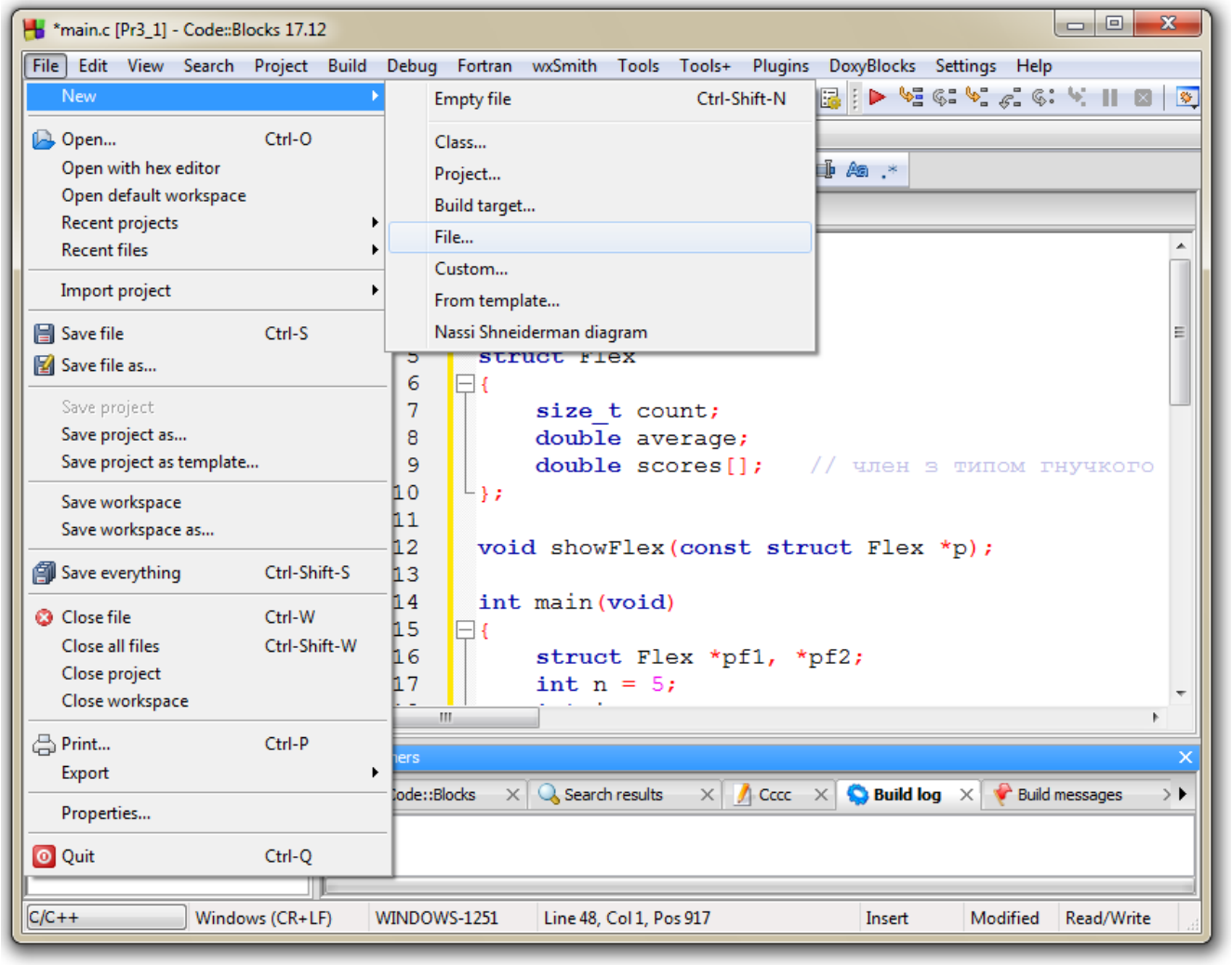


Рисунок 3.2 – Вікно програми *Code::Blocks* з текстом програми

В наступному вікні (рис. 3.4) ставимо позначку «Skip this page next time» для того, щоб пропускати його в подальшому. У вікні «*C*/*C*++ source» необхідно обрати мову *C* (рис. 3.5) та натиснути на кнопку «Next».

Після цього з'явиться вікно «*C*/*C*++ source», в якому необхідно поставити всі позначки і додати ім'я нового файлу *C*-програми разом з повним шляхом до нього (рис. 3.6) та натиснути на кнопку «Finish».

В результаті відкриється вікно для вводу тексту функції до файлу **`ShowFlex.c`** (рис. 3.7).

На рис. 3.8 зображено введений текст функції `showFlex()`.

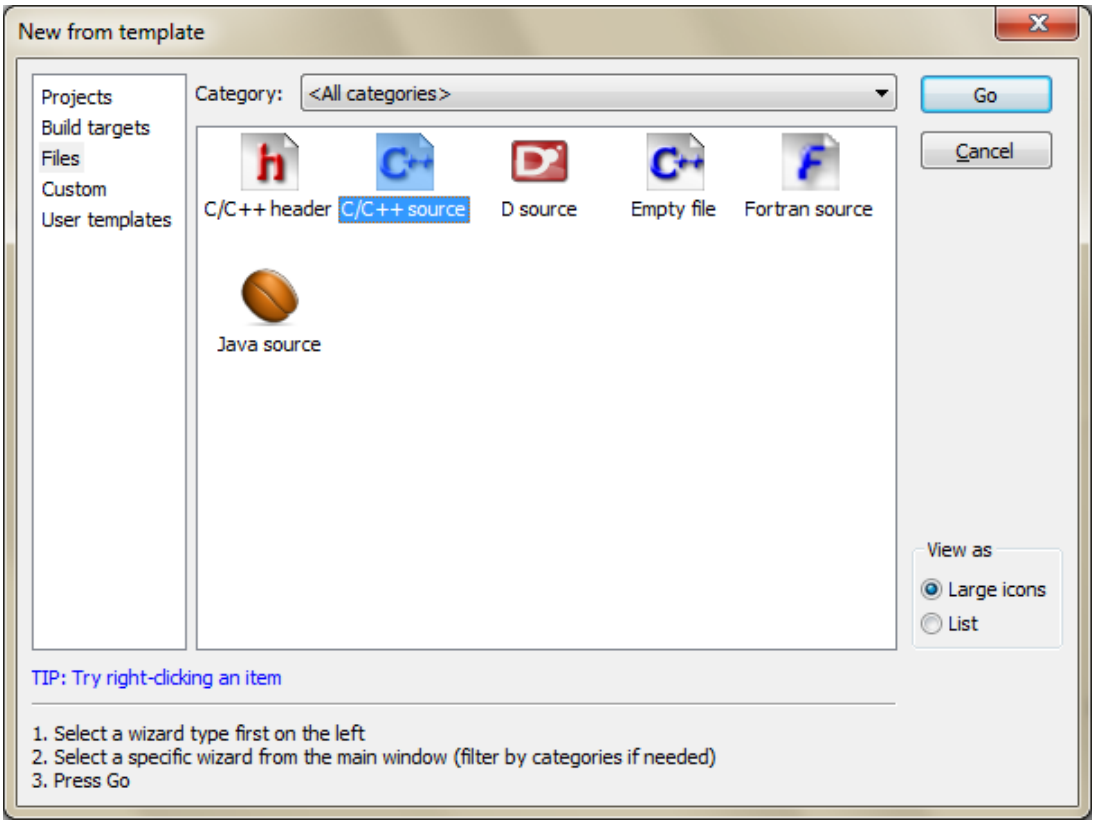


Рисунок 3.3 – Вікно «New from template» для вибору шаблону файлу

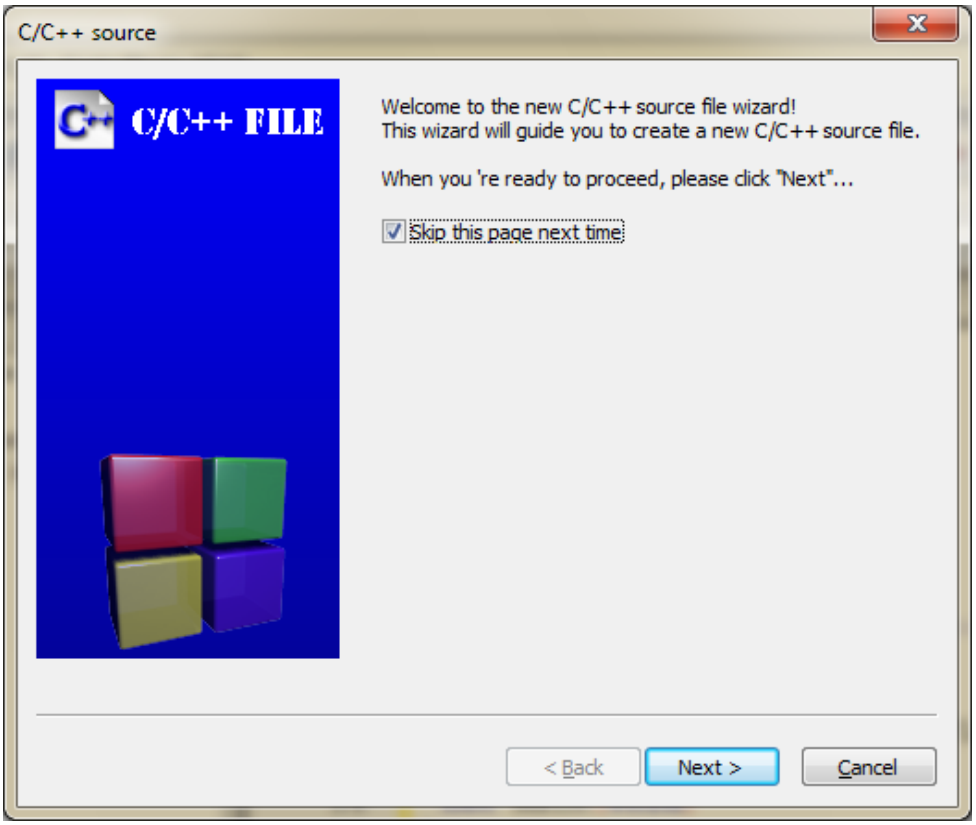


Рисунок 3.4 – Вікно «*C/C*++ source», в якому необхідно поставити позначку

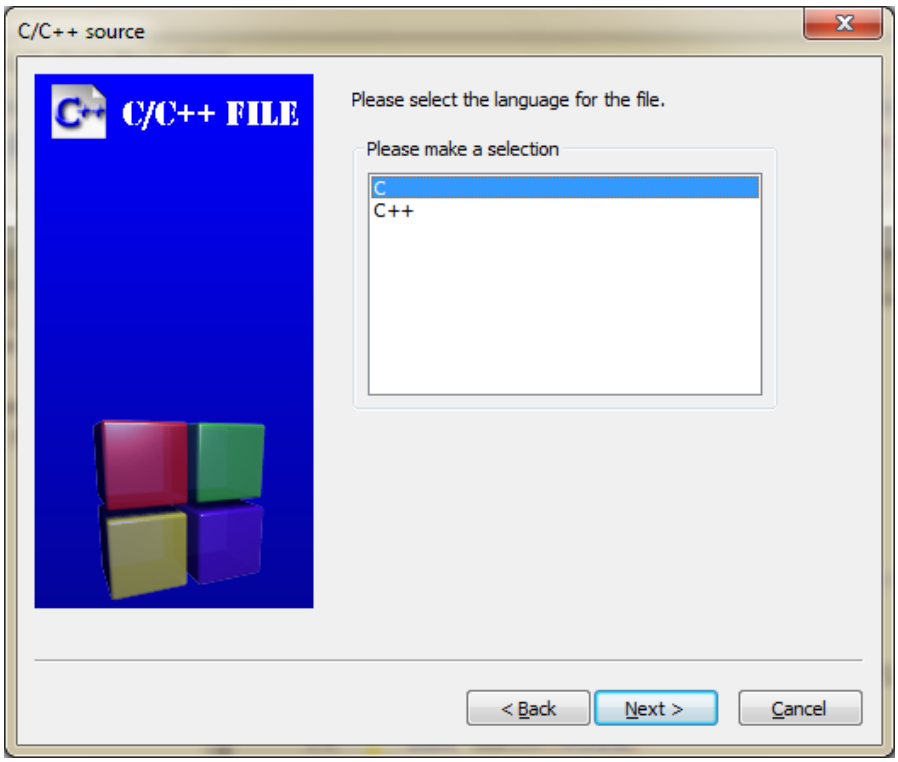


Рисунок 3.5 – Вікно «*C*/*C*++ source», в якому необхідно обрати мову *C*

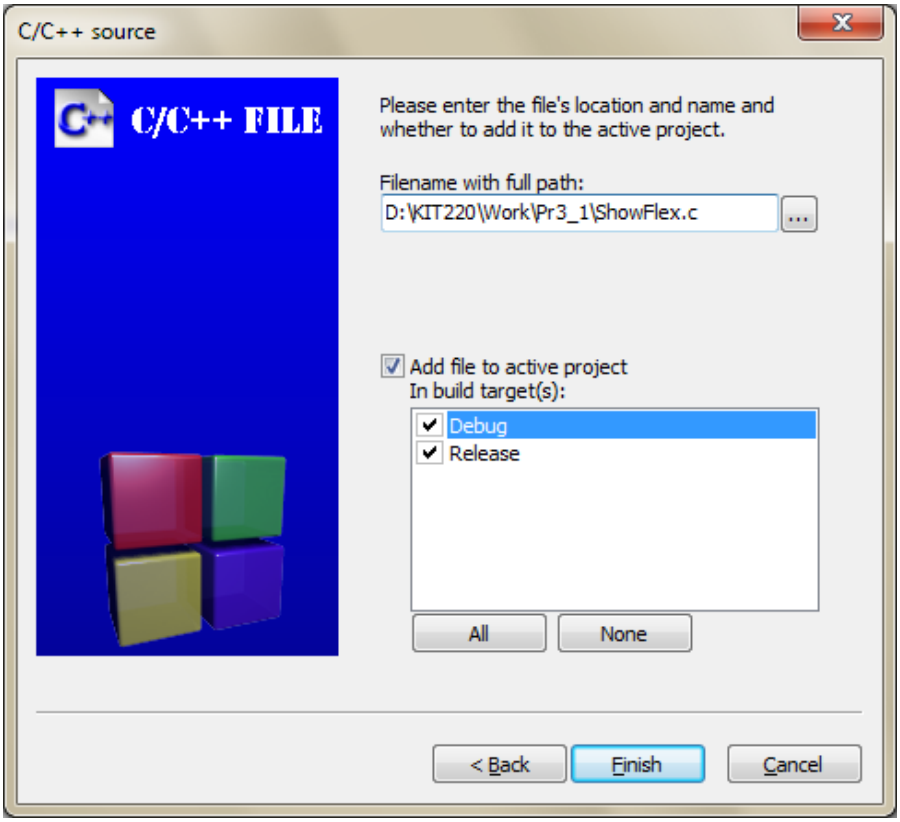


Рисунок 3.6 – Вікно «*C*/*C*++ source», де необхідно поставити всі позначки і додати ім’я нового файлу *C*-програми разом з повним шляхом до нього

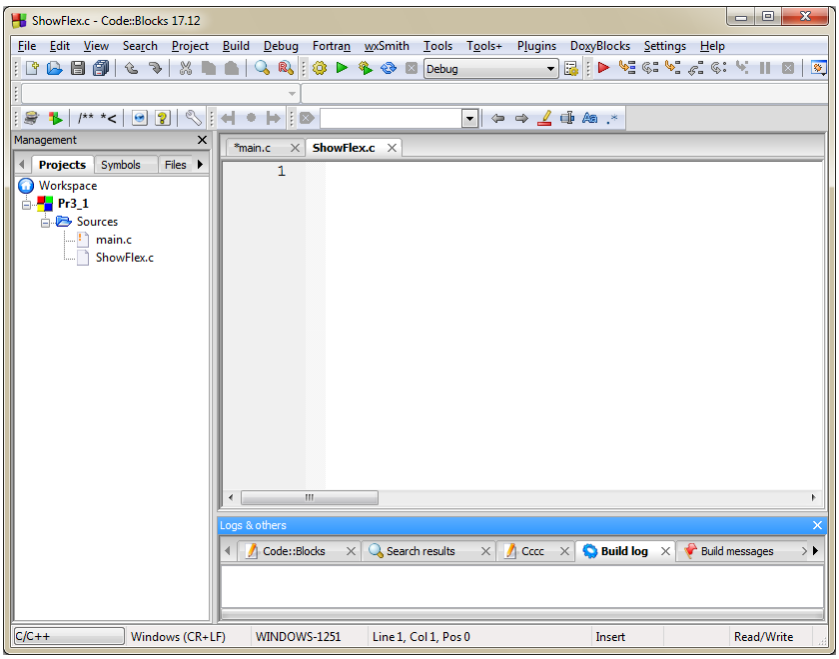


Рисунок 3.7 – Вікно «ShowFlex.c» для вводу тексту функції `showFlex()`

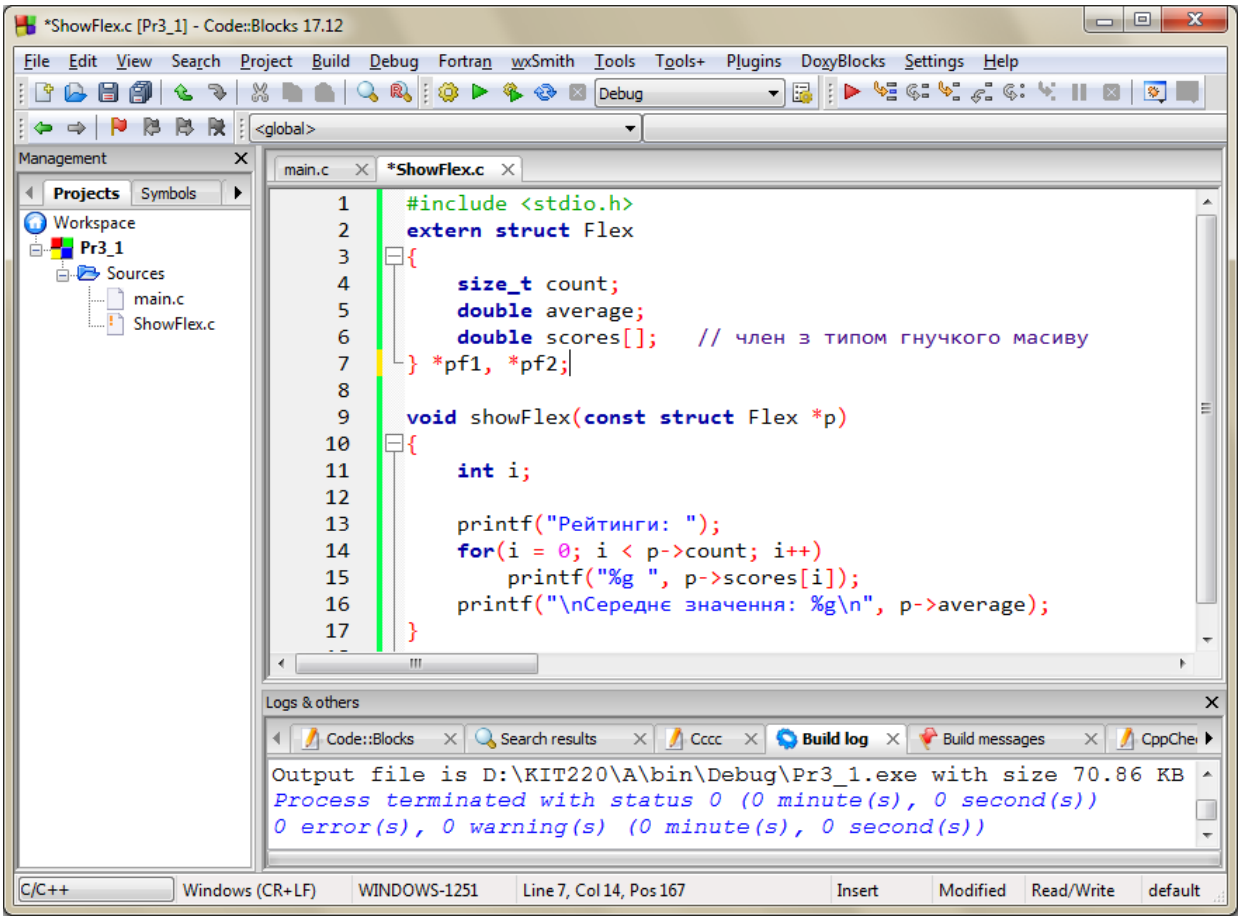


Рисунок 3.8 – Вікно з текстом функції `showFlex()` і результатом компіляції

При роботі з проєктом виникає необхідність взаємодії окремих файлів. Тобто функція з одного файлу, повинна мати доступ до змінної або функції з іншого файлу. Розглянемо, які зміни необхідно зробити з вхідним файлом для компіляції проєкту, який складається з двох *C*-файлів.

У файлі **main.c** необхідно видалити текст функції `showFlex()` і директиву препроцесора `#include <stdio.h>`, оскільки вивід інформації буде відбуватися в іншому файлі, де буде розміщатися функція `showFlex()`.

Таким чином текст програми з файлу **main.c** буде мати такий вид:

```
#include <windows.h>
#include <stdlib.h>

struct Flex
{
    size_t count;
    double average;
    double scores[];   // член структури з типом гнучкого масиву
} *pf1, *pf2;

void showFlex(const struct Flex *p);

int main(void)
{
    int n = 5;
    int i;
    int tot = 0;

    SetConsoleOutputCP(1251);

    // виділення пам'яті для структури та масиву
    pf1 = malloc(sizeof(struct Flex) + n * sizeof(double));
    pf1->count = n;
    for(i = 0; i < n; i++)
    {
        pf1->scores[i] = 20.0 - i;
        tot += pf1->scores[i];
    }
    pf1->average = tot / n;
    showFlex(pf1);
    n = 9;
    tot = 0;
    pf2 = malloc(sizeof(struct Flex) + n * sizeof(double));
    pf2->count = n;
    for(i = 0; i < n; i++)
    {
        pf2->scores[i] = 20.0 - i / 2.0;
        tot += pf2->scores[i];
    }
```

```
    pf2->average = tot / n;
    showFlex(pf2);
    free(pf1);
    free(pf2);
    return 0;
}
```

Текст файлу **ShowFlex.c** буде мати такий вигляд:

```
#include <stdio.h>

extern struct Flex
{
    size_t count;
    double average;
    double scores[];     // член структури з типом гнучкого масиву
} *pf1, *pf2;            // вказівники на структуру Flex,
                         // що знаходяться в іншому
                         // файлі (специфікатор extern)

void showFlex(const struct Flex *p)
{
    int i;

    printf("Рейтинги: ");
    for(i = 0; i < p->count; i++)
        printf("%g ", p->scores[i]);
    printf("\nСереднє значення: %g\n", p->average);
}
```

Завдання до практичної роботи:

1. Опрацюйте матеріал лекції «Специфікатори класів зберігання. Виділення пам’яті», а також відповідні матеріали слайдів презентації.

2. Напишіть *C*-програму, яка крім функції `main()` містить ще три функції `fun_malloc()`, `fun_calloc()` і `fun_realloc()`. Кожну з функцій треба розмістити в окремому файлі та провести багатофайлову компіляцію. У тексті функції `main()` необхідно передбачити нескінчений цикл з можливістю виклику будь-якої з перелічених трьох функцій у довільному порядку. Слід також передбачити вихід з програми при натисканні на клавішу «`0`». Дані для реалізації функцій `fun_malloc()` і `fun_calloc()` знаходяться в табл. 3.1.

При запуску програми на виконання екран повинен мати такий вид:

```
================================================================
Програма демонструє можливості функцій для виділення пам'яті.
Для реалізації програми використовується багатофайлова компіляція.
================================================================
    1 - Виділення блоку пам'яті за допомогою функції malloc()
    2 - Виділення блоку пам'яті за допомогою функції calloc()
    3 - Збільшення блоку пам'яті за допомогою функції realloc()
    0 - Вихід з програми
================================================================
Оберіть необхідну дію: 1
================================================================
Демонстрація роботи функції malloc()
...
================================================================
Оберіть необхідну дію: 0
================================================================
Програма завершена!!!
================================================================
```

Розглянемо призначення функцій програми:

1) Функція `fun_malloc()` виділяє за допомогою `malloc()` пам'ять під двовимірний масив чисел типу **`int`** розмірністю **`m×n`**. Потім вона заносить до нього випадкові числа в діапазоні від $\mathbf{d_1}$ до $\mathbf{d_2}$ та виводить елементи масиву на консоль. Наприкінці функції обчислюється сума елементів масиву, які знаходяться в **`r`**-му рядку.

2) Функція `fun_calloc()` виділяє за допомогою `calloc()` пам'ять під одновимірний масив, що містить **`s`** чисел типу **`float`**. Потім вона заносить до нього випадкові числа в діапазоні від $\mathbf{t_1}$ до $\mathbf{t_2}$ та виводить елементи масиву на консоль. Наприкінці функції обчислюється сума елементів масиву, значення яких більше або дорівнюють значенню **`p`**.

3) Функція `fun_realloc()` виділяє за допомогою `malloc()` блок пам'яті під символьний масив, в який заносить через пробіл ваше прізвище та ім'я. Далі за допомогою функції `realloc()` вона збільшує розмір блоку з таким розрахунком, щоб вмістити в нього через пробіл ваше прізвище, ім'я та по батькові. Наприкінці функції `fun_realloc()` на консоль виводиться вміст блоків до та після збільшення розміру.

Таблиця 3.1 – Вхідні дані для реалізації функцій

№ варіанту	Функції								
	fun_malloc()					fun_calloc()			
	m	**n**	**d_1**	**d_2**	**r**	**s**	**t_1**	**t_2**	**p**
1	5	6	100	200	3	10	10.24	20.50	15.00
2	4	4	56	67	2	15	22.87	33.88	27.46
3	7	4	30	70	5	20	54.09	98.98	70.88
4	3	3	11	33	2	11	27.99	37.76	32.00
5	5	5	28	58	4	16	22.00	44.00	31.00
6	6	4	25	76	3	18	43.09	97.00	70.45
7	4	6	20	43	1	8	21.00	23.00	22.00
8	6	6	55	66	4	14	41.93	57.39	51.34
9	3	8	34	73	2	22	34.09	76.88	50.00
10	4	7	33	99	3	24	47.00	78.56	60.94
11	5	8	43	69	4	12	32.98	43.29	40.00
12	7	7	33	77	5	16	34.09	54.39	44.00
13	4	4	11	33	2	10	32.09	35.87	34.00
14	6	6	25	75	4	12	22.77	44.43	40.65
15	5	5	47	87	3	16	12.37	43.78	30.68
16	4	8	28	63	4	17	26.09	69.33	32.11
17	8	7	77	99	5	14	12.65	28.98	20.00
18	5	6	26	46	4	12	15.51	37.43	22.43
19	4	7	12	56	2	17	24.22	57.23	32.35
20	5	4	32	68	3	15	37.57	67.32	43.32

Практична робота №4

РОБОТА ЗІ ЗВ’ЯЗНИМИ СПИСКАМИ

Мета роботи: отримати навички роботи зі зв’язними списками; навчитися використовувати основні операції зі створення та видалення списку, додавання або видалення вузла на початку чи наприкінці списку.

Теоретична частина

Зв’язний список – це структура даних, в якій кожен елемент (вузол) зберігає деяку інформацію, а також посилання на наступний елемент [6].

Останній елемент списку посилається на `NULL`.

Для нас зв’язний список корисний тим, що він:

– дуже легко може бути побудований;

– усі алгоритми роботи зі списком є інтуїтивно зрозумілими;

– надає можливість відпрацювати роботу з вказівниками;

– легко візуалізується, що дозволяє наочно пояснити суть алгоритму;

– часто використовується у програмуванні;

– легко визначається рекурсивно, через що досить часто застосовується в рекурсивних алгоритмах.

Для спрощення роботи розглянемо зв'язний список, який зберігає цілочислові значення (рис. 4.1).

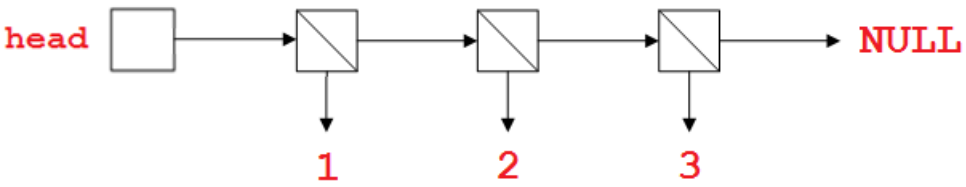


Рисунок 4.1 – Зв'язний список, що складається з трьох вузлів

Такий зв'язний список складається з вузлів, кожен з яких містить цілочислове значення та вказівник на наступний вузол. Вузол можна представити у вигляді структури **node**. Щоб не писати кожного разу **struct** node визначимо новий тип **Node** за допомогою **typedef**:

```
typedef struct node
{
    int value;
    struct node *next;
} Node;
```

Тепер необхідно написати функцію push(), яка створює список зі значень, що передаються їй в якості параметрів. Нове значення буде вставлятися на початку списку. Кожен новий елемент списку будемо створювати в купі – пам'яті, що використовується для розміщення об'єктів, які динамічно створюються програмою.

Для подальших дій зі списком буде досить корисним мати вказівник на перший елемент списку:

```
Node *head = NULL;
```

Тобто спочатку списку немає, а вказівник посилається на `NULL` (рис. 4.2).

Рисунок 4.2 – Ініціалізація вказівника на голову списку

4.1. Додавання нового вузла на початку списку

Для додавання нового вузла необхідно:

– виділити під нього пам’ять;

– присвоїти йому значення;

– зробити так, щоб він посилався на перший елемент списку (або на `NULL`, якщо його немає);

– спрямувати вказівник **`head`** на доданий вузол.

Розглянемо реалізацію функції `push()`.

1) Виділяємо фрагмент пам’яті під новий вузол і посилаємося на нього за допомогою вказівника **`tmp`** (рис. 4.3).

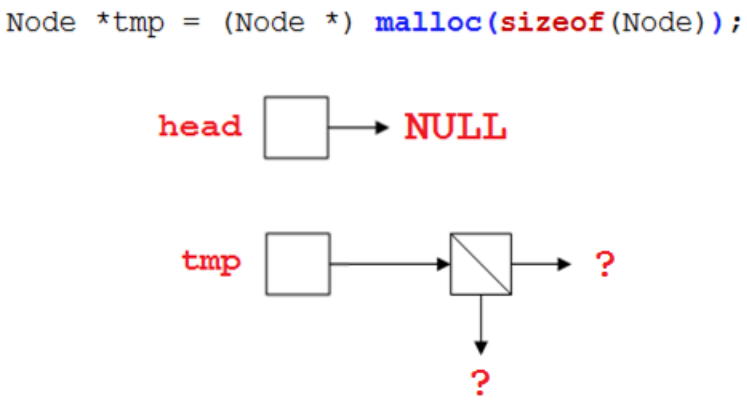


Рисунок 4.3 – Створення нового вузла, на який посилається змінна `tmp`

2) Присвоюємо полю `tmp->value` значення **`data`**, яке передається до функції `push()` в якості параметру (рис. 4.4).

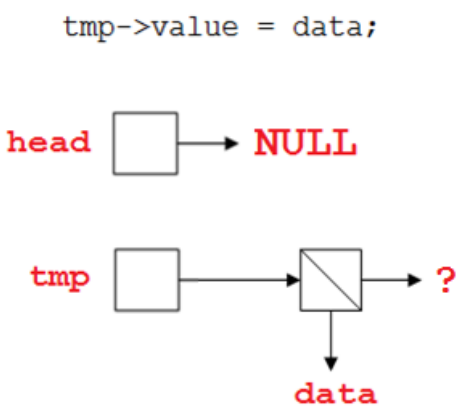


Рисунок 4.4 – Присвоєння полю `tmp->value` значення `data`

3) Присвоюємо вказівнику на наступний вузол для **`tmp`** значення адреси для голови списку (рис. 4.5).

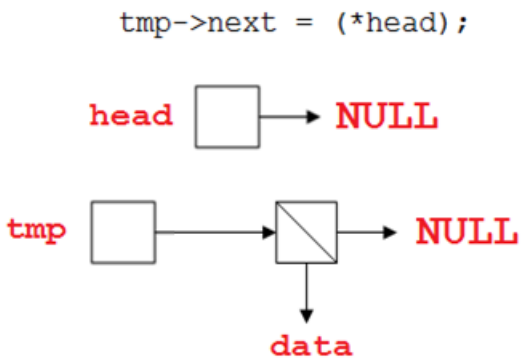


Рисунок 4.5 – Присвоєння вказівнику на наступний вузол для `tmp` значення адреси для голови списку

4) Змінюємо значення вказівника **`head`** на адресу нового вузла (рис. 4.6).

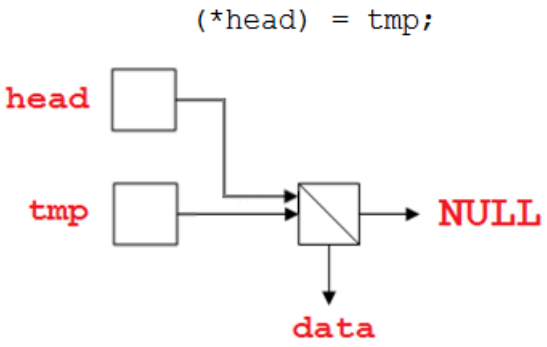


Рисунок 4.6 – Спрямування вказівника `head` на щойно створений вузол

5) Після виходу з функції `push()` локальна змінна **`tmp`** буде знищена. В результаті отримаємо список, до якого буде доданий новий елемент (рис. 4.7).

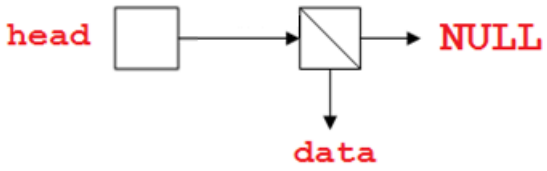


Рисунок 4.7 – Додавання нового вузла на початку списку

Таким чином, текст функції `push()` для додавання нового вузла на початку списку має такий вигляд:

```
void push(Node **head, int data)
{
    Node *tmp = (Node *) malloc(sizeof(Node));
    tmp->value = data;
    tmp->next = (*head);
    (*head) = tmp;
}
```

Оскільки вказівник **`head`** змінюється, то необхідно в параметрах функції `push()` передавати вказівник на вказівник (`**head`).

4.2. Видалення вузла на початку списку

Тепер напишемо функцію `pop()`, що видаляє елемент, на який вказує **`head`** і повертає його значення.

Якщо спрямувати вказівник **`head`** на другий елемент, то можна втратити адресу першого, і тоді не можливо буде видалити цей елемент та повернути його значення. Тому необхідно спочатку створити локальну змінну **`prev`**, яка потім буде зберігати адресу першого елемента.

1) Створюємо вказівник **`prev`**, який поки нікуди не посилається (рис. 4.8).

2) Присвоюємо вказівнику **`prev`** адресу першого елемента, на який посилається вказівник **`head`** (рис. 4.9).

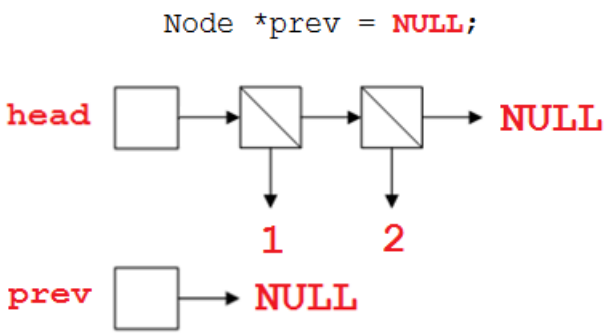


Рисунок 4.8 – Локальна змінна зберігає адресу першого вузла

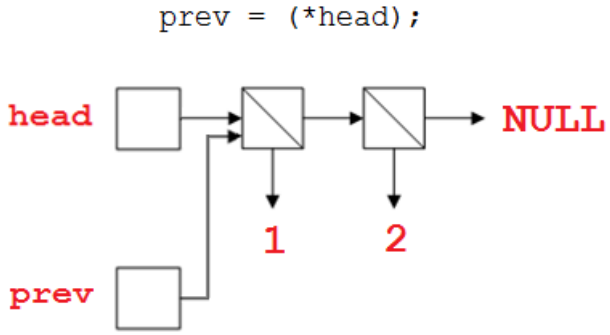


Рисунок 4.9 – Локальна змінна зберігає адресу першого вузла

3) Фіксуємо значення **`val`** першого елемента списку, яке потім буде повернуте функцією `pop()`, та спрямовуємо вказівник на голову списку на другий елемент списку (рис. 4.10).

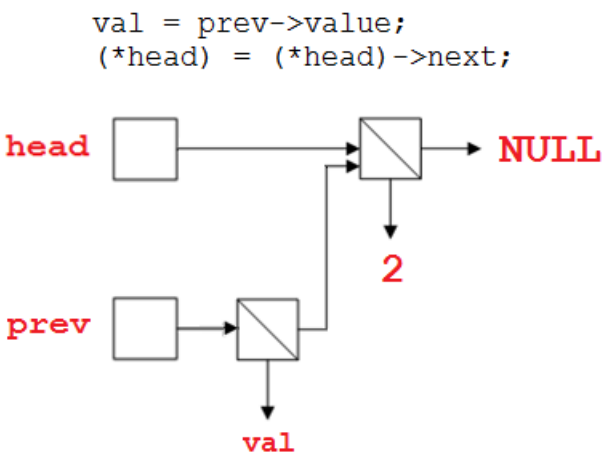


Рисунок 4.10 – Локальна змінна зберігає адресу першого вузла

4) Видаляємо перший елемент за допомогою функції `free()` і повертаємо його значення **`val`** до головної функції (рис. 4.11).

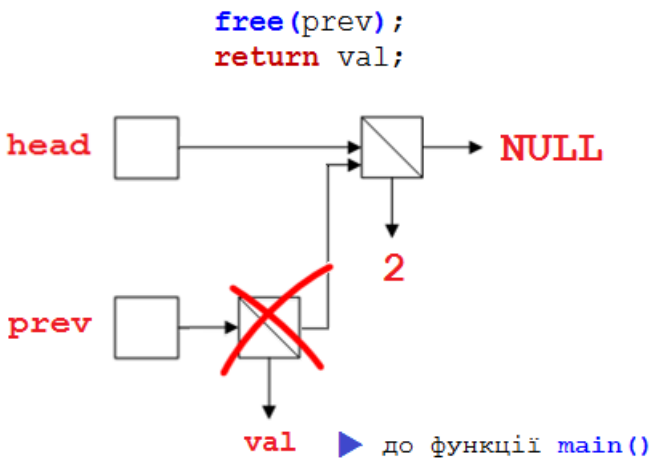


Рисунок 4.11 – Спрямовуємо вказівник **`head`** на наступний елемент і видаляємо перший вузол

Таким чином, текст функції `pop()` для видалення вузла на початку списку має такий вигляд:

```
int pop(Node **head)
{
    Node *prev = NULL;
    int val;
    if(head == NULL)
        exit(-1);
    prev = (*head);
    val = prev->value;
    (*head) = (*head)->next;
    free(prev);
    return val;
}
```

Не забуваємо, що необхідно перевірити на `NULL` голову списку.

Таким чином, були реалізовані дві функції `push()` і `pop()`, які дозволяють тепер використовувати зв'язний список в якості стеку.

4.3. Додавання нового елемента наприкінці списку та видалення останнього елемента списку

Перш ніж перейти до додавання нового елемента наприкінці списку та видалення останнього елементу списку, реалізуємо функції `getLast()`, яка по-

вертає вказівник на останній елемент списку, і `getnth()`, яка повертає вказівник на **n**-й елемент списку.

Оскільки нам відома адреса тільки першого елемента, то єдиним способом дійти до **n**-го вузла буде послідовний перебір усіх елементів списку. Для того щоб отримати наступний елемент, треба перейти до нього за допомогою вказівника **next** поточного вузла:

```
Node *getNth(Node *head, int n)
{
    int counter = 0;
    while(counter < n && head)
    {
        head = head->next;
        counter++;
    }
    return head;
}
```

Переходячи до наступного елемента, слід перевіряти його наявність. Цілком можливо, що був вказаний номер, який перевищує розмір списку. Функція поверне в такому випадку `NULL`. Складність цієї операції складає `O(n)`, яка є однією з проблем зв'язного списку.

Для знаходження останнього елемента будемо перебирати один за одним елементи до тих пір, поки вказівник **next** одного з елементів не стане дорівнювати `NULL`:

```
Node *getLast(Node *head)
{
    if(head == NULL)
        return NULL;
    while(head->next)
        head = head->next;
    return head;
}
```

Тепер додамо ще дві функції – `pushBack()`, яка додає новий елемент наприкінці списку, і `popBack()`, яка видаляє останній елемент списку і повертає його значення.

4.3.1. Додавання нового елемента наприкінці списку

Для додавання нового елемента наприкінці списку, спочатку отримуємо

вказівник на останній елемент, потім створюємо новий елемент, присвоюємо йому значення та спрямовуємо вказівник **next** останнього елемента на новий:

```
void pushBack(Node *head, int value)
{
    Node *last = getLast(head);
    Node *tmp = (Node *) malloc(sizeof(Node));
    tmp->value = value;
    tmp->next = NULL;
    last->next = tmp;
}
```

Зв'язний список зберігає адресу тільки наступного елемента. Якщо ми бажаємо видалити останній елемент, то необхідно змінити вказівник **next** передостаннього елемента. Для цього нам знадобиться функція `getLastButOne()`, що повертає вказівник на передостанній елемент:

```
Node *getLastButOne(Node *head)
{
    if(head == NULL)
        exit(-2);
    if(head->next == NULL)
        return NULL;
    while(head->next->next)
        head = head->next;
    return head;
}
```

Функція повинна працювати навіть тоді, коли список складається всього з одного елемента. Тепер є можливість видалити останній елемент:

```
void popBack(Node **head)
{
    Node *lastbn = NULL;
    // Отримали NULL
    if(!head)
        exit(-1);
    // Список пустий
    if(!(*head))
        exit(-1);
    lastbn = getLastButOne(*head);
    // Якщо у списку один елемент
    if(lastbn == NULL)
    {
        free(*head);
        *head = NULL;
    }
    else
    {
```

```
        free(lastbn->next);
        lastbn->next = NULL;
    }
}
```

Видалення останнього елемента та додавання елемента наприкінці списку мають складність `O(n)`.

Можна написати алгоритм дещо простіше. Для цього будемо використовувати два вказівника. Один – на поточний вузол, другий – на попередній. Тоді можна обійтись без виклику функції `getLastButOne()`:

```
int popBack(Node **head)
{
    Node *pFwd = NULL;        // поточний вузол
    Node *pBwd = NULL;        // попередній вузол
    // Отримали NULL
    if(!head)
        exit(-1);
    // Список пустий
    if(!(*head))
        exit(-1);
    pFwd = *head;
    while(pFwd->next)
    {
        pBwd = pFwd;
        pFwd = pFwd->next;
    }
    if(pBwd == NULL)
    {
        free(*head);
        *head = NULL;
    }
    else
    {
        free(pFwd->next);
        pBwd->next = NULL;
    }
    return pFwd->value;
}
```

Тепер напишемо функцію `insert()`, яка додає на **`n`**-е місце новий вузол. Для цього, спочатку треба буде перейти до необхідного вузла, потім створити новий елемент і поміняти місцями вказівники. Якщо ми вставляємо вузол наприкінці списку, то вказівник **`next`** нового вузла буде вказувати на `NULL`, інакше на такий елемент:

```
void insert(Node *head, unsigned n, int val)
{
    unsigned i = 0;
    Node *tmp = NULL;
    // Знаходимо потрібний елемент.
    // Якщо вийшли за межі списку, то виходимо з циклу
    while(i < n && head->next)
    {
        head = head->next;
        i++;
    }
    tmp = (Node *) malloc(sizeof(Node));
    tmp->value = val;
    // Якщо це не останній елемент,
    // то next спрямовуємо на наступний вузол
    if(head->next)
        tmp->next = head->next;
    else   // інакше на NULL
        tmp->next = NULL;
    head->next = tmp;
}
```

Покажемо візуально послідовність дій для додавання нового вузла у визначене місце, виходячи з того, що це місце вже знайдено і на нього посилається вказівник **head**.

1) Створюємо новий вузол і присвоюємо йому значення **val**, що передається до функції `insert()` в якості параметру (рис. 4.12).

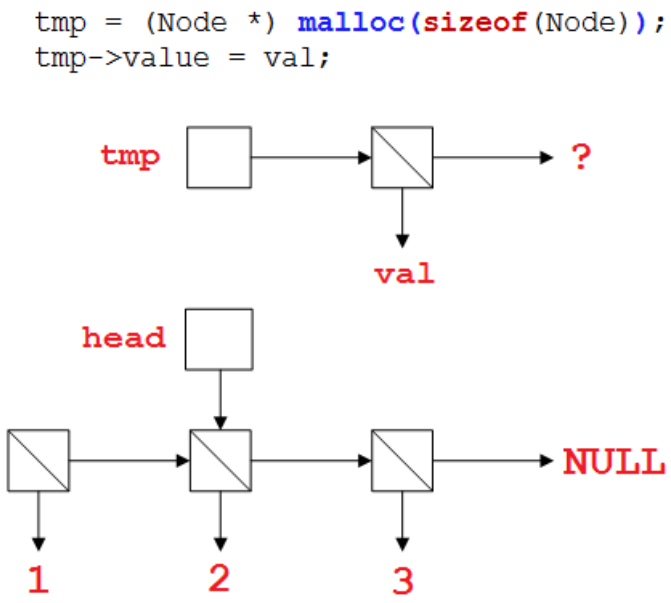


Рисунок 4.12 – Створення нового вузла та присвоювання йому значення, що передається до функції `insert()` в якості параметру

2) Робимо так, щоб новий елемент списку посилався на наступний елемент після **n**-го (рис. 4.13).

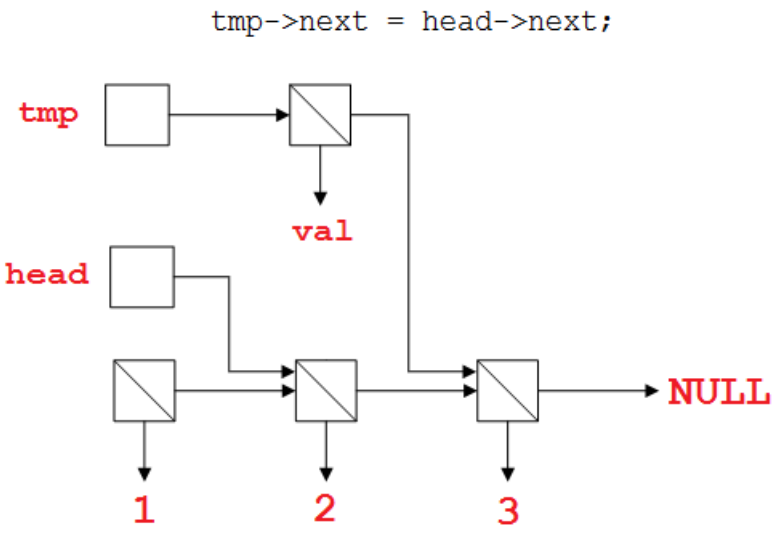


Рисунок 4.13 – Новий елемент списку посилається на наступний після *n*-го

Тепер значення **next** нового вузла зберігає адресу того ж самого вузла, що й елемент, на який посилається **head**.

3) Спрямовуємо вказівник **next** **n**-го елемента на щойно створений вузол (рис. 4.14).

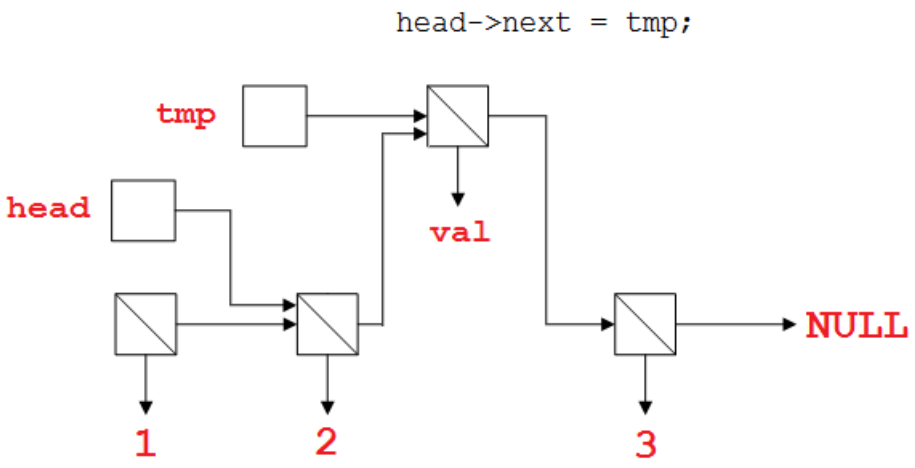


Рисунок 4.14 – Спрямування вказівника `next` *n*-го елемента на щойно створений вузол

Тепер вузол, адресу якого зберігає **head**, вказує на новий вузол **tmp**.

4.3.2. Видалення останнього елемента списку

Функція видалення останнього елемента списку схожа на функцію додавання елемента. Спочатку отримуємо вказівник на елемент, який знаходиться до того, що видаляється, і спрямовуємо посилання на наступний елемент, який знаходиться за тим, що видаляється. Потім видаляємо елемент:

```
int deleteNth(Node **head, int n)
{
    if(n == 0)
        return pop(head);
    else
    {
        Node *prev = getNth(*head, n-1);
        Node *elm  = prev->next;
        int val = elm->value;
        prev->next = elm->next;
        free(elm);
        return val;
    }
}
```

Розглянемо візуально процес видалення останнього елемента списку.

1) Спочатку знаходимо адреси елементу, що видаляється, і того, що знаходиться перед ним (рис. 4.15).

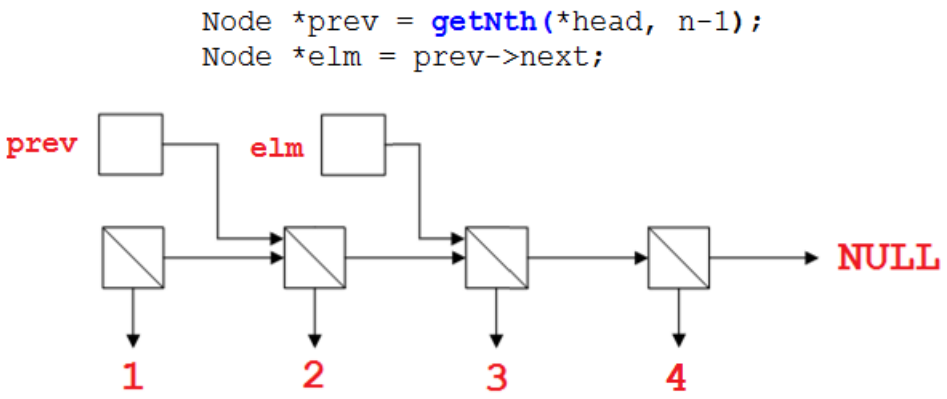


Рисунок 4.15 – Знаходження адрес елементу, що видаляється, і того, що знаходиться перед ним

Для видалення вузла, на який посилається **elm**, потрібен попередній вузол, адресу якого зберігає **prev**.

2) Спрямовуємо вказівник **next** далі, а сам елемент видаляємо (рис. 4.16).

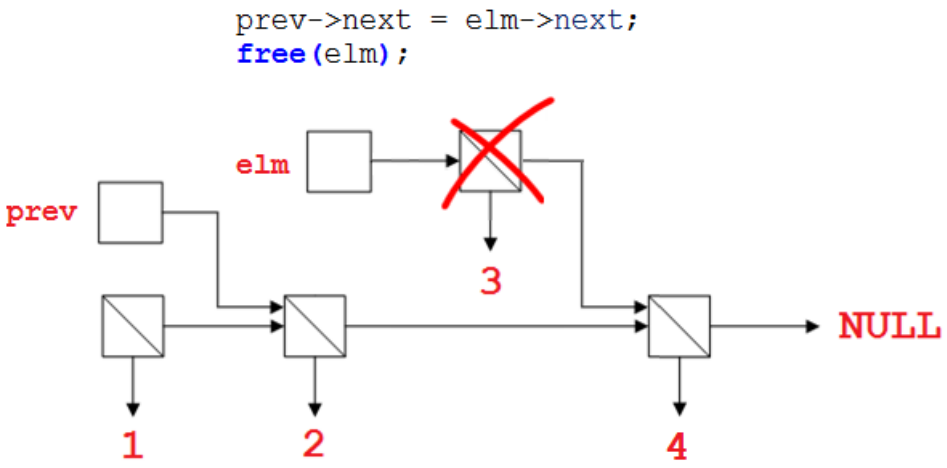


Рисунок 4.16 – Спрямовуємо вказівник на наступний за видаленим вузол і звільняємо пам'ять за допомогою функції `free()`

Крім функції створення списку, необхідно передбачити функцію для його видалення. Оскільки найшвидша функція у нас це `pop()`, то для видалення будемо послідовно видаляти елементи зі списку:

```
void deleteList(Node **head)
{
    while((*head)->next)
    {
        pop(head);
        *head = (*head)->next;
    }
    free(*head);
}
```

Виклик `pop()` можна замінити на тіло функції і прибрати непотрібні перевірки та повернення значення:

```
void deleteList(Node **head)
{
    Node *prev = NULL;
    while((*head)->next)
    {
        prev = (*head);
        (*head) = (*head)->next;
        free(prev);
    }
    free(*head);
}
```

Залишилося написати декілька допоміжних функцій, які спростять і прискорять роботу.

1) `fromArray()` – створення списку з елементів масиву.

Оскільки функція `push()` має мінімальну складність, то додавати елементи будемо саме за її допомогою. Через те, що додавання буде відбуватися у зворотному напрямку, масив будемо обходити від кінця до початку:

```
void fromArray(Node **head, int *arr, size_t size)
{
    size_t i = size - 1;
    if(arr == NULL || size == 0)
        return;
    do
    {
        push(head, arr[i]);
    } while(i-- != 0);
}
```

2) `toArray()` – повернення масиву елементів, що зберігаються у списку.

Оскільки ми будемо створювати масив динамічно, то спочатку визначимо його розмір, а тільки потім занесемо туди значення:

```
int *toArray(const Node *head)
{
    int leng = strlen(head);
    int *values = (int *) malloc(leng * sizeof(int));
    while(head)
    {
        values[--leng] = head->value;
        head = head->next;
    }
    return values;
}
```

3) `printLinkedList()` – вивід на екран вмісту списку.

```
void printLinkedList(const Node *head)
{
    while(head)
    {
        printf("%4d", head->value);
        head = head->next;
    }
    printf("\n");
}
```

Тепер можна провести перевірку та переглянути як працює зв'язний список. Текст програми має такий вигляд:

```
int main(void)
{
    Node *head = NULL;
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    // Створюємо список з масиву
    fromArray(&head, arr, 10);
    printLinkedList(head);
    // Додаємо вузол зі значенням 55 після 5-го елемента
    // (стане шостим)
    insert(head, 4, 55);
    printLinkedList(head);
    pushBack(head, 11);
    pushBack(head, 12);
    pushBack(head, 13);
    pushBack(head, 14);
    printLinkedList(head);
    printf("%d\n", pop(&head));
    printf("%d\n", popBack(&head));
    printLinkedList(head);

    // Видаляємо п'ятий елемент списку
    deleteNth(&head, 4);
    printLinkedList(head);
    deleteList(&head);
    return 0;
}
```

Повний текст програми з результатами роботи наведено в додатку З.

Завдання до практичної роботи:

1. Опрацюйте матеріал лекцій «Зв'язні списки» і «Абстрактні типи даних», а також матеріал теоретичної частини даної практичної роботи.

2. Напишіть відповідно до вашого варіанту (див. табл. 4.1) *C*-програму, яка виконує наступні дії:

– створює в циклі зв'язний список, записує в нього **k** випадкових чисел визначеного типу в діапазоні від **a** до **b** та виводить його на екран;

– додає число **m** після **n**-го елемента списку та виводить результат на екран;

– додає наприкінці списку **t** випадкових чисел в діапазоні від t_1 до t_2 та виводить результат на екран;

– виводить на екран окремо голову та хвіст списку;

– виводить список після видалення голови та хвоста;

– видаляє **s**-й елемент списку та виводить результат на екран.

Таблиця 4.1 – Вхідні дані для виконання практичної роботи

Варіант	**k**	Тип	**a**	**b**	**m**	**n**	**t**	**t_1**	**t_2**	**s**
1	12	**int**	34	79	50	8	3	51	84	8
2	13	**float**	57.03	77.66	65.27	4	4	28.75	58.69	6
3	10	**int**	22	77	44	7	5	11	33	11
4	11	**float**	33.76	58.78	48.12	5	4	13.76	32.98	6
5	12	**int**	11	53	42	9	3	62	76	12
6	13	**float**	15.98	34.09	22.67	4	5	24.62	63.81	7
7	7	**int**	16	43	35	2	5	45	88	7
8	10	**float**	22.33	45.69	33.43	5	4	32.11	36.65	10
9	9	**int**	62	80	71	5	9	62	75	8
10	8	**float**	16.11	34.65	27.92	3	7	42.28	71.59	11
11	14	**int**	22	77	57	3	4	23	45	4
12	10	**float**	54.22	88.23	67.87	6	7	31.43	65.38	2
13	9	**int**	46	99	77	5	8	41	78	7
14	8	**float**	45.48	67.38	63.98	3	4	18.99	51.27	4
15	10	**int**	72	87	82	7	6	37	91	7
16	11	**float**	32.16	48.50	35.45	2	5	61.02	71.81	4
17	14	**int**	22	88	77	9	4	11	34	6
18	15	**float**	23.15	54.21	33.27	3	5	22.13	45.77	5
19	16	**int**	25	65	45	5	6	15	42	9
20	12	**float**	24.53	67.22	33.22	8	7	35.36	56.23	10

Практична робота №5
РОБОТА ЗІ СТЕКОМ І ЧЕРГОЮ

Мета роботи: отримати навички роботи зі стеком і чергою; навчитися використовувати основні операції роботи зі стеком; навчитися використовувати основні операції роботи з чергою; розглянути різні варіанти використання стеку та черги на конкретних прикладах.

Теоретична частина

5.1. Реалізація стеку

Стек – упорядкований набір елементів, в якому розміщення нових і видалення існуючих елементів відбувається з одного кінця (вершини) [1–3, 7].

Принцип роботи стеку порівнюється зі стопкою аркушів паперу: щоб взяти другий зверху аркуш, треба зняти верхній.

В стеку реалізується принцип обслуговування LIFO (Last Input – First Output – останній ввійшов – перший вийшов).

Розрізняють апаратний і програмний стеки.

Апаратний стек використовується для зберігання адрес повернення з функцій та їх аргументів.

Програмний стек – це модель користувача (структура даних).

5.1.1. Функції для роботи зі стеком

Для роботи зі стеком необхідно реалізувати наступні функції:

– `init()` – ініціалізація стеку;

– `push()` – додавання елемента до стеку;

– `pop()` – видалення елементу зі стеку;

– `stkTop()` – отримання верхнього елемента стеку без його видалення;

– `getCount()` – отримання кількості елементів у стеку;

– `isEmpty()` – визначення того, чи є стек пустим (повертає 1, якщо стек пустий і 0 – у протилежному випадку);

– `stkPrint()` – вивід елементів стеку на екран.

5.1.2. Способи реалізації стеку

Існує декілька способів реалізації стеку, серед яких використовують реалізації за допомогою одновимірного масиву та зв’язного списку.

Розглянемо спосіб реалізації стеку за допомогою одновимірного масиву.

Стек для зберігання 30 дійсних чисел типу **`float`** реалізуємо у вигляді наступної структури:

```
#define NMAX 30
struct stack
{
    float elem[NMAX];
    int top;
};
```

В даному визначенні стеку:

– `NMAX` – максимальна кількість елементів у стеку;

– `elem` – масив з `NMAX` чисел типу **`float`**, який призначений для зберігання елементів стеку;

– `top` – індекс елемента, що знаходиться у вершині стеку.

За допомогою засобу **`typedef`** визначимо новий тип **`Stack`**, який будемо використовувати далі замість запису **`struct`** `stack`:

```
typedef struct stack Stack;
```

5.1.3. Ініціалізація стеку

Для ініціалізації стеку будемо використовувати функцію `init()`, яка встановлює індекс елемента, що знаходиться у вершині стеку, в 0:

```
void init(Stack *stk)
{
    stk->top = 0;
}
```

5.1.4. Додавання елемента до стеку

Додавання елемента до стеку будемо здійснювати за допомогою функції `push()` наступним чином: до елемента масиву з індексом `stk->top` записується значення **`val`** типу **`float`**. Після цього вершина стеку, що відповідає кількості елементів у масиві, зміщується на 1 елемент (`stk->top++`).

```
void push(Stack *stk, float val)
{
    if(stk->top < NMAX)
    {
        stk->elem[stk->top] = val;
        stk->top++;
    }
    else
        printf("Стек повний, кількість елементів: %d!\n",
               stk->top);
}
```

5.1.5. Видалення елемента зі стеку

Видалення елемента зі стеку будемо здійснювати за допомогою функції `pop()`. Якщо в масиві, що реалізує стек, є елементи, то кількість елементів зменшується на 1. Після цього повертається останній елемент, тобто той, що знаходиться на вершині стеку:

```
float pop(Stack *stk)
{
    float elem;

    if(stk->top > 0)
    {
        stk->top--;
        elem = stk->elem[stk->top];
        return elem;
    }
    else
    {
        printf("Стек пустий!\n");
        return 0;
    }
}
```

5.1.6. Отримання верхнього елемента стеку без його видалення

Для отримання верхнього елемента стеку без його видалення будемо застосовувати функцію `stkTop()`:

```
float stkTop(Stack *stk)
{
    if(stk->top > 0)
        return stk->elem[stk->top-1];
    else
    {
        printf("Стек пустий!\n");
        return 0;
    }
}
```

5.1.7. Отримання кількості елементів у стеку

Для отримання кількості елементів у стеку скористаємося функцією `getCount()`, яка повертає значення індексу останнього елемента масиву:

```
int getCount(Stack *stk)
{
    return stk->top;
}
```

5.1.8. Визначення факту того, чи є стек пустим

Для з'ясування питання про те, пустий стек чи ні, скористаємося функцією `isEmpty()`. Якщо кількість елементів у стеку дорівнює 0, то стек пустий – повертається 1, якщо в ньому вже є елементи – повертається 0.

```
int isEempty(Stack *stk)
{
    if(stk->top == 0)
        return 1;
    else
        return 0;
}
```

5.1.9. Вивід елементів стеку

Вивід елементів стеку будемо здійснювати за допомогою функції `stkPrint()`. Якщо стек не пустий, рухаємося від останнього елемента до початку масиву з одночасним виводом елементів на екран:

```
void stkPrint(Stack *stk)
{
    int i;
    i = stk->top;           // i – кількість елементів у стеку
    if(isEmpty(stk) == 1)
        return;             // стек пустий
    do
    {
        i--;
        printf("%5.2f\n", stk->elem[i]);
    } while(i > 0);
}
```

Приклад №1. Створимо стек з **n** елементів та витягнемо їх зі стеку. Кількість елементів будемо вводити з клавіатури.

Текст *C*-програми для реалізації даної задачі має такий вигляд:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#define NMAX 100
typedef struct stack
{
    float elem[NMAX];
    int top;
} Stack;
void  init(Stack *stk);
void  push(Stack *stk, float val);
```

```
float pop(Stack *stk);
int   getCount(Stack *stk);
void  stkPrint(Stack *stk);
float stkTop(Stack *stk);
int   isEmpty(Stack *stk);
int main(void)
{
    Stack *stk;
    int i, n;
    float elem;
    SetConsoleOutputCP(1251);
    stk = (Stack *) malloc(sizeof(Stack));
    init(stk);
    printf("========================================\n");
    printf("Введіть розмірність стеку:   ");
    scanf("%d", &n);
    printf("========================================\n");
    for(i = 0; i < n; i++)
    {
        printf("Введіть елемент %d: ", i);
        scanf("%f", &elem);
        push(stk, elem);
    }
    printf("========================================\n");
    printf("Кількість елементів у стеку: %d\n",
           getCount(stk));
    printf("========================================\n");
    printf("Виводимо елементи стеку                 \n");
    printf("========================================\n");
    stkPrint(stk);
    printf("========================================\n");
    printf("Вершина стеку:     %5.2f\n", stkTop(stk));
    printf("========================================\n");
    do
    {
        printf("Витягаємо елемент  %5.2f\n", pop(stk));
        printf("========================================\n");
        printf("Кількість елементів у стеку: %d\n",
               getCount(stk));
        printf("========================================\n");
    } while(isEmpty(stk) == 0);
    return 0;
}

void init(Stack *stk)
{
    stk->top = 0;
}

void push(Stack *stk, float val)
{
    if(stk->top < NMAX)
    {
```

```
        stk->elem[stk->top] = val;
        stk->top++;
    }
    else
    {
        printf("=======================================\n");
        printf("Стек повний, кількість елементів: %d!\n",
               stk->top);
    }
}
int getCount(Stack *stk)
{
    return stk->top;
}
void stkPrint(Stack *stk)
{
    int i;                  // i – кількість елементів у стеку
    i = stk->top;
    if(isEmpty(stk) == 1)
        return;             // стек пустий
    do
    {
        i--;
        printf("%5.2f\n", stk->elem[i]);
    } while(i>0);
}
float stkTop(Stack *stk)
{
    if(stk->top > 0)
        return stk->elem[stk->top-1];
    else
    {
        printf("Стек пустий!\n");
        return 0;
    }
}
float pop(Stack *stk)
{
    float elem;
    if(stk->top > 0)
    {
        stk->top--;
        elem = stk->elem[stk->top];
        return elem;
    }
    else
    {
        printf("Стек пустий!\n");
        return 0;
    }
}
```

```
int isEmpty(Stack *stk)
{
    if(stk->top == 0)
        return 1;
    else
        return 0;
}
```

Результат роботи програми наведено на рис. 5.1.

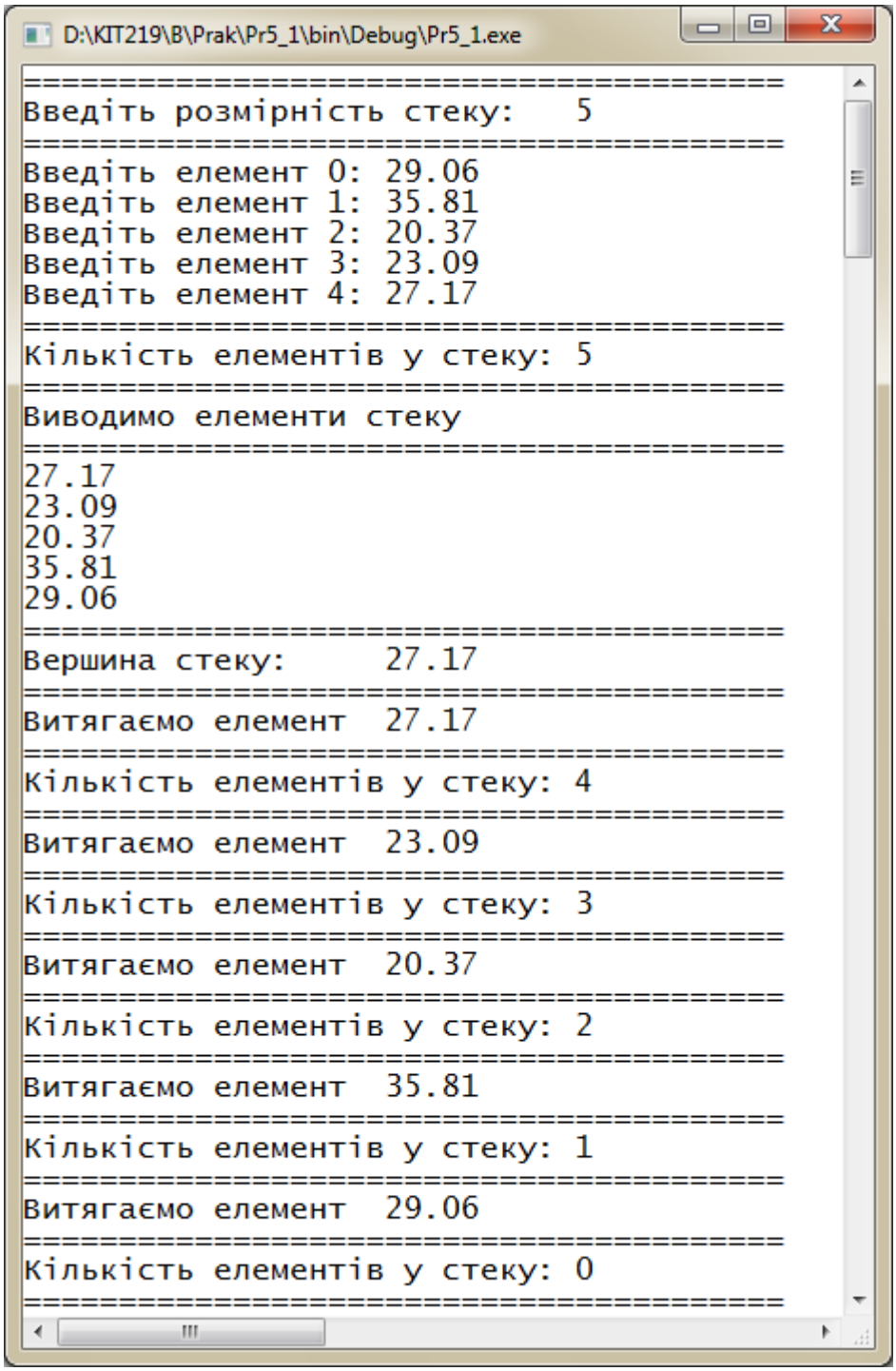


Рисунок 5.1 – Результат виконання програми для демонстрації роботи зі стеком

Приклад №2. Переведемо введене ціле десяткове число у систему числення з заданою основою.

На рис. 5.2 наведено приклад переведення числа 26 з десяткової системи числення у двійкову.

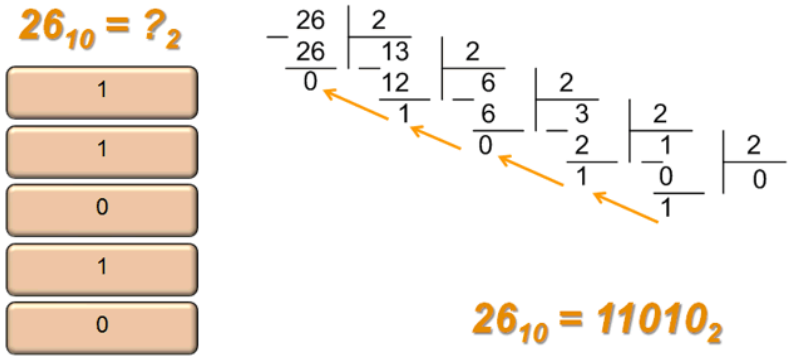


Рисунок 5.2 – Ділення числа на основу системи числення відбувається до тих пір, поки частка не буде дорівнювати 0

Текст програми, яка здійснює переведення цілого числа з десяткової у двійкову, вісімкову та шістнадцяткову системи числення має такий вигляд:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#define NMAX 30
typedef struct stack
{
    unsigned elem[NMAX];
    int top;
} Stack;
void init(Stack *stk);
void push(Stack *stk, unsigned val);
unsigned pop(Stack *stk);
int isEmpty(Stack *stk);
int main(void)
{
    Stack *stk;
    int n, value;
    SetConsoleOutputCP(1251);
    stk = (Stack *) malloc(sizeof(Stack));
    init(stk);
    printf("============================\n");
    printf("Введіть число:   ");
    scanf("%d", &value);
    printf("============================\n");
    printf("Введіть основу:  ");
```

```
    scanf("%d", &n);
    printf("============================\n\n");
    do {
        push(stk, value % n);
        value = value / n;
    } while(value > 0);
    printf("============================\n");
    printf("Результат роботи програми    \n");
    printf("============================\n");
    do
    {
        // виводимо цифри (включаючи цифри A...F)
        printf("%X", pop(stk));
    } while(isEmpty(stk) == 0);
    printf("\n============================\n\n");
    return 0;
}
void init(Stack *stk)
{
    stk->top = 0;
}
void push(Stack *stk, unsigned val)
{
    if(stk->top < NMAX)
    {
        stk->elem[stk->top] = val;
        stk->top++;
    }
    else
    {
        printf("======================================\n");
        printf("Стек повний, кількість елементів: %d!\n",
               stk->top);
    }
}
unsigned pop(Stack *stk)
{
    unsigned elem;
    if(stk->top > 0)
    {
        stk->top--;
        elem = stk->elem[stk->top];
        return elem;
    }
    else
    {
        printf("Стек пустий!\n");
        return 0;
    }
}
int isEmpty(Stack *stk)
{
```

```
    if(stk->top == 0)
        return 1;
    else
        return 0;
}
```

Результат роботи програми для переведення десяткового числа на двійкову систему числення наведено на рис. 5.3.

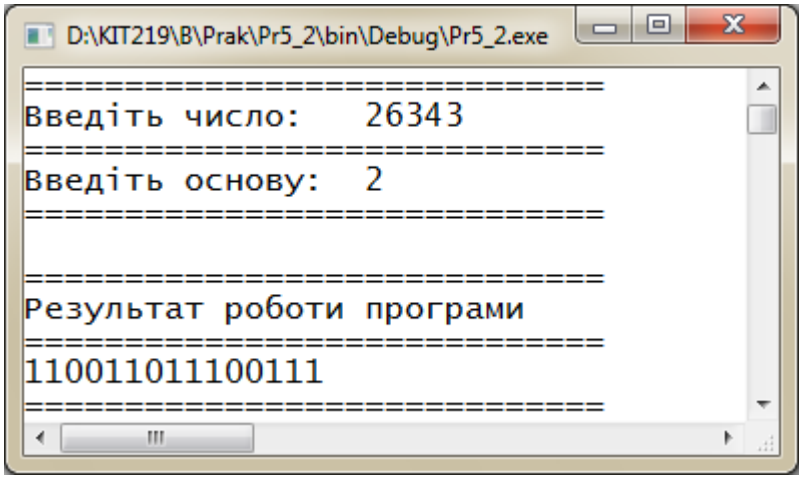


Рисунок 5.3 – Результат роботи програми для переведення десяткового числа у двійкову систему числення

Результат роботи програми для переведення десяткового числа у вісімкову систему числення наведено на рис. 5.4.

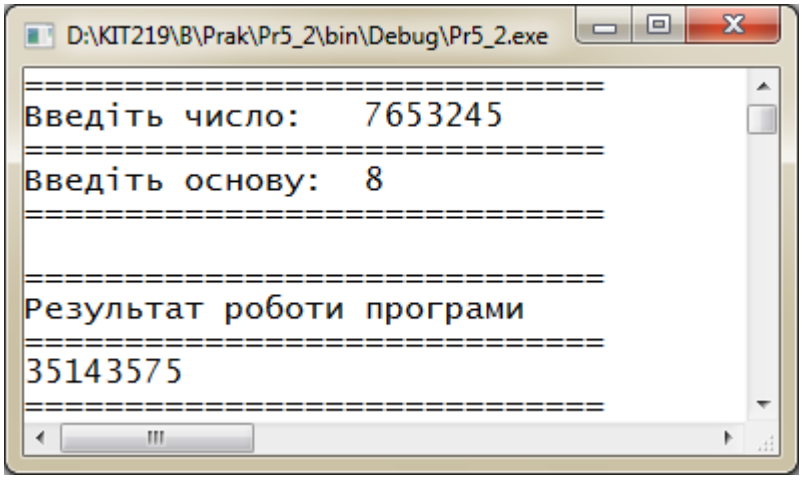


Рисунок 5.4 – Результат роботи програми для переведення десяткового числа у вісімкову систему числення

Результат роботи програми для переведення десяткового числа у шістнадцяткову систему числення наведено на рис. 5.5.

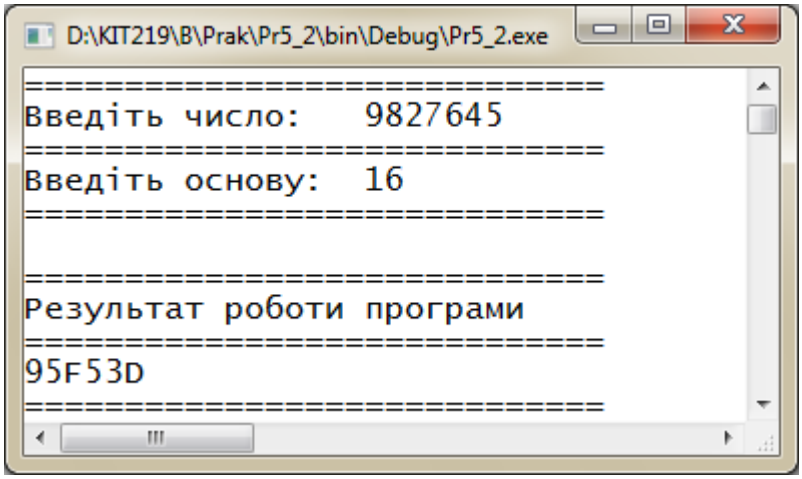


Рисунок 5.5 – Результат роботи програми для переведення десяткового числа у шістнадцяткову систему числення

5.2. Реалізація черги

Черга – упорядкований набір елементів, які можуть видалятися з її початку і додаватися до її кінця (рис. 5.6).

Черга організована за принципом обслуговування FIFO (First Input – First Output – першим прийшов – першим вийшов) [8].

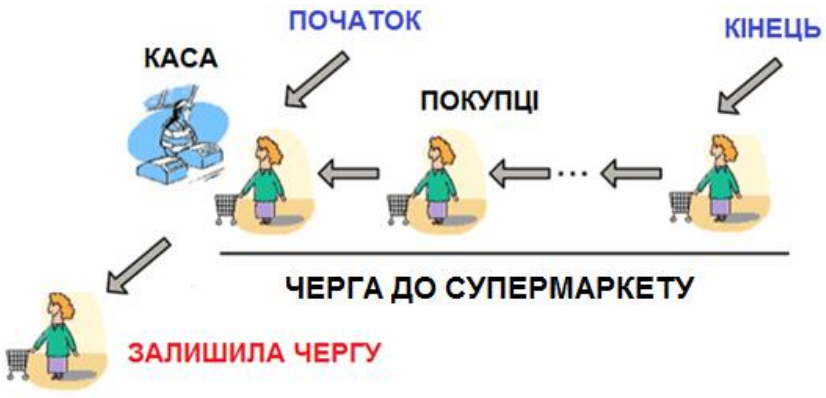


Рисунок 5.6 – Приклад черги

Черга в програмуванні використовується, коли треба виконати будь-які дії в порядку їх надходження, виконавши їх послідовно. Прикладом черги може

бути організація подій в ОС Windows. Коли користувач впливає на роботу працюючої програми, в ній одразу не викликається відповідна функція, оскільки в цей момент програма може здійснювати інші дії. В цьому випадку програмі надсилається повідомлення, що містить інформацію про дію, яка повинна відбутися. Це повідомлення ставиться в чергу, і лише коли будуть оброблені повідомлення, які прийшли раніше, програма виконає необхідну дію.

5.2.1. Функції для роботи з чергою

Для роботи з чергою необхідно реалізувати наступні функції:

– `init()` – ініціалізація черги;

– `insert()` – додавання елемента наприкінці черги;

– `remove()` – видалення елемента з черги;

– `isEmpty()` – визначення того, чи є черга пустою (повертає 1, якщо черга є пустою, і 0 – у протилежному випадку);

– `print()` – вивід елементів черги на екран.

Існує декілька способів реалізації черги, серед яких зазначимо реалізації за допомогою одновимірного масиву та за допомогою зв’язного списку.

5.2.2. Реалізація черги на базі одновимірного масиву

Розглянемо спочатку реалізацію черги на базі одновимірного масиву. Будемо використовувати масив і дві змінні:

– **`front`** – позиція першого елемента в черзі;

– **`rear`** – позиція останнього елемента в черзі.

Спочатку `front = 1` і `rear = 0`.

Черга вважається пустою, якщо `rear < front`.

Число елементів в черзі можна визначити інструкцією

```
n = rear - front + 1;
```

Для реалізації черги на базі масиву будемо використовувати структуру, визначення якої має такий вигляд:

```
#define QMAX 100
```

```
typedef struct queue
{
    int que[QMAX];
    int front;
    int rear;
} Queue;
```

5.2.2.1. Ініціалізація черги

Ініціалізацію черги будемо здійснювати за допомогою функції `init()`, текст якої має такий вигляд:

```
void init(Queue *q)
{
    q->front = 1;
    q->rear  = 0;
    return;
}
```

5.2.2.2. Додавання елемента до черги

Для додавання елемента до черги будемо застосовувати функцію `insert()`, текст якої має такий вигляд:

```
void insert(Queue *q, int val)
{
    if(q->rear < QMAX - 1)
    {
        q->rear++;
        q->que[q->rear] = val;
    }
    else
        printf("Черга заповнена!\n");
    return;
}
```

5.2.2.3. Визначення стану черги

Перевірку факту того, чи є черга пустою, чи ні, будемо здійснювати за допомогою функції `isEmpty()`, текст якої має такий вигляд:

```
int isEmpty(Queue *q)
{
    if(q->rear < q->front)
        return 1;
    else
        return 0;
}
```

5.2.2.4. Вивід елементів черги на екран

Вивід елементів черги на екран будемо здійснювати за допомогою функції `print()`, текст якої має такий вигляд:

```
void print(Queue *q)
{
    int h;
    if(isEmpty(q) == 1)
    {
        printf("Черга є пустою!\n");
        return;
    }
    for(h = q->front; h <= q->rear; h++)
        printf("%d ", q->que[h]);
    return;
}
```

5.2.2.5. Видалення елемента з черги

Видалення елемента з черги будемо здійснювати за допомогою функції `removeQ1()`, текст якої має такий вигляд:

```
int removeQ1(Queue *q)
{
    int x;
    if(isEmpty(q) == 1)
    {
        printf("Черга є пустою!\n");
        return(0);
    }
    x = q->que[q->front];
    q->front++;
    return x;
}
```

Недоліком запропонованої реалізації черги є те, що черга зміщується в бік старших адрес масиву. Це може викликати переповнення.

Для усунення цього недоліку розглянемо інший варіант реалізації функції видалення елемента з черги:

```
int removeQ2(Queue *q)
{
    int x, h;
    if(isEmpty(q) == 1)
    {
         printf("Черга є пустою!\n");
         return 0;
    }
```

```
    x = q->que[q->front];
    for(h = q->front; h < q->rear; h++)
        q->que[h] = q->que[h+1];
    q->rear--;
    return x;
}
```

Приклад №3. Створимо чергу з 8 елементів цілого типу та виведемо її вміст на екран. Потім один за одним видалимо усі елементи черги.

Текст програми, що вирішує поставлену задачу, має такий вигляд:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#define QMAX 8

typedef struct queue
{
    int que[QMAX];
    int front;
    int rear;
} Queue;

void init(Queue *q);
void print(Queue *q);
void insert(Queue *q, int x);
int isEmpty(Queue *q);
int removeQ2(Queue *q);

int main(void)
{
    Queue *q;
    int a;

    SetConsoleOutputCP(1251);

    q = (Queue *) malloc(sizeof(Queue));
    init(q);
    print(q);
    printf("================================\n");

    for(int i = 0; i < QMAX; i++)
    {
         printf("Введіть елемент черги: ");
         scanf("%d", &a);
         insert(q, a);
    }
    printf("================================\n");
    printf("СКЛАД ЧЕРГИ                     \n");
    printf("================================\n");
    print(q);
    printf("\n================================\n");
```

```
    while(q->front <= q->rear)
    {
        a = removeQ2(q);
        printf("Елемент %d видалено\n", a);
        printf("================================\n");
        print(q);
    }
    return 0;
}

void init(Queue *q)
{
    q->front = 1;
    q->rear  = 0;
    return;
}

void print(Queue *q)
{
    int h;
    if(isEmpty(q) == 1)
    {
        printf("ЧЕРГА Є ПУСТОЮ!\n");
        return;
    }
    for(h = q->front; h <= q->rear; h++)
        printf("%d ", q->que[h]);
    printf("\n================================\n");
    return;
}

void insert(Queue *q, int val)
{
    if(q->rear < QMAX - 1)
    {
        q->rear++;
        q->que[q->rear] = val;
    }
    else
    {
        printf("================================\n");
        printf("ЧЕРГА ЗАПОВНЕНА!\n");
        printf("================================\n");
        printf("Елемент %d не був занесений.\n", val);
    }
    return;
}

int isEmpty(Queue *q)
{
    if(q->rear < q->front) return 1;
    else return 0;
}
```

```
int removeQ2(Queue *q)
{
    int x, h;
    if(isEmpty(q) == 1)
    {
         printf("ЧЕРГА Є ПУСТОЮ!\n");
         return 0;
    }
    x = q->que[q->front];
    for(h = q->front; h < q->rear; h++)
        q->que[h] = q->que[h+1];
    q->rear--;
    return x;
}
```

Результат роботи програми наведено на рис. 5.7.

5.2.3. Реалізація черги на базі зв'язного списку

Для реалізації черги на базі зв'язного списку будемо використовувати структури **list** і **queue**:

```
typedef struct list                 // list - для списку
{
    int field;
    struct list *ptr;
} List;

typedef struct queue                // queue - для черги
{
    List *front;
    List *rear;
} Queue;
```

5.2.3.1. Ініціалізація черги

Для ініціалізації черги будемо використовувати функцію `initQueue()`, яка встановлює в нуль вказівники на початок і кінець черги.

Текст функції має такий вигляд:

```
void initQueue(Queue *q)
{
    q->front = 0;
    q->rear  = 0;
}
```

5.2.3.2. Ініціалізація елемента списку

Для ініціалізації елемента списку (вузла) будемо застосовувати функцію `initList()`, яка має такий вигляд:

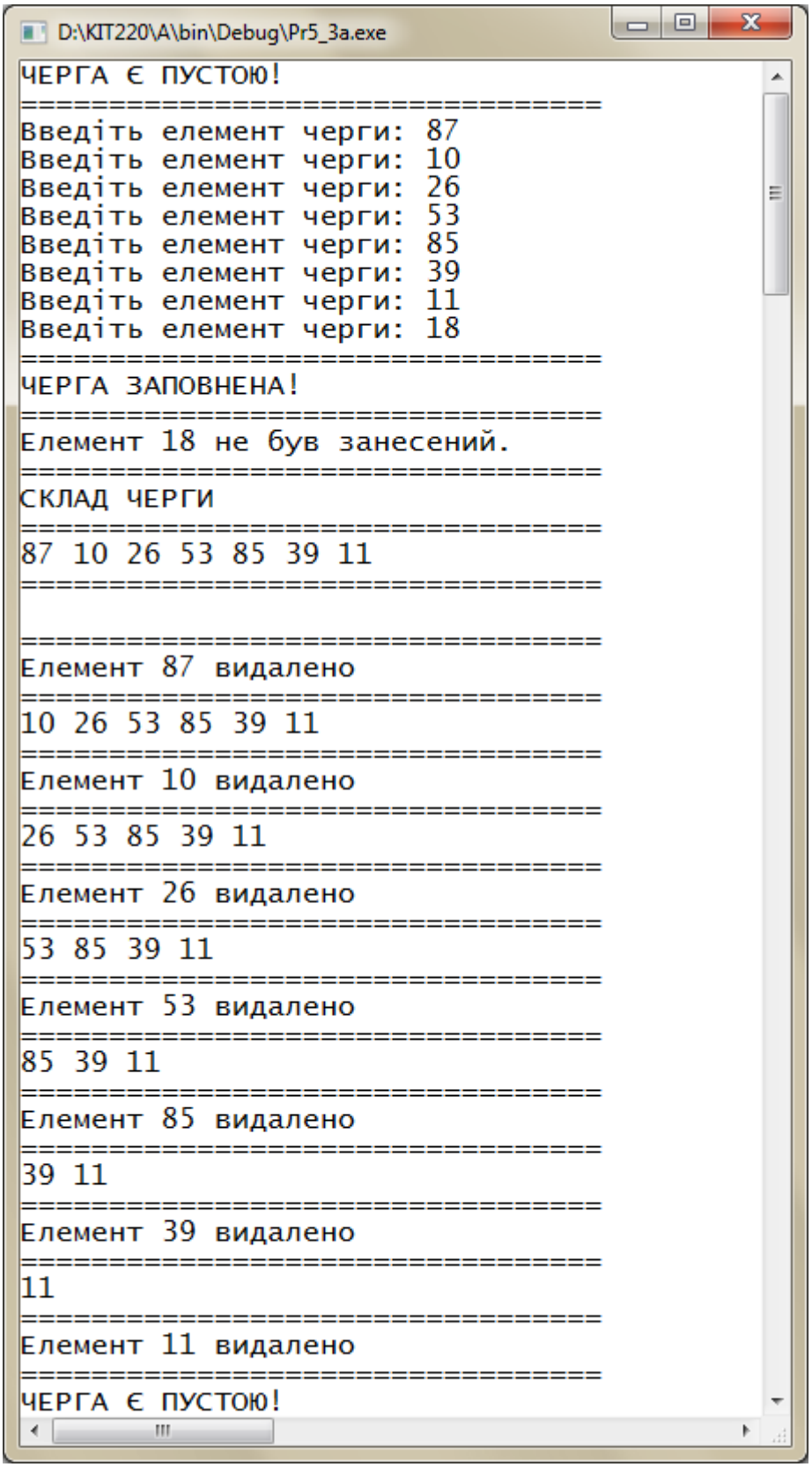


Рисунок 5.7 – Результат роботи програми для реалізації черги на базі одновимірного масиву

```
List *initList(int val)
{
    List *plist;

    plist = (List *) malloc(sizeof(List));
    plist->field = val;
    plist->ptr = NULL;
    return(plist);
}
```

5.2.3.3. Визначення факту того, чи є черга пустою

Перевірку факту того, чи є черга пустою, чи ні, будемо здійснювати за допомогою функції `isEmpty()`:

```
int isEmpty(Queue *q)
{
    if(q->front == 0)
        return 1;
    else
        return 0;
}
```

5.2.3.4. Додавання елемента до черги

Додавання елемента до черги будемо здійснювати за допомогою функцій `insert()` і `addElem()`:

```
void insert(Queue *q, int val)
{
    if((q->rear == 0) && (q->front == 0))
    {
        q->rear  = initList(val);
        q->front = q->rear;
    }
    else
        q->rear = addelem(q->rear, val);
}

List *addElem(List *plist, int number)
{
    List *temp, *p;

    temp = (List *) malloc(sizeof(List));
    p = plist->ptr;
    plist->ptr = temp;
    temp->field = number;
    temp->ptr = p;
    return(temp);
}
```

5.2.3.5. Вивід елементів черги

Вивід елементів черги відбувається за допомогою функції `print()`:

```
void print(Queue *q)
{
    List *h;

    if(isEmpty(q) == 1)
    {
        printf("Черга є пустою!\n");
        return;
    }
    for(h = q->front; h != NULL; h = h->ptr)
        printf("%d ", h->field);
    return;
}
```

5.2.3.6. Видалення елемента з черги

Видалення елемента з черги будемо здійснювати за допомогою функції `removeElem()`:

```
int removeElem(Queue *q)
{
    List *temp;
    int x;

    if(isEmpty(q) == 1)
    {
        printf("Черга є пустою!\n");
        return 0;
    }
    x = q->front->field;
    temp = q->front;
    q->front = q->front->ptr;
    free(temp);
    return x;
}
```

5.2.3.7. Отримання першого елемента черги без його видалення

Отримання першого елемента черги без його видалення будемо здійснювати за допомогою функції `view()`:

```
int view(Queue *q)
{
    int x;

    x = q->front->field;
    return x;
}
```

Приклад №4. Створити чергу з 8 елементів на базі зв'язного лінійного списку, вивести її елементи на екран та по черзі видалити усі елементи.

Текст програми для вирішення поставленої задачі має такий вигляд:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#define QMAX 8
typedef struct list
{
    int field;
    struct list *ptr;
} List;

typedef struct queue
{
    List *front;
    List *rear;
} Queue;
void  initQueue(Queue *q);
void  print(Queue *q);
void  insert(Queue *q, int x);
List *addElem(List *lst, int number);
int   removeElem(Queue *q);
int   isEmpty(Queue *q);
List *initList(int val);
int main(void)
{
    Queue *q;
    int elem;
    SetConsoleOutputCP(1251);
    q = (Queue *) malloc(sizeof(Queue));
    initQueue(q);
    print(q);
    printf("================================\n");
    for(int i = 0; i < QMAX; i++)
    {
         printf("Введіть елемент черги: ");
         scanf("%d", &elem);
         insert(q, elem);
    }
    printf("================================\n");
    printf("СКЛАД ЧЕРГИ                     \n");
    printf("================================\n");
    print(q);
    printf("\n================================\n");
    while(q->front != NULL)
    {
        elem = removeElem(q);
        printf("Елемент %d видалено\n", elem);
        printf("================================\n");
```

```
        print(q);
    }
    return 0;
}
List *initList(int val)
{
    List *plist;
    plist = (List *) malloc(sizeof(List));
    plist->field = val;
    plist->ptr = NULL;
    return(plist);
}
void initQueue(Queue *q)
{
    q->front = 0;
    q->rear  = 0;
}
List *addElem(List *plist, int number)
{
    List *temp, *p;
    temp = (List *) malloc(sizeof(List));
    p = plist->ptr;
    plist->ptr = temp;
    temp->field = number;
    temp->ptr = p;
    return(temp);
}
void print(Queue *q)
{
    List *h;
    if(isEmpty(q) == 1)
    {
        printf("ЧЕРГА Є ПУСТОЮ!\n");
        return;
    }
    for(h = q->front; h != NULL; h = h->ptr)
        printf("%d ", h->field);
    printf("\n================================\n");
    return;
}
void insert(Queue *q, int val)
{
    if((q->rear == 0) && (q->front == 0))
    {
        q->rear  = initList(val);
        q->front = q->rear;
    }
    else q->rear = addElem(q->rear, val);
}
int removeElem(Queue *q)
{
    List *temp;
```

```
    int x;
    if(isEmpty(q) == 1)
    {
        printf("ЧЕРГА Є ПУСТОЮ!\n");
        return 0;
    }
    x = q->front->field;
    temp = q->front;
    q->front = q->front->ptr;
    free(temp);
    return x;
}
int isEmpty(Queue *q)
{
    if(q->front == 0)
        return 1;
    else
        return 0;
}
```

Результат виконання програми для реалізації черги на базі зв'язного списку наведено на рис. 5.8.

Завдання до практичної роботи:

1. Опрацюйте матеріал лекцій «Стек» і «Черги», а також теоретичну частину даної практичної роботи.

2. Напишіть *C*-програму, яка демонструє роботу зі стеками на базі одновимірних масивів і виконує наступні дії:

– заповнює стеки **stack1** і **stack2** випадковими цілими числами в діапазоні від **a** до **b** (див. табл. 5.1) за принципом: до стеку **stack1** заносяться парні числа, а до стеку **stack2** – непарні (загальна кількість чисел дорівнює **k**);

– виводить вміст стеків так, як показано на рис. 5.9.

– урівнює кількість елементів в обох стеках шляхом витягання елементів зі стеку з більшою кількістю елементів та виводить на екран (рис. 5.10) вміст трьох стеків, де вміст стеку **stack3** визначається попарними добутками елементів, що знаходяться в стеках **stack1** і **stack2** з однаковим індексом.

3. Напишіть *C*-програму, яка демонструє роботу з чергою на базі зв'язного списку і стеками на базі масивів та виконує наступні дії:

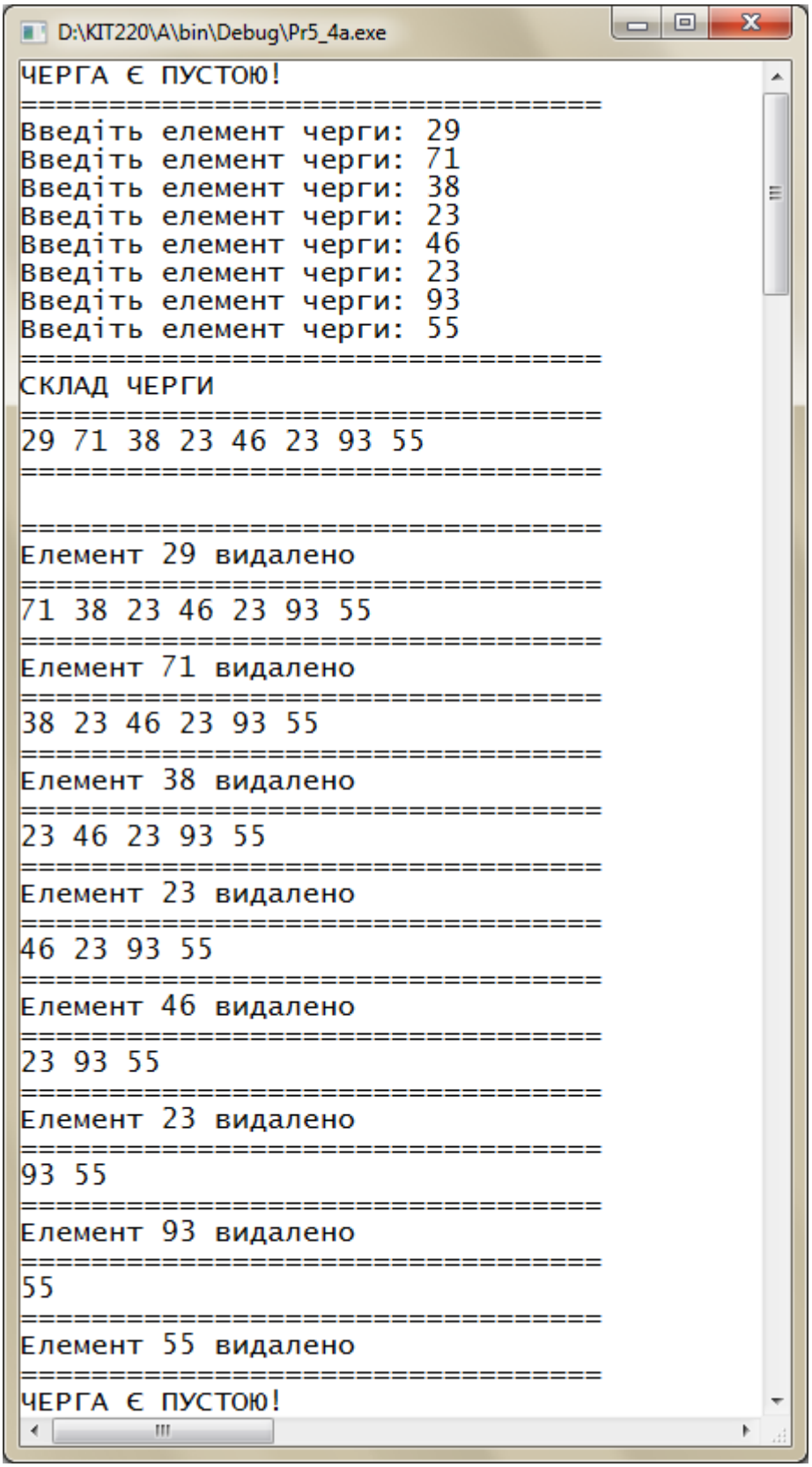


Рисунок 5.8 – Результат роботи програми для реалізації черги на базі зв’язного списку

Таблиця 5.1 – Дані для виконання завдань

Варіант	Одновимірний масив			Зв’язний список			
	k	**a**	**b**	**m**	**d**	**e**	**n**
1	18	39	91	10	31	72	52
2	17	14	48	11	23	65	44
3	16	62	87	12	43	95	69
4	15	65	98	13	27	54	40
5	19	24	37	11	22	87	54
6	15	32	69	10	27	34	30
7	20	37	87	19	83	99	91
8	19	53	76	18	53	85	69
9	18	24	37	17	21	63	42
10	15	43	86	19	16	95	54
11	17	27	49	13	42	95	68
12	16	13	49	14	35	76	55
13	20	35	67	19	26	73	50
14	19	33	73	18	11	23	17
15	20	11	65	10	17	55	36
16	18	43	67	12	14	33	23
17	14	22	65	13	11	68	33
18	15	26	54	14	17	56	34
19	17	55	76	15	22	76	54
20	18	32	74	16	36	65	46

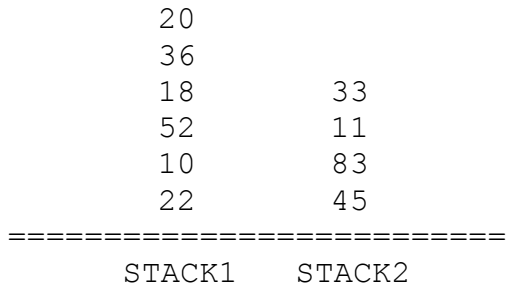


Рисунок 5.9 – Приклад виводу на екран вмісту стеків **stack1** і **stack2**

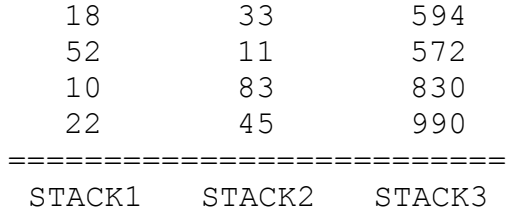


Рисунок 5.10 – Приклад виводу на екран вмісту всіх стеків

– заповнює чергу випадковими цілими числами в діапазоні від **d** до **e** (загальна кількість чисел дорівнює **m**);

– виводить на екран вміст черги, як показано на рис. 5.11;

```
========================================
ВМІСТ ЧЕРГИ
========================================
45 11 22 90 53 87 14 36 72 41 22 30 46 23
========================================
```

Рисунок 5.11 – Приклад виводу на екран вмісту черги

– заповнює стеки **stack1** і **stack2** елементами черги за принципом: числа, які менші **n**, заносяться до стеку **stack1**, інші заносяться до стеку **stack2**;

– виводить вміст стеків так, як показано на рис. 5.12.

```
                46
                41
        23      72
        30      36
        22      87
        14      53
        22      90
        11      45
=========================
    STACK1   STACK2
```

Рисунок 5.12 – Приклад виводу на екран вмісту стеків, якщо n дорівнює 35

Практична робота №6
РЕАЛІЗАЦІЯ ДЕКУ НА ОСНОВІ МАСИВУ

Мета роботи: отримати навички роботи зі створення та видалення деку, а також зміни його розміру у разі необхідності; відпрацювати основні операції роботи з деком: додавання елементів до деку, видалення елементів з деку, отримання інформації про стан деку, отримання крайніх елементів без видалення їх з деку; навчитися здійснювати перебір елементів деку за допомогою ітератора.

Теоретична частина

Дек (від англ. Deq – double ended queue, тобто черга з двома кінцями) – це послідовний список, в якому додавання та видалення елементів може здійснюватися з будь-якого кінця списку з двох можливих. Окремий випадок деку – дек з обмеженим входом і дек з обмеженим виходом. Логічна та фізична структури деку схожі на логічну та фізичну структури кільцевої FIFO-черги [1–3, 9].

Основними операціями над деком є:

– ініціалізація списку;

– додавання вузла до списку;

– видалення вузла зі списку;

– видалення голови та хвоста списку;

– вивід елементів списку у прямому та зворотному порядку.

Фізична структура деку в статичній пам’яті ідентична до структури кільцевої черги. Динамічна реалізація є очевидним об’єднанням стеку та черги.

Завдання, що потребують застосування структури деку, зустрічаються в обчислювальній техніці та програмуванні набагато рідше, ніж завдання, які реалізуються на базі стеку або черги. Як правило, вся організація деку виконується програмістом без будь-яких спеціальних засобів системної підтримки.

Прикладом деку може бути будь-який термінал, в якому вводяться команди, кожна з яких виконується певний час. Якщо ввести команду, не дочекавшись, поки закінчиться виконання попередньої команди, то вона встане в чергу і почне виконуватися тільки тоді, коли звільниться термінал. Це FIFO черга. Якщо ж додатково ввести операцію скасування останньої введеної команди, то ми будемо мати справу з деком.

Завдання до практичної роботи:

1. Опрацюйте матеріал лекції «Дек. Реалізація деку на основі масиву».

2. Напишіть *C*-програму, яка демонструє роботу деку на основі масиву та складається з трьох файлів. Сам дек повинен бути реалізований у файлах **`deq.h`**

і **deq.c**, а в третьому файлі **main.c** слід розташовувати функції, які призначені для тестування деку.

3. Програма повинна реалізувати наступні дії:

– створити дек розмірністю *N* (табл. 6.1);

Таблиця 6.1 – Дані для виконання завдань

Варіант	**N**	**a**	**b**	**n**	**m**	**n1**	**m1**	**N1**	**s**
1	16	30	58	3	5	4	4	4	14
2	18	11	45	7	6	6	7	5	10
3	20	76	98	5	5	3	7	2	16
4	17	23	32	3	8	5	6	4	10
5	12	28	54	2	3	3	2	6	10
6	19	21	54	4	7	5	6	3	12
7	10	43	65	2	4	3	3	8	9
8	17	16	48	4	6	5	5	5	17
9	15	32	87	3	7	4	6	6	11
10	18	32	75	5	3	4	4	4	12
11	20	41	76	7	6	4	9	2	10
12	14	27	75	3	5	4	4	6	13
13	11	15	37	2	2	3	1	8	12
14	13	72	99	4	3	3	4	7	15
15	16	37	83	4	5	6	3	6	11
16	18	54	79	6	4	5	5	4	8
17	17	22	47	4	6	5	5	3	10
18	16	17	57	3	5	4	4	2	12
19	15	33	68	2	4	3	3	4	11
20	14	19	46	5	4	3	6	5	10

– заповнити дек випадковими цілими числами в діапазоні від ***a*** до ***b*** за принципом: до голови деку заносяться числа, які є меншими за значення **(*a*+*b*)/2**, а інші – до хвоста деку. Після занесення чергового числа необхідно вивести на екран вміст деку, щоб контролювати правильність його наповнення;

– витягти з голови деку **n** чисел і вивести на екран вміст деку;

– витягти з хвоста деку **m** чисел і вивести на екран вміст деку;

– вивести на екран кількість парних чисел, які на даний момент знаходяться в деку;

– додати до голови деку **n1** випадкових чисел і вивести вміст деку;

– додати до хвоста деку ***m*1** випадкових чисел і вивести вміст деку;

– вивести на екран голову та хвіст деку, а також всі елементи деку в зворотному порядку;

– збільшити розмір деку на **N1** елементів від початкової кількості *N*;

– додати **N1** випадкових елементів почергово до голови і хвоста деку (*N***1** – кількість елементів, які додаються до голови і хвоста разом). Після занесення чергового числа необхідно виводити на екран вміст деку;

– зменшити розмір деку до *s* елементів і вивести його вміст на екран.

Практична робота №7
РОБОТА З БІНАРНИМИ ДЕРЕВАМИ

Мета роботи: отримати навички роботи зі створення абстрактного типу даних для двійкового дерева; відпрацювати основні операції зі створення інтерфейсу двійкового дерева пошуку; реалізувати двійкове дерево на мові *C*.

Теоретична частина

Двійкові дерева – це структури даних, що складаються з вузлів, які зберігають значення, а також посилання на свою ліву та праву гілки [1-3, 10]. Кожна гілка, у свою чергу, також є деревом. Вузол, що знаходиться в самій вершині дерева прийнято називати коренем (root). Вузли, що знаходяться в самому низу дерева і не мають потомків називають листами (leaves). Гілки вузла називають потомками (descendants). По відношенню до своїх потомків вузол є батьком (parent) або предком (ancestor). Також, існують сестринські вузли (siblings – рідні брати або сестри) – вузли зі спільним батьком. Аналогічно, у вузла можуть бути дяді (uncle nodes) і дідусі та бабусі (grandparent nodes). Такі назви допомагають розуміти різні алгоритми.

На рис. 7.1 вузол 10 – це корінь, а 7 і 12 – його потомки. Вузли 6, 9, 11 і 14 – це листи. Вузли 7 і 12 – також, як і 6 та 9 є сестринськими вузлами, вузол

10 є дідусем для вузла 6, а 12 – дядя для вузлів 6 і 9.

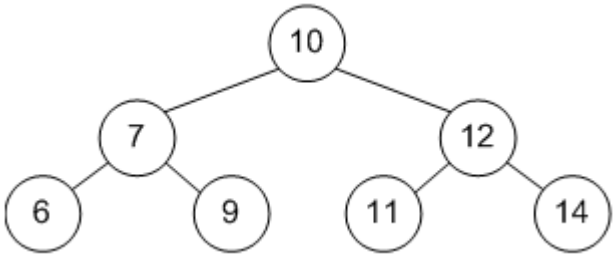


Рисунок 7.1 – Двійкове дерево

Двійкові дерева є найпростішими структурами у порівнянні, наприклад, з іншими деревами. Вони зазвичай реалізують базовий і найприродніший спосіб класифікації елементів – поділяють їх за певною ознакою, розміщуючи одну групу в лівому піддереві, а іншу групу – в правому. В піддеревах рекурсивно підтримується такий самий порядок, за рахунок чого вузли дерева впорядковуються.

Двійкове дерево пошуку (далі ДДП) – це незбалансоване двійкове дерево, в якому елементи, які є більшими ніж корінь розміщуються праворуч, а елементи, які є меншими за корінь – розміщуються ліворуч.

Таке розміщення (зліва менше значення, справа – більше) є необов'язковим, можна розташовувати елементи за зворотним принципом. Відношення «більше» і «менше» – це не обов'язково природне сортування за величиною. Це будь-яка бінарна операція, що дозволяє поділити елементи на дві групи за будь-яким принципом.

Для реалізації бінарного дерева пошуку будемо використовувати структуру **`Node`**, яка містить значення, посилання на праве та ліве піддерево, а також посилання на батьківський вузол, яке в принципі, не є обов'язковим, однак дещо спрощує та прискорює усі алгоритми.

Будемо розглядати випадок, коли в дереві усі значення є різними і не дорівнюють `NULL`.

Зазвичай в якості типу даних використовується тип **void** *, який далі передається функції порівняння через вказівники. На цей раз будемо використовувати тип користувача і макроси:

```
typedef int Item;

#define CMP_LT(a, b) ((a) < (b))
#define CMP_GT(a, b) ((a) > (b))

typedef struct nodeBT
{
    Item data;
    struct nodeBT *left;
    struct nodeBT *right;
    struct nodeBT *parent;
} Node;
```

7.1. Створення нового вузла

Спочатку напишемо функцію `getFreeNode()`, яка створює новий вузол. Вона приймає в якості аргументів значення визначеного типу та вказівник на батьківський вузол. Кореневий елемент не має батька, тому значення вказівника **parent** дорівнює `NULL`.

```
Node *getFreeNode(Item value, Node *parent)
{
    Node *tmp = (Node *) malloc(sizeof(Node));
    tmp->left = tmp->right = NULL;
    tmp->data = value;
    tmp->parent = parent;
    return tmp;
}
```

7.2. Формування двійкового дерева пошуку

Розглянемо ситуації, які можуть виникнути під час формування ДДП шляхом додавання нового вузла. Серед них слід виділити наступні:

1) Дерево є пустим. В цьому випадку новий вузол стає коренем ДДП.

2) Нове значення менше кореневого. В цьому випадку значення повинне бути додано зліва. Якщо зліва вже стоїть елемент, то повторюємо цю операцію, але в якості кореневого вузла розглядаємо лівий вузол. Якщо зліва немає елемента, то додаємо новий вузол.

3) Нове значення більше кореневого. В цьому випадку нове значення повинне бути додано з правого боку. Якщо праворуч вже стоїть елемент, то повторюємо операцію, тільки в якості кореневого розглядаємо правий вузол. Якщо праворуч немає вузла, то додаємо новий вузол.

Припустимо, що нам необхідно помістити до ДДП наступні значення: 10, 7, 9, 12, 6, 14, 11, 3, 4.

Перше значення (10) стає коренем (рис. 7.2).

Рисунок 7.2 – Дерево з одним вузлом (коренем)

Друге значення (7) є меншим ніж 10, тому воно поміщається зліва (рис. 7.3).

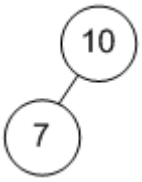


Рисунок 7.3 – Дерево з коренем і потомком, що розміщується ліворуч

Далі число 9 є меншим ніж 10, тому вузол повинен розташовуватися зліва, але ліворуч вже стоїть значення. Порівнюємо: 9 більше ніж 7, тому новий вузол стає правим потомком для вузла 7 (рис. 7.4).

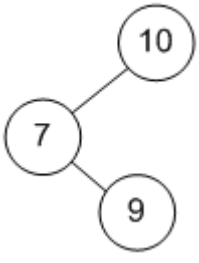


Рисунок 7.4 – Двійкове дерево пошуку після додавання вузлів 10, 7 і 9

Потім число 12 поміщаємо праворуч від 10 (рис. 7.5).

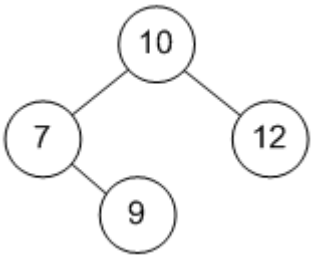


Рисунок 7.5 – Двійкове дерево пошуку після додавання вузлів 10, 7 , 9 і 12

Число 6 менше 10 і менше 7, тому розміщуємо його ліворуч від 7 (рис. 7.6).

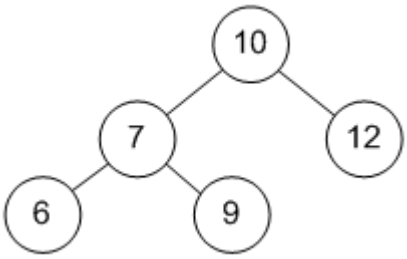


Рисунок 7.6 – Двійкове дерево пошуку після додавання вузлів 10, 7 , 9, 12 і 6

Далі додаємо за розглянутим раніше принципом до двійкового дерева вузли 14, 11, 3 та 4 (рис. 7.7).

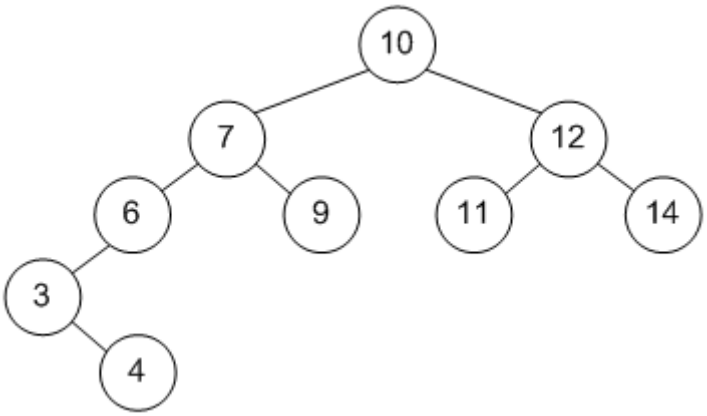


Рисунок 7.7 – Двійкове дерево пошуку після додавання всіх вузлів

7.3. Функція додавання вузла до двійкового дерева

Для реалізації функції додавання `insert()` нам буде потрібна змінна **`tmp`** як вказівник на структуру **`Node`**, що дорівнює `NULL`.

```
Node *tmp = NULL;
```

Перевіряємо, чи є дерево пустим. Якщо це так, то вставляємо корінь:

```
if(*head == NULL)
{
    *head = getFreeNode(value, NULL);
    return;
}
```

Далі проходимо по дереву та шукаємо місце для додавання:

```
tmp = *head;
```

Поки не дійшли до пустого вузла, виконуємо цикл **`while`**`(tmp)`:

```
while(tmp) { ... }
```

Якщо значення більше, ніж значення поточного вузла:

```
if(CMP_GT(value, tmp->data)) { ... }
```

Якщо при цьому правий вузол не є пустим, то коренем будемо вважати тепер праву гілку і починаємо цикл спочатку:

```
if(tmp->right)
{
    tmp = tmp->right;
    continue;
}
```

Якщо правої гілки немає, то додаємо вузол справа:

```
else
{
    tmp->right = getFreeNode(value, tmp);
    return;
}
```

Також обробляємо ліву гілку:

```
else
{
    if(CMP_LT(value, tmp->data))
    {
        if(tmp->left)
        {
            tmp = tmp->left;
            continue;
        }
```

```
        else
        {
            tmp->left = getFreeNode(value, tmp);
            return;
        }
    }
    else
        exit(2);
}
```

Повний код функції `insert()` має такий вигляд:

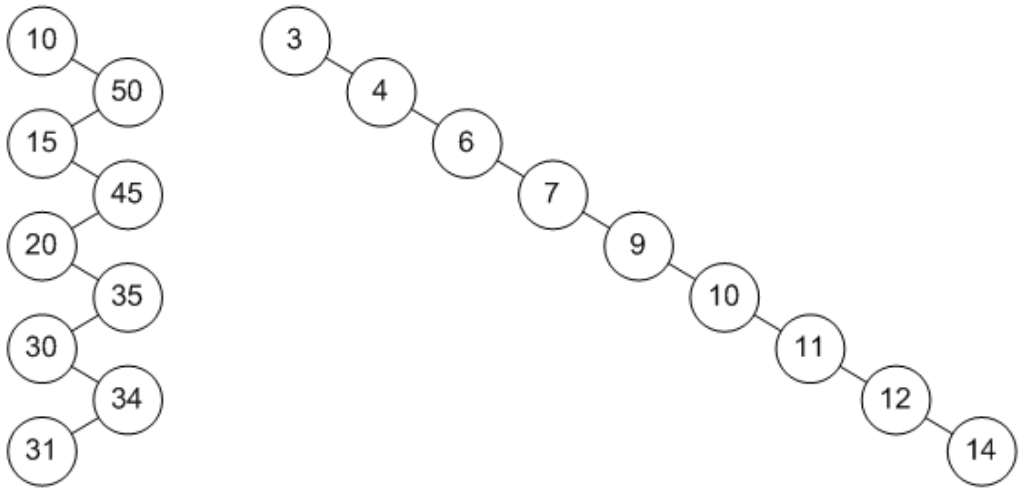
```
void insert(Node **head, int value)
{
    Node *tmp = NULL;
    if(*head == NULL)
    {
        *head = getFreeNode(value, NULL);
        return;
    }
    tmp = *head;
    while(tmp)
    {
        if(CMP_GT(value, tmp->data))
        {
            if(tmp->right)
            {
                tmp = tmp->right;
                continue;
            }
            else
            {
                tmp->right = getFreeNode(value, tmp);
                return;
            }
        }
        else
        {
            if(CMP_LT(value, tmp->data))
            {
                if(tmp->left)
                {
                    tmp = tmp->left;
                    continue;
                }
                else
                {
                    tmp->left = getFreeNode(value, tmp);
                    return;
                }
            }
```

```
                else
                {
                    exit(2);
                }
            }
        }
    }
```

Розглянемо результат додавання вузлів до дерева. Очевидно, що структура дерева буде залежати від порядку, в якому елементи додаються до дерева.

Якщо елементи не упорядковані та їх значення розподілені рівномірно, то дерево буде достатньо збалансованим, тобто шлях від вершини до усіх листів буде однаковим. В такому випадку максимальний час доступу до листа дорівнює `log(n)`, тобто дорівнює висоті дерева. Змінна **`n`** – це кількість вузлів дерева. Але це тільки в найсприятливішому випадку. Якщо ж елементи впорядковані, то дерево не буде збалансованим і буде простягатися в один бік, немов список. Тоді час доступу до останнього вузла буде мати порядок **`n`**. Це є слабкою стороною ДДП, через що застосування цієї структури є досить обмеженим.

Наприклад, дерево може бути отримане шляхом почергового додавання зростаючої та спадної послідовностей чисел (рис. 7.8, а) або шляхом додавання впорядкованої за зростанням послідовності (рис. 7.8, б).



а) б)

Рисунок 7.8 – Приклад збалансованого (а) та незбалансованого дерева (б)

Для вирішення цієї проблеми можна зробити балансування дерева, або використовувати структури, які автоматично проводять самобалансування під час додавання або видалення.

7.4. Пошук вузла в дереві

Відомо, що ліворуч від вузла розташовується елемент, який є меншим ніж поточний вузол. Відповідно, коли у вузла немає лівого потомка, то він є мінімумом в дереві. Таким чином, можна знайти мінімальний елемент дерева:

```
Node *findMinNode(Node *root)
{
    while(root->left)
        root = root->left;
    return root;
}
```

Аналогічним чином можна знайти максимальний елемент:

```
Node *findMaxNode(Node *root)
{
    while(root->right)
        root = root->right;
    return root;
}
```

Знов таки, якщо дерево добре збалансовано, то пошук мінімуму і максимуму буде мати складність порядку `log(n)`, а у випадку поганого балансування буде наближатися до **n**.

Пошук необхідного вузла за значенням схожий на алгоритм бінарного пошуку у відсортованому масиві. Якщо значення більше вузла, то продовжуємо пошук у правому піддереві, якщо менше, то продовжуємо в лівому. Якщо вузлів вже немає, то елемент не міститься в дереві.

```
Node *findNodeByValue(Node *root, Item value)
{
    while(root)
    {
        if(CMP_GT(root->data, value))
        {
            root = root->left;
            continue;
        }
```

```
            else
            {
                if(CMP_LT(root->data, value))
                {
                    root = root->right;
                    continue;
            }
            else
            {
                return root;
            }
        }
        return NULL;
}
```

7.5. Видалення вузла дерева

При видаленні вузла з дерева існує три можливі ситуації:

1) У вузла немає потомків (рис. 7.9). Тоді він просто видаляється (рис. 7.10), а його батьківський вузол обнуляє вказівник на нього.

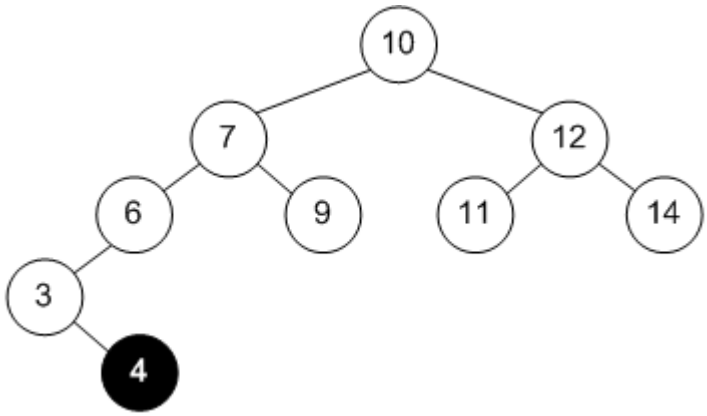


Рисунок 7.9 – Приклад вузла (вузол 4), який не має потомків

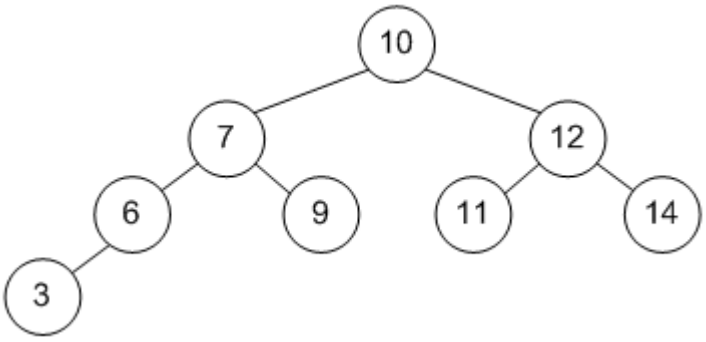


Рисунок 7.10 – Дерево після видалення вузла 4

2) У вузла один потомок (рис. 7.11). В цьому випадку вузол замінюється на свого потомка (рис. 7.12).

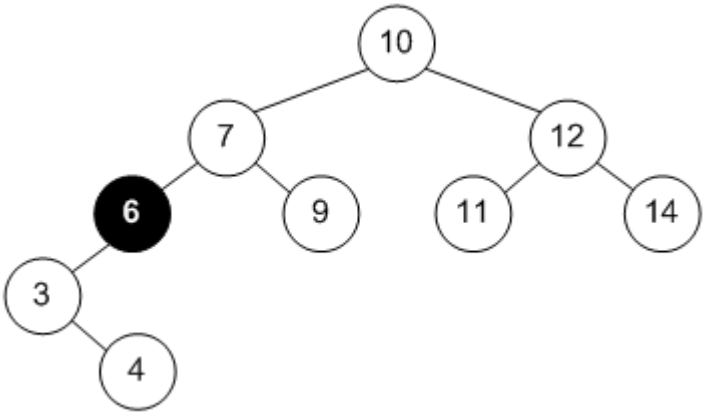


Рисунок 7.11 – Приклад вузла (вузол 6), який має одного потомка (вузол 3)

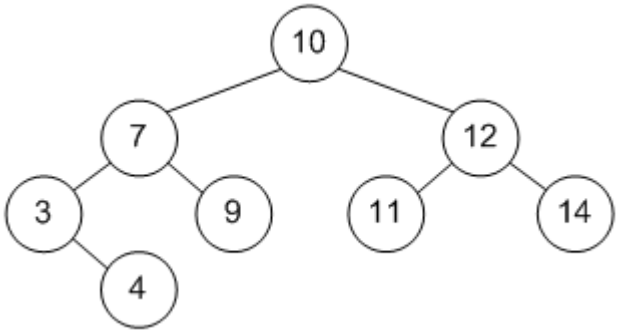


Рисунок 7.12 – Вузол 6 був замінений на вузол 3

3) У вузла два потомки (рис. 7.13). В Цьому випадку вузол не видаляємо, а міняємо його значення на максимум лівого піддерева. Після цього видаляємо максимум лівого піддерева (рис. 7.14).

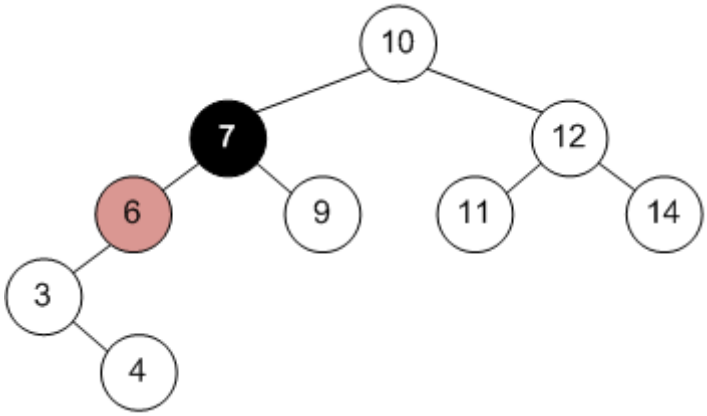


Рисунок 7.13 – Вузол 7 має два потомки (вузол 6 і вузол 9)

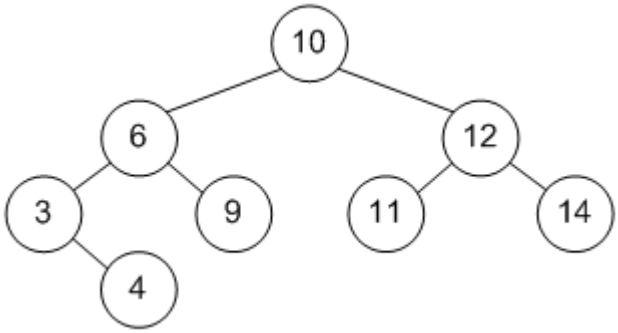


Рисунок 7.14 – Вузол 7 міняємо на максимум лівого піддерева (вузол 6)

Максимум лівого піддерева має не більше одного потомка, тому він просто видаляється. Відомо, що усі значення, які розташовані зліва від кореня є меншими за корінь. Відповідно, максимум лівого піддерева буде, з одного боку, більше усіх елементів лівого піддерева, з іншого боку, менше усіх значень правого піддерева.

Наша функція видалення вузла буде приймати в якості аргументу вузол, який необхідно видалити. Тоді ми повинні розглянути усі зазначені умови та наприкінці функції видалити пам’ять, яку займав вузол:

```
if(target->left && target->right)
{
    // є обидва потомки
}
else
{
    if(target->left)
    {
        // є тільки лівий потомок
    }
    else
    {
        if(target->right)
        {
            // є тільки правий потомок
        }
        else
        {
            // немає потомків
        }
    }
}
free(target);
```

Якщо немає потомків, то треба дізнатися, яким піддеревом є вузол по відношенню до батьківського вузла:

```
Node *ref = target->parent;
if(target == ref->left)
{
    ref->left = NULL;
}
else
{
    ref->right = NULL;
}
```

Якщо є тільки правий або тільки лівий потомок, то підміняємо потомком вузол, що видаляється. Перед цим треба з'ясувати, правим або лівим потомком є вузол, що видаляється:

```
if(target == ref->left)
{
    ref->left = target->left;
}
else
{
    ref->right = target->left;
}
```

або

```
if(target == ref->right)
{
    ref->right = target->right;
}
else
{
    ref->left = target->right;
}
```

Якщо маємо два потомки, то спочатку знаходимо максимум лівого піддерева:

```
Node *localMax = findMaxNode(target->left);
```

Потім підміняємо на цей максимум значення вузла, що видаляється:

```
target->data = localMax->data;
```

Після чого видаляємо цей вузол:

```
removeNodeByPtr(localMax);
return;
```

Ми застосували рекурсію `removeNodeByPtr()`. Виклик було зроблено лише один, оскільки відомо, що максимум не містить обох потомків і є правим потомком свого батьківського вузла. Якщо є бажання замінити виклик функції, то доведеться скопіювати код, що залишився.

Увесь код функції `removeNodeByPtr()` має такий вигляд:

```
void removeNodeByPtr(Node *target)
{
    Node *ref = target->parent;
    if(target->left && target->right)
    {
        Node *localMax = findMaxNode(target->left);
        target->data = localMax->data;
        removeNodeByPtr(localMax);
        return;
    }
    else
    {
        if(target->left)
        {
            if(target == ref->left)
            {
                ref->left = target->left;
            }
            else
            {
                ref->right = target->left;
            }
        }
        else
        {
            if(target->right)
            {
                if(target == ref->right)
                {
                    ref->right = target->right;
                }
                else
                {
                    ref->left = target->right;
                }
            }
            else
            {
                if(target == ref->left)
                {
                    ref->left = NULL;
                }
```

```
                else
                {
                    ref->right = NULL;
                }
            }
        }
    }
    free(target);
}
```

Спростимо дещо нашу роботу та напишемо функцію `deleteValue()`, яка буде видаляти вузол дерева за вказаним значенням:

```
void deleteValue(Node *root, Item value)
{
    Node *target = findNodeByValue(root, value);
    removeNodeByPtr(target);
}
```

Для перевірки результату можна ввести функцію `printTree()` для виводу дерева на екран:

```
void printTree(Node *root, const char *dir, int level)
{
    if(root)
    {
        printf("Рівень: %d    %s = %2d\n",
               level, dir, root->data);
        printTree(root->left, "лівий ", level+1);
        printTree(root->right, "правий", level+1);
    }
}
```

Для перевірки розроблених функцій напишемо *C*-програму, яка створює двійкове дерево пошуку, шукає в ньому максимальний і мінімальний елементи, а також видаляє різні вузли на ньому. Повний текст програми має такий вигляд:

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>

typedef int Item;
#define CMP_LT(a, b) ((a) < (b))
#define CMP_GT(a, b) ((a) > (b))

typedef struct nodeBT
{
    Item data;
    struct nodeBT *left;
    struct nodeBT *right;
```

```
    struct nodeBT *parent;
} Node;

Node *getFreeNode(Item value, Node *parent);
void  insert(Node **head, int value);
Node *findMinNode(Node *root);
Node *findMaxNode(Node *root);
Node *findNodeByValue(Node *root, Item value);
void  removeNodeByPtr(Node *target);
void  deleteValue(Node *root, Item value);
void  printTree(Node *root, const char *dir, int level);

int main(void)
{
    Node *root = NULL;
    Node *ref;

    SetConsoleOutputCP(1251);

    // Додаємо вузли до дерева
    insert(&root, 10);
    insert(&root, 12);
    insert(&root, 8);
    insert(&root, 9);
    insert(&root, 7);
    insert(&root, 3);
    insert(&root, 4);

    printf("=======================================\n");
    printf("ДЕРЕВО ПОШУКУ                          \n");
    printf("=======================================\n");
    printTree(root, "корінь", 0);
    printf("=======================================\n");
    printf("Максимальний вузол  = %2d\n",
           findMaxNode(root)->data);
    printf("Мінімальний вузол   = %2d\n",
           findMinNode(root)->data);
    printf("=======================================\n\n");

    // Видаляємо 4-й вузол
    printf("Видаляємо 4-й вузол\n");
    deleteValue(root, 4);
    printf("=======================================\n");
    printf("ДЕРЕВО ПОШУКУ ПІСЛЯ ВИДАЛЕННЯ ВУЗЛА    \n");
    printf("=======================================\n");
    printTree(root, "корінь", 0);

    ref = findNodeByValue(root, 7)->parent;
    printf("=======================================\n");
    printf("Батьківським вузлом для 7 є %d-й вузол\n",
           ref->data);
    printf("=======================================\n\n");

    // Видаляємо 8-й вузол
    printf("Видаляємо 8-й вузол\n");
```

```
    deleteValue(root, 8);
    printf("=====================================\n");
    printf("ДЕРЕВО ПОШУКУ ПІСЛЯ ВИДАЛЕННЯ ВУЗЛА    \n");
    printf("=====================================\n");
    printTree(root, "корінь", 0);

    printf("\nВидаляємо корінь\n");
    deleteValue(root, 10);
    printf("=====================================\n");
    printf("ДЕРЕВО ПОШУКУ ПІСЛЯ ВИДАЛЕННЯ КОРЕНЯ   \n");
    printf("=====================================\n");
    printTree(root, "корінь", 0);
    return 0;
}

Node *getFreeNode(Item value, Node *parent)
{
    Node *tmp = (Node *) malloc(sizeof(Node));
    tmp->left = tmp->right = NULL;
    tmp->data = value;
    tmp->parent = parent;
    return tmp;
}
void insert(Node **head, int value)
{
    Node *tmp = NULL;

    if(*head == NULL) // Якщо дерево є пустим -
    {                 // вставляємо корінь
        *head = getFreeNode(value, NULL);
        return;
    }
    tmp = *head;
    while(tmp)    // Поки не дійшли до пустого вузла
    {
        if(CMP_GT(value, tmp->data)) // Якщо значення більше,
        {                            // ніж значення поточного
            if(tmp->right)           // вузла
            {
                tmp = tmp->right;
                continue;
            }
            else
            {
                tmp->right = getFreeNode(value, tmp);
                return;
            }
        }
        else
        {
```

```
            if(CMP_LT(value, tmp->data))
            {
                if(tmp->left)
                {
                    tmp = tmp->left;
                    continue;
                }
                else
                {
                    tmp->left = getFreeNode(value, tmp);
                    return;
                }
            }
            else
            {
                exit(2);
            }
        }
    }
}

Node *findMinNode(Node *root)
{
    while(root->left)
        root = root->left;
    return root;
}
Node *findMaxNode(Node *root)
{
    while(root->right)
        root = root->right;
    return root;
}

Node *findNodeByValue(Node *root, Item value)
{
    while(root)
    {
        if(CMP_GT(root->data, value))
        {
            root = root->left;
            continue;
        }
        else
        {
            if(CMP_LT(root->data, value))
            {
                root = root->right;
                continue;
            }
            else
            {
```

```
                return root;
            }
        }
    }
    return NULL;
}

void removeNodeByPtr(Node *target)
{
    Node *ref = target->parent;

    if(target->left && target->right)
    {
        Node *localMax = findMaxNode(target->left);
        target->data = localMax->data;
        removeNodeByPtr(localMax);
        return;
    }
    else
    {
        if(target->left)
        {
            if(target == ref->left)
            {
                ref->left = target->left;
            }
            else
            {
                ref->right = target->left;
            }
        }
        else
        {
            if(target->right)
            {
                if(target == ref->right)
                {
                    ref->right = target->right;
                }
                else
                {
                    ref->left = target->right;
                }
            }
            else
            {
                if(target == ref->left)
                {
                    ref->left = NULL;
                }
                else
                {
```

```
                    ref->right = NULL;
                }
            }
        }
    }
    free(target);
}

void deleteValue(Node *root, Item value)
{
    Node *target = findNodeByValue(root, value);
    removeNodeByPtr(target);
}

void printTree(Node *root, const char *dir, int level)
{
    if(root)
    {
        printf("Рівень: %d    %s = %2d\n",
               level, dir, root->data);
        printTree(root->left, "лівий ", level + 1);
        printTree(root->right, "правий", level + 1);
    }
}
```

Результат роботи програми наведено на рис. 7.15.

Зовнішній вигляд двійкового дерева пошуку, яке було створене в щойно розглянутій програмі, наведено на рис. 7.16. На ньому розшифрована інформація, яка виводиться на екран функцією `printTree()`.

Далі в програмі необхідно було видалити вузол 4 (рис. 7.17).

Двійкове дерево пошуку після видалення вузла 4 за допомогою функції `deleteValue()` наведено на рис. 7.18.

Знаходження батьківського вузла для вузла 7 пояснюється на рис. 7.19. Спочатку в програмі був знайдений вузол 7, а потім для нього був визначений батьківський (вузол 8).

Далі в програмі необхідно було видалити вузол 8 (рис. 7.19). Після видалення ДДП приймає вигляд як на рис. 7.20.

Наступним у програмі необхідно було видалити вузол 10, тобто корінь дерева (рис. 7.21).

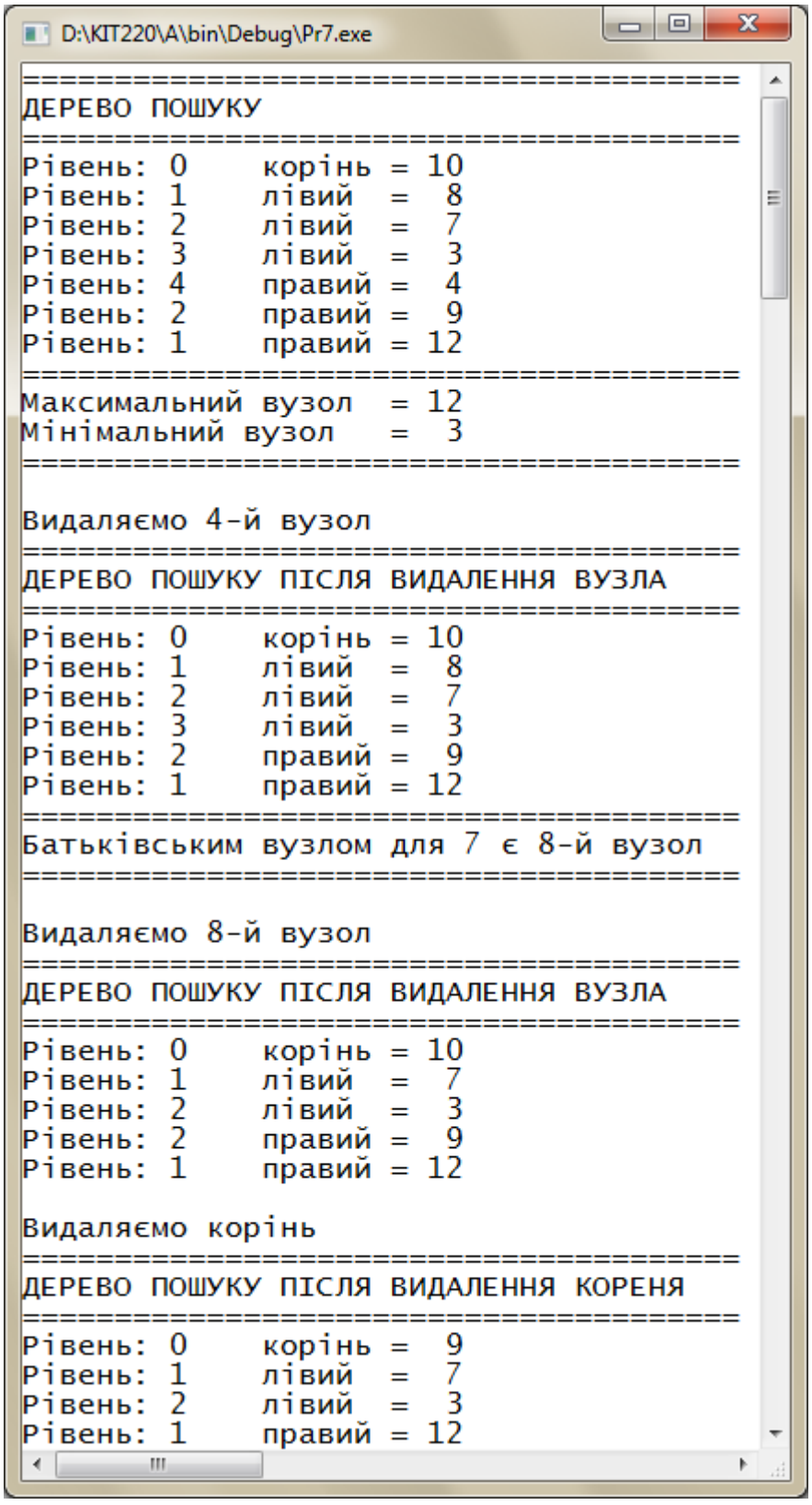


Рисунок 7.15 – Результат роботи програми, що демонструє застосування функцій для двійкового дерева пошуку

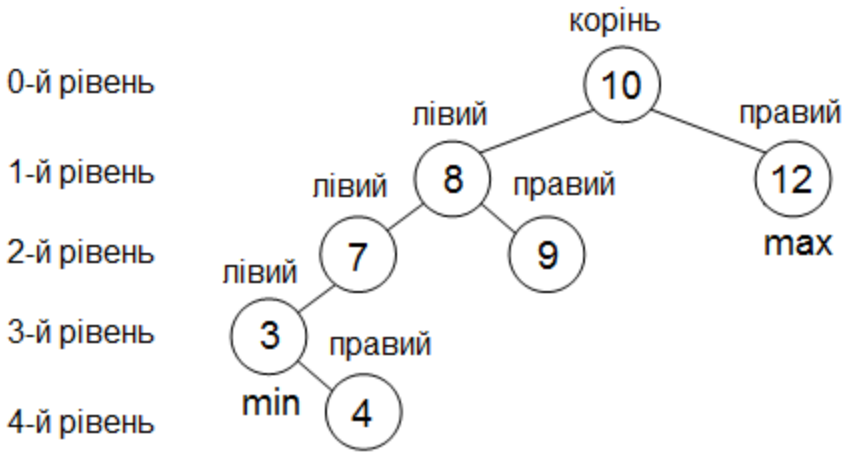


Рисунок 7.16 – Результат роботи частини програми, яка відповідає за створення двійкового дерева та знаходження мінімального і максимального елементів

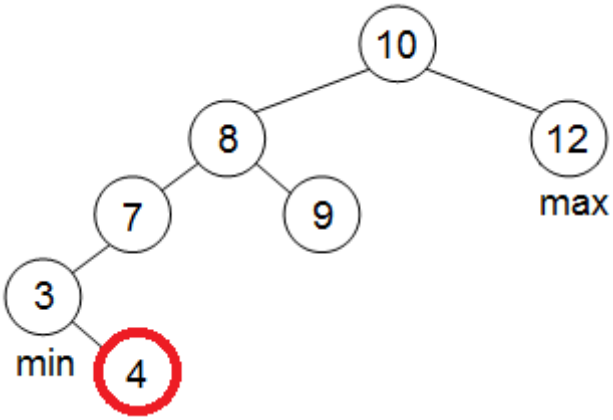


Рисунок 7.17 – Вузол 4, що підлягає видаленню

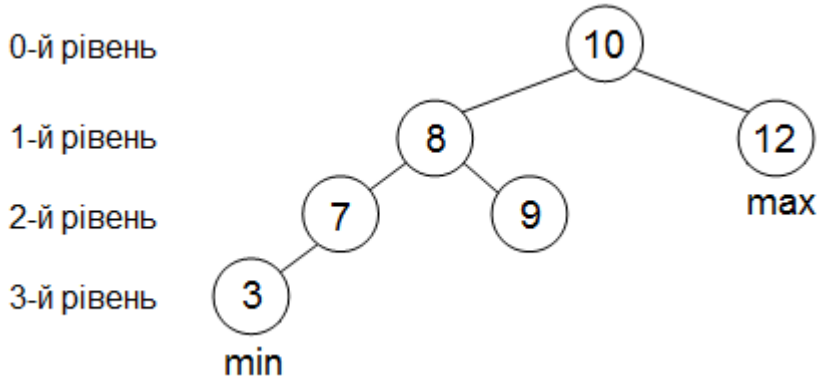


Рисунок 7.18 – ДДП після видалення вузла 4

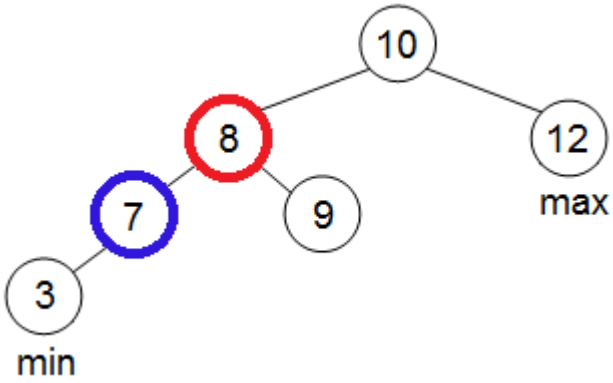


Рисунок 7.19 – Визначення батьківського вузла для вузла 7

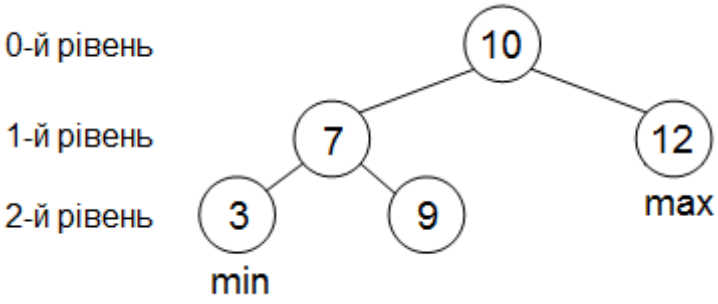


Рисунок 7.20 – ДДП після видалення вузла 8

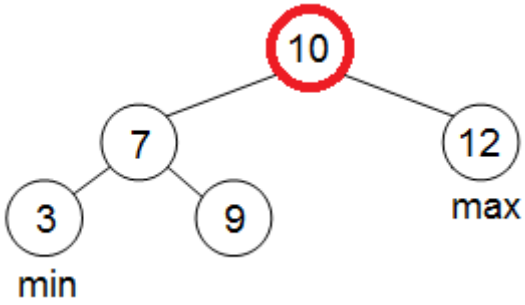


Рисунок 7.21 – Вузол 10 (корінь), що підлягає видаленню

Двійкове дерево пошуку, після видалення вузла 10 за допомогою функції `deleteValue()`, наведено на рис. 7.22.

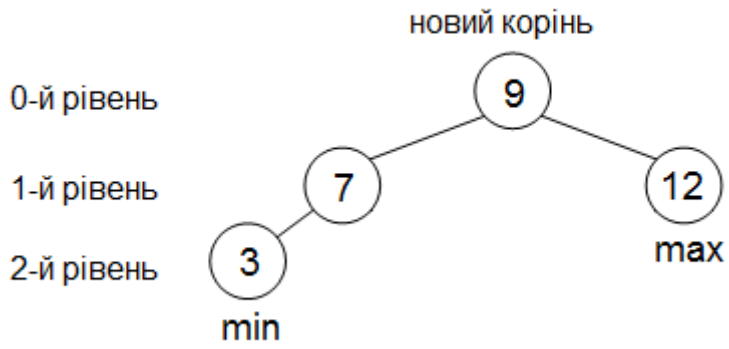


Рисунок 7.22 – ДДП після видалення вузла 10 (кореня)

В результаті видалення кореня дерева ми отримали новий корінь, яким став найбільший правий вузол лівого піддерева.

Завдання до практичної роботи:

1. Опрацюйте матеріал лекції «Двійкове дерево пошуку», а також теоретичну частину даної практичної роботи.

2. Напишіть *C*-програму, яка:

– створює двійкове дерево пошуку відповідно до вашого варіанту (див. додаток 4) та знаходить в ньому максимальний і мінімальний елементи;

– видаляє визначені в завданні корінь дерева, його лист та вузол, що має два потомки;

– знаходить батьківський вузол для визначеного в завданні вузла.

3. Нарисуйте в робочому зошиті ДДП після створення (на зразок рис. 7.16) з зазначенням його рівнів та максимального і мінімального елементів.

4. Нарисуйте двійкові дерева після кожного видалення вузла.

5. Позначте на рисунку батьківський вузол для визначеного у вашому завданні вузла.

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1. Прата С. Язык программирования С. Лекции и упражнения – М.: Диалектика-Вильямс, 2015. – 928 с.

2. Дейтел П., Дейтел Х. С для программистов с введением в С11 – М.: ДМК Пресс, 2014. – 544 с.

3. Дейтел П., Дейтел Х. Как программировать на *C*, 7-е издание, 2014. – 1000 с.

4. Керниган Б., Ритчи Д. Язык программирования С – М.: Диалектика–Вильямс, 2015. – 288 с.

5. Шилдт Г. С: полное руководство, классическое издание – М.: "Диалектика–Вильямс", 2017. – 704 с.

6. Односвязный список. URL: https://learnc.info/adt/linked_list.html (дата обращения: 26.03.2020).

7. Стек. URL: https://learnc.info/adt/stack.html (дата обращения: 03.04.2020).

8. Очередь. URL: https://prog-cpp.ru/data-queue/ (дата обращения: 25.03.2020).

9. Двухсвязный линейный список. URL: https://prog-cpp.ru/data-dls/ (дата обращения: 27.03.2020).

10. Обход двоичного дерева. URL: https://learnc.info/adt/binary_tree_traversal.html (дата обращения: 28.05.2020).

ДОДАТКИ

Додаток 1

Варіанти завдань до практичної роботи №1

Варіант №1

Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього інформацію про десерти у вигляді таблиці:

```
============================================================
Dessert             Calories        Fat     Carbs     Protein
============================================================
Frozen yoghurt        159            6.0       24        4.0
Ice cream sandwich    237            9.0       37        4.3
Eclair                262           16.0       24        6.0
Cupcake               305            3.7       67        4.3
Gingerbread           356           16.0       49        3.9
============================================================
```

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **`input.txt`**, та створіть масив структур визначеного типу.

2) Занесіть інформацію з файлу **`input.txt`** до масиву структур і виведіть її на екран за допомогою окремої функції. Зверніть увагу на те, що під час виводу інформації спочатку треба додати стовпець з номерами рядків (нумерація починається з 1), а стовпець `Calories` необхідно розмістити останнім.

3) Напишіть функцію, яка б передавала до функції `main()` в якості результату та виводила на екран сумарну кількість жирів, вуглеводів та протеїнів, що зазначені в стовпцях `Fat`, `Carbs` і `Protein` відповідно.

4) Напишіть функцію, яка б видаляла інформацію про будь-який десерт. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Виведіть на екран таблицю, отриману після видалення рядка.

5) Напишіть функцію, яка б додавала інформацію про новий десерт в заданому рядку. Номер рядка та інформацію про десерт слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

Продовження додатка 1

6) Напишіть функцію, яка сортує масив структур за збільшенням калорійності (`Calories`). Результат занесіть до файлу **`output.txt`**, а також виведіть його на екран.

Варіант №2

Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього інформацію у вигляді таблиці:

```
================================================================================
Name               Position                    Office          Age   Start date
================================================================================
Airi Satou         Accountant                  Tokyo           33    28.11.2018
Angelica Ramos     Chief Executive Office      London          47    09.10.2009
Ashton Cox         Junior Technical Author     San Francisco   66    12.01.2009
Bradley Greer      Software Engineer           London          41    13.10.2012
Brenden Wagner     Software Engineer           San Francisco   28    07.06.2011
Brielle Williamson Integration Specialist      New York        61    02.12.2012
Bruno Nash         Software Engineer           London          38    03.05.2011
Caesar Vance       Pre-Sales Support           New York        21    12.12.2013
Cara Stevens       Sales Assistant             New York        46    06.12.2011
Cedric Kelly       Senior Javascript Developer Edinburgh       22    29.03.2012
================================================================================
```

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **`input.txt`**, та створіть масив структур визначеного типу. Зверніть увагу на те, що поле `Start date` являє собою вкладену структуру.

2) Перенесіть інформацію з файлу **`input.txt`** до масиву структур і виведіть її на екран за допомогою окремої функції. Зверніть увагу на те, що під час виводу першим слід додати стовпець з номерами рядків (нумерація починається з 1), а стовпець `Start date` необхідно розмістити після стовпця `Name`.

3) Напишіть функцію, яка б передавала до функції `main()` в якості результату та виводила на екран сумарний вік осіб, що зазначені в стовпці `Age`.

4) Напишіть функцію, яка б видаляла інформацію про будь-якого працівника. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Результат видалення рядка виведіть на екран.

5) Напишіть функцію, яка б додавала інформацію про нового працівника в заданому рядку. Номер рядка та інформацію про працівника слід вводити з

клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за збільшенням віку (`Age`) працівників. Результат сортування занесіть до файлу **`output.txt`**, а також виведіть його на екран.

Варіант №3

Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього інформацію у вигляді таблиці:

```
================================================================================
First name  Last name   Birthday    Company  Country       Balance Points Rating
================================================================================
Travis      Banks       05.04.1969  Apple    Italy          61.17     8     6.6
Lawrence    Cain        05.16.1994  IBM      United States  54.81     5     2.5
Dennis      Middleton   01.10.1992  Google   Spain          24.83     4     5.2
Lunea       Valenzuela  01.09.1966  Cisco    Germany        74.07     7     3.7
Quynn       Burgess     09.24.1988  Mozilla  France         16.13     8     2.9
================================================================================
```

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **`input.txt`**, та створіть масив структур визначеного типу. Зверніть увагу на те, що поле `Birthday` являє собою вкладену структуру.

2) Перенесіть інформацію з файлу **`input.txt`** до масиву структур і виведіть її за допомогою окремої функції на екран. Під час виводу інформації про структуру додайте стовпець з поточними номерами (нумерація рядків починається з 1), а стовпці `Country` та `Birthday` поміняйте місцями.

3) Напишіть функцію, яка б передавала до `main()` в якості результату та виводила на екран сумарний баланс працівників усіх компаній. Дані для розрахунку суми зазначені в стовпці `Balance`.

4) Напишіть функцію, яка б видаляла інформацію про будь-яку особу. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Після видалення рядка необхідно знову застосувати функцію виводу структури.

Продовження додатка 1

5) Напишіть функцію, яка б додавала інформацію про нову особу в заданому рядку. Номер рядка та інформацію про нову особу слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за зменшенням рейтингу (`Rating`). Результат сортування занесіть до файлу **`output.txt`**, а також виведіть його на екран.

Варіант №4

Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього інформацію у вигляді таблиці:

```
============================================================
Name            ISO Code   Population     Size (km²)   Density
============================================================
India              IN      1324171354      3287263      402.82
China              CN      1403500365      9596961      146.24
Italy              IT        60483973       301340      200.72
United States      US       327167434      9833520       33.27
Canada             CA        37602103      9984670        3.77
Australia          AU        25475400      7692024        3.31
Germany            DE        83019200       357578      232.17
============================================================
```

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **`input.txt`**, та створіть масив структур визначеного типу.

2) Перенесіть інформацію з файлу **`input.txt`** до масиву структур і виведіть її за допомогою окремої функції на екран. Зверніть увагу на те, що під час виводу першим слід додати стовпець з номерами рядків (нумерація починається з 1), а стовпець `Density` необхідно розмістити після стовпця `Name`.

3) Напишіть функцію, яка б передавала до `main()` в якості результату та виводила на екран сумарну щільність населення (`Density`) в Індії та Китаї.

4) Напишіть функцію, яка б видаляла рядок з інформацією про будь-яку країну. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Результат видалення рядка виведіть на екран.

Продовження додатка 1

5) Напишіть функцію, яка б додавала інформацію про нову країну в заданому рядку. Номер рядка та інформацію про нову країну слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за зменшенням кількості населення (`Population`). Результат сортування занесіть до файлу **`output.txt`**, а також виведіть його на екран.

Варіант №5

Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього інформацію у вигляді таблиці:

TOTAL GOVERNMENT EXPENDITURES

Fiscal Year	Total Government Expenditures	Federal Government Outlays			Addendum: Federal Grants	State and Local Government Expenditures From Own Sources
		Total	On-Budget	Off-Budget		
1948	45.2	29.8	29.4	0.4	1.1	15.4
1949	56.5	38.8	38.4	0.4	1.6	17.6
1950	63.0	42.6	42.0	0.5	1.9	20.4
1951	67.3	45.5	44.2	1.3	1.9	21.8
1952	91.1	67.7	66.0	1.7	1.9	23.4
1953	100.7	76.1	73.8	2.3	2.2	24.6
1954	98.0	70.9	67.9	2.9	2.2	27.1
1955	98.8	68.4	64.5	4.0	2.3	30.3

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **`input.txt`**, та створіть масив структур визначеного типу. Зверніть увагу на те, що стовпець `Federal Government Outlays` являє собою вкладену структуру.

2) Перенесіть інформацію з файлу **`input.txt`** до масиву структур і виведіть її за допомогою окремої функції на екран.

3) Напишіть функцію, яка б передавала до `main()` в якості результату та виводила на екран суму значень останнього стовпця в таблиці `TOTAL GOVERNMENT EXPENDITURES`.

Продовження додатка 1

4) Напишіть функцію, яка б видаляла інформацію про будь-який введений з клавіатури рік. Виведіть на екран таблицю, отриману після видалення рядка.

5) Напишіть функцію, яка б додавала інформацію про 1956 рік в заданому рядку. Номер рядка та нову інформацію слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за зменшенням років (`Fiscal Year`). Результат сортування занесіть до файлу **`output.txt`**, а також виведіть його на екран.

Варіант №6

Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього інформацію у вигляді таблиці:

```
============================================================
Dessert             Calories        Fat     Carbs     Protein
============================================================
Gingerbread            356          16.0       49         3.9
Ice cream sandwich     237           9.0       37         4.3
Eclair                 262          16.0       24         6.0
Frozen yoghurt         159           6.0       24         4.0
Cupcake                305           3.7       67         4.3
============================================================
```

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **`input.txt`**, та створіть масив структур визначеного типу.

2) Перенесіть інформацію з файлу **`input.txt`** до масиву структур і виведіть її за допомогою окремої функції на екран. Зверніть увагу на те, що під час виводу першим слід додати стовпець з номерами рядків (нумерація починається з 1), а стовпці `Fat` і `Protein` необхідно поміняти місцями.

3) Напишіть функцію, яка б передавала до `main()` в якості результату та виводила на екран сумарну кількість калорій, що зазначені в стовпці `Calories`.

4) Напишіть функцію, яка б видаляла інформацію про будь-який десерт.

Номер рядка, який необхідно видалити, треба вводити з клавіатури. Виведіть на екран таблицю, отриману після видалення рядка.

5) Напишіть функцію, яка б додавала інформацію про новий десерт в заданому рядку. Номер рядка та інформацію про десерт слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за зменшенням калорійності (`Calories`). Результат занесіть до файлу **`output.txt`**, а також виведіть його на екран.

Варіант №7

Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього інформацію у вигляді таблиці:

```
================================================================================
Name                Position                    Office          Age   Start date
================================================================================
Charde Marshall     Regional Director           San Francisco   36    16.10.2018
Colleen Hurst       Javascript Developer        San Francisco   39    15.09.2009
Dai Rios            Personnel Lead              Edinburgh       35    26.09.2012
Donna Snider        Customer Support            New York        27    25.01.2011
Doris Wilder        Sales Assistant             Sydney          23    20.09.2010
Finn Camacho        Support Engineer            San Francisco   47    07.07.2019
Fiona Green         Chief Operating Officer     San Francisco   48    11.03.2010
Garrett Winters     Accountant                  Tokyo           63    25.07.2011
Gavin Cortez        Team Leader                 San Francisco   22    26.10.2018
Gavin Joyce         Developer                   Edinburgh       42    22.12.2010
================================================================================
```

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **`input.txt`**, та створіть масив структур визначеного типу. Зверніть увагу на те, що поле `Start date` являє собою вкладену структуру.

2) Перенесіть інформацію з файлу **`input.txt`** до масиву структур і виведіть її за допомогою окремої функції на екран. Зверніть увагу на те, що під час виводу першим слід додати стовпець з номерами рядків (нумерація починається з 1), а стовпець `Start date` необхідно розмістити після стовпця `Position`.

Продовження додатка 1

3) Напишіть функцію, яка б передавала до `main()` в якості результату та виводила на екран сумарний вік осіб, які народилися у вересні.

4) Напишіть функцію, яка б видаляла рядок з інформацією про будь-якого працівника. Номер рядка слід вводити з клавіатури. Виведіть на екран таблицю, отриману після видалення рядка.

5) Напишіть функцію, яка б додавала інформацію про нового працівника в заданому рядку. Номер рядка та нову інформацію слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за зменшенням дати початку роботи (`Start date`). Результат занесіть до файлу **`output.txt`**, а також виведіть його на екран.

Варіант №8

Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього інформацію у вигляді таблиці:

TOTAL GOVERNMENT EXPENDITURES

Fiscal Year	Total Government Expenditures	Federal Government Outlays			Addendum: Federal Grants	State and Local Government Expenditures From Own Sources
		Total	On-Budget	Off-Budget		
1956	103.7	70.6	65.7	5.0	2.3	33.0
1957	112.7	76.6	70.6	6.0	2.6	36.2
1958	122.7	82.4	74.9	7.5	2.9	40.3
1959	135.3	92.1	83.1	9.0	3.5	43.2
1960	137.9	92.2	81.3	10.9	3.8	45.7
1961	147.1	97.7	86.0	11.7	4.0	49.4
1962	160.0	106.8	93.3	13.5	4.5	53.2
1963	168.2	111.3	96.4	15.0	4.9	56.9
1964	179.9	118.5	102.8	15.7	5.7	61.3

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **`input.txt`**, та створіть масив структур визначеного типу. Зверніть увагу на те,

що стовпець `Federal Government Outlays` являє собою вкладену структуру.

2) Перенесіть інформацію з файлу **`input.txt`** до масиву структур і виведіть її за допомогою окремої функції на екран.

3) Напишіть функцію, яка б передавала до `main()` в якості результату та виводила на екран суму значень для другого та останнього стовпців, що зазначені в таблиці `TOTAL GOVERNMENT EXPENDITURES`.

4) Напишіть функцію, яка б видаляла інформацію про будь-який введений з клавіатури рік. Виведіть на екран таблицю після видалення рядка.

5) Напишіть функцію, яка б додавала інформацію про 1965 рік в заданому рядку. Номер рядка та нову інформацію слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за збільшенням значень в стовпці `Total Government Expenditures`. Результат сортування занесіть до файлу **`output.txt`**, а також виведіть його на екран.

Варіант №9

Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього інформацію у вигляді таблиці:

```
=============================================================================
Last name  First name   Birthday    Company  Country       Balance Points Rating
=============================================================================
Banks      Travis       05.04.1969  Apple    Italy          61.17    8      6.6
Cain       Lawrence     05.16.1994  IBM      United States  54.81    5      2.5
Middleton  Dennis       01.10.1992  Google   Spain          24.83    4      5.2
Valenzuela Lunea        01.09.1966  Cisco    Germany        74.07    7      3.7
Burgess    Quynn        09.24.1988  Mozilla  France         16.13    8      2.9
Lott       Lillith      12.10.1978  Dell     Australia      30.12    1      7.9
Freeman    Cheyenne     01.28.1981  LG       Germany        49.82    1      6.0
Houston    Jorden       05.03.1978  Adobe    Canada         47.31    2      1.5
Harris     Merritt      12.29.1989  Apple    United States  77.33    6      0.1
Gregory    Brynn        01.13.1994  Adobe    Spain          90.24    9      8.2
=============================================================================
```

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу

input.txt, та створіть масив структур визначеного типу. Зверніть увагу на те, що поле `Birthday` являє собою вкладену структуру.

2) Перенесіть інформацію з файлу **input.txt** до масиву структур і виведіть її за допомогою окремої функції на екран. Під час виводу інформації про структуру додайте стовпець з поточними номерами (нумерація рядків починається з 1), а стовпці `Country` та `Balance` поміняйте місцями.

3) Напишіть функцію, яка б передавала до `main()` в якості результату та виводила на екран сумарний рейтинг працівників усіх компаній. Значення рейтингу визначається додаванням чисел в стовпці `Rating`.

4) Напишіть функцію, яка б видаляла інформацію про будь-яку особу. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Після видалення рядка необхідно знову застосувати функцію виводу структури.

5) Напишіть функцію, яка б додавала інформацію про нову особу в заданому рядку. Номер рядка та інформацію про нову особу слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за збільшенням балансу (`Balance`). Результат занесіть до файлу **output.txt** та виведіть його на екран.

Варіант №10

Створіть в директорії з *C*-програмою текстовий файл **input.txt** та занесіть до нього інформацію у вигляді таблиці:

```
============================================================
Name            Population     Size (km²)   Density   ISO Code
============================================================
India          1324171354      3287263      402.82      IN
China          1403500365      9596961      146.24      CN
Italy            60483973       301340      200.72      IT
United States   327167434      9833520       33.27      US
Canada           37602103      9984670        3.77      CA
Australia        25475400      7692024        3.31      AU
Germany          83019200       357578      232.17      DE
============================================================
```

Продовження додатка 1

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **input.txt**, та створіть масив структур визначеного типу.

2) Перенесіть інформацію з файлу **input.txt** до масиву структур і виведіть її за допомогою окремої функції на екран. Зверніть увагу на те, що під час виводу першим слід додати стовпець з номерами рядків (нумерація починається з 1), а стовпець `Density` необхідно розмістити після стовпця `Name`.

3) Напишіть функцію, яка б передавала до `main()` в якості результату та виводила на екран сумарну площу всіх країн, що зазначені в таблиці.

4) Напишіть функцію, яка б видаляла рядок з інформацією про будь-яку країну. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Результат видалення рядка виведіть на екран.

5) Напишіть функцію, яка б додавала інформацію про нову країну в заданому рядку. Номер рядка та інформацію про нову країну слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за зменшенням площі країн (`Size`). Результат занесіть до файлу **output.txt** та виведіть на екран.

Варіант №11

Створіть в директорії з *C*-програмою текстовий файл **input.txt** та занесіть до нього інформацію у вигляді таблиці:

```
================================================================================
Name               Position                    Office          Age   Start date
================================================================================
Gloria Little      Systems Administrator       New York        59    10.04.2019
Haley Kennedy      Senior Marketing Designer   London          43    18.12.2012
Hermione Butler    Regional Director           London          47    21.03.2011
Herrod Chandler    Sales Assistant             San Francisco   59    06.08.2012
Hope Fuentes       Secretary                   San Francisco   41    12.02.2010
Howard Hatfield    Office Manager              San Francisco   51    16.12.2018
Jackson Bradshaw   Director                    New York        65    26.09.2008
Jena Gaines        Office Manager              London          30    19.12.2018
Jenette Caldwell   Development Lead            New York        30    03.09.2011
Jennifer Acosta    Junior JavaScript Developer Edinburgh       43    01.02.2013
================================================================================
```

Продовження додатка 1

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **input.txt**, та створіть масив структур визначеного типу. Зверніть увагу на те, що поле `Start date` являє собою вкладену структуру.

2) Перенесіть інформацію з файлу **input.txt** до масиву структур і виведіть її на екран за допомогою окремої функції. Зверніть увагу на те, що під час виводу першим слід додати стовпець з номерами рядків (нумерація починається з 1), а стовпець `Start date` необхідно розмістити після стовпця `Position`.

3) Напишіть функцію, яка б виводила на екран сумарний вік осіб, що знаходяться в `San Francisco`.

4) Напишіть функцію, яка б видаляла інформацію про будь-якого працівника. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Результат видалення рядка виведіть на екран.

5) Напишіть функцію, яка б додавала інформацію про нового працівника в заданому рядку. Номер рядка та інформацію про працівника слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за збільшенням дати початку роботи в компанії (`Start date`). Результат сортування занесіть до файлу **output.txt**, а також виведіть його на екран.

Варіант №12

Створіть в директорії з *C*-програмою текстовий файл **input.txt** та занесіть до нього інформацію у вигляді таблиці:

```
=========================================================
Last Name   First Name   Birthday     Company   Phone
=========================================================
Middleton   Dennis       01.10.1992   Google    580-9501
Banks       Travis       05.04.1969   Apple     200-4324
Cain        Lawrence     05.16.1994   IBM       701-3108
Valenzuela  Lunea        01.09.1966   Cisco     970-8655
Burgess     Quynn        09.24.1988   Mozilla   430-8943
=========================================================
```

Продовження додатка 1

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **input.txt**, та створіть масив структур визначеного типу. Зверніть увагу на те, що поле `Birthday` являє собою вкладену структуру.

2) Занесіть інформацію з файлу **input.txt** до масиву структур і виведіть її на екран за допомогою окремої функції. Зверніть увагу на те, що під час виводу інформації спочатку треба додати стовпець з номерами рядків (нумерація починається з 1), а стовпці `Company` та `Phone` необхідно поміняти місцями.

3) Напишіть функцію, яка б виводила на екран сумарну кількість років народження працівників, що зазначені в таблиці.

4) Напишіть функцію, яка б видаляла інформацію про працівника. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Виведіть на екран таблицю, отриману після видалення рядка.

5) Напишіть функцію, яка б додавала інформацію про нового працівника в заданому рядку. Номер рядка та інформацію про працівника слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за алфавітом для стовпця `Company`. Результат занесіть до файлу **output.txt** та виведіть на екран.

Варіант №13

Створіть в директорії з *C*-програмою текстовий файл **input.txt** та занесіть до нього інформацію у вигляді таблиці:

```
============================================================
Dessert              Calories      Protein    Fat     Carbs
============================================================
Eclair                 262           6.0     16.0       24
Cupcake                305           4.3      3.7       67
Frozen yoghurt         159           4.0      6.0       24
Ice cream sandwich     237           4.3      9.0       37
Gingerbread            356           3.9     16.0       49
============================================================
```

Продовження додатка 1

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **`input.txt`**, та створіть масив структур визначеного типу.

2) Занесіть інформацію з файлу **`input.txt`** до масиву структур і виведіть її на екран за допомогою окремої функції. Зверніть увагу на те, що під час виводу інформації спочатку треба додати стовпець з номерами рядків (нумерація починається з 1), а стовпець `Fat` необхідно розмістити останнім.

3) Напишіть функцію, яка б виводила на екран сумарну кількість жирів, що зазначені в стовпці `Fat`.

4) Напишіть функцію, яка б видаляла інформацію про будь-який десерт. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Виведіть на екран таблицю, отриману після видалення рядка.

5) Напишіть функцію, яка б додавала інформацію про новий десерт в заданому рядку. Номер рядка та інформацію про десерт слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за збільшенням вмісту жирів (`Fat`). Результат занесіть до файлу **`output.txt`** та виведіть на екран.

Варіант №14

Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього інформацію у вигляді таблиці:

```
================================================================================
Name               Position                    Office          Age   Start date
================================================================================
Jennifer Chang     Regional Director           Singapore       28    14.11.2010
Jonas Alexander    Developer                   San Francisco   30    14.07.2010
Lael Greer         Systems Administrator       London          21    27.02.2009
Martena Mccray     Post-Sales support          Edinburgh       46    09.03.2011
Michael Bruce      Javascript Developer        Singapore       29    27.06.2011
Michael Silva      Marketing Designer          London          66    27.11.2012
Michelle House     Integration Specialist      Sydney          37    02.06.2011
Olivia Liang       Support Engineer            Singapore       64    03.02.2011
Paul Byrd          Chief Financial Officer     New York        64    09.06.2010
Prescott Bartlett  Technical Author            London          27    07.05.2011
================================================================================
```

Продовження додатка 1

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **input.txt**, та створіть масив структур визначеного типу. Зверніть увагу на те, що поле `Start date` являє собою вкладену структуру.

2) Перенесіть інформацію з файлу **input.txt** до масиву структур і виведіть її на екран за допомогою окремої функції. Зверніть увагу на те, що під час виводу першим слід додати стовпець з номерами рядків (нумерація починається з 1), а стовпець `Office` необхідно розмістити останнім.

3) Напишіть функцію, яка б передавала до `main()` в якості результату та виводила на екран сумарний вік осіб, яким більше 40 років.

4) Напишіть функцію, яка б видаляла інформацію про будь-якого працівника. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Результат видалення рядка виведіть на екран.

5) Напишіть функцію, яка б додавала інформацію про нового працівника в заданому рядку. Номер рядка та інформацію про працівника слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за зменшенням віку (`Age`) працівників. Результат занесіть до файлу **output.txt** і виведіть на екран.

Варіант №15

Створіть в директорії з *C*-програмою текстовий файл **input.txt** та занесіть до нього інформацію у вигляді таблиці, яка розміщена на наступній сторінці зверху.

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **input.txt**, та створіть масив структур визначеного типу. Зверніть увагу на те, що стовпець `Federal Government Outlays` являє собою вкладену структуру.

2) Перенесіть інформацію з файлу **input.txt** до масиву структур і виведіть її за допомогою окремої функції на екран.

Продовження додатка 1

TOTAL GOVERNMENT EXPENDITURES

Fiscal Year	Total Government Expenditures	Federal Government Outlays			Addendum: Federal Grants	State and Local Government Expenditures From Own Sources
		Total	On-Budget	Off-Budget		
1965	184.7	118.2	101.7	16.5	6.3	66.5
1966	207.5	134.5	114.8	19.7	7.9	73.0
1967	237.3	157.5	137.0	20.4	9.9	79.8
1968	267.3	178.1	155.8	22.3	11.9	89.2
1969	284.5	183.6	158.4	25.2	12.2	100.8
1970	303.7	195.6	168.0	27.6	16.1	108.1
1971	332.1	210.2	177.3	32.8	20.4	122.0
1972	363.1	230.7	193.5	37.2	25.8	132.4

3) Напишіть функцію, яка б передавала до `main()` в якості результату та виводила на екран суму значень для стовпців `Total`, `On-Budget` та `Off-Budget`, що об'єднані під назвою `Federal Government Outlays`.

4) Напишіть функцію, яка б видаляла інформацію про будь-який введений з клавіатури рік. Результат видалення рядка виведіть на екран.

5) Напишіть функцію, яка б додавала інформацію про 1973 рік в заданому рядку. Номер рядка та нову інформацію слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за зменшенням значень у стовпці `Total`. Результат занесіть до файлу **`output.txt`** та виведіть на екран.

Варіант №16

Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього інформацію у вигляді таблиці, яка знаходиться на наступній сторінці зверху.

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **`input.txt`**, та створіть масив структур визначеного типу. Зверніть увагу, що поле `Date` являє собою вкладену структуру.

Продовження додатка 1

NEW YORK SPOT PRICE

Metals	Date	Weight	Bid	Ask	Change	Low	High
GOLD	17.04.2019	1001	1815.20	1816.20	21.30	1793.20	1817.00
SILVER	03.05.2020	1701	26.74	27.24	0.64	25.67	27.31
PLATINUM	22.01.2021	2231	1123.00	1133.00	29.00	1100.00	1134.00
PALLADIUM	02.05.2020	1148	2274.00	2424.00	65.00	2230.00	2424.00
RHODIUM	12.02.2021	2001	19000.00	21000.00	3.00	1.00	2.00

2) Перенесіть інформацію з файлу **`input.txt`** до масиву структур і виведіть її на екран за допомогою окремої функції. Зверніть увагу на те, що під час виводу першим слід додати стовпець з номерами рядків (нумерація починається з 1), а стовпці `Bid` і `Ask` необхідно поміняти місцями.

3) Напишіть функцію, яка б передавала до `main()` в якості результату та виводила на екран сумарну вагу дорогоцінних та платинових металів (`Weight`).

4) Напишіть функцію, яка б видаляла інформацію про будь-який дорогоцінний або платиновий метал. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Результат видалення рядка виведіть на екран.

5) Напишіть функцію, яка б додавала інформацію про новий платиновий метал (наприклад, іридій, рутеній, осмій) в заданому рядку. Номер рядка та інформацію про метал слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за зменшенням дати (`Date`). Результат занесіть до файлу **`output.txt`** та виведіть на екран.

Варіант №17

Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього інформацію про десерти у вигляді таблиці.

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **`input.txt`**, та створіть масив структур визначеного типу.

Продовження додатка 1

Dessert	Calories	Fat	Carbs	Protein
Gingerbread	456	15.0	50	3.5
Frozen yoghurt	253	6.4	25	4.0
Eclair	362	15.1	24	6.1
Cupcake	288	3.7	66	4.3
Ice cream sandwich	230	9.3	33	4.4

2) Занесіть інформацію з файлу **input.txt** до масиву структур і виведіть її на екран за допомогою окремої функції. Зверніть увагу на те, що під час виводу інформації спочатку треба додати стовпець з номерами рядків (нумерація починається з 1), а стовпець `Fat` необхідно розмістити останнім.

3) Напишіть функцію, яка б передавала до функції `main()` в якості результату та виводила на екран сумарну кількість жирів і протеїнів, що зазначені в стовпцях `Fat` і `Protein` відповідно.

4) Напишіть функцію, яка б видаляла інформацію про будь-який десерт. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Виведіть на екран таблицю, отриману після видалення рядка.

5) Напишіть функцію, яка б додавала інформацію про новий десерт в заданому рядку. Номер рядка та інформацію про десерт слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за збільшенням кількості жирів (`Fat`). Результат занесіть до файлу **output.txt**, а також виведіть його на екран.

Варіант №18

Створіть в директорії з *C*-програмою текстовий файл **input.txt** та занесіть до нього інформацію у вигляді таблиці, що наведена нижче.

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **input.txt**, та створіть масив структур визначеного типу.

Зверніть увагу, що поле `Start date` являє собою вкладену структуру.

```
=============================================================================
Name               Position                    Office          Age   Start date
=============================================================================
Airi Price         Accountant                  Rome            33    09.10.2011
Ashton Cox         Junior Technical Author     San Francisco   44    12.03.2012
Bradley Greer      Software Engineer           London          46    13.11.2014
Brenden Wagner     Software Engineer           New York        35    07.04.2015
Brielle Williamson Integration Specialist      New York        37    02.10.2015
Bruno Nash         Software Engineer           London          39    03.04.2018
Caesar Vance       Pre-Sales Support           San Francisco   26    12.11.2015
Cara Stevens       Sales Assistant             New York        49    06.10.2013
Cedric Kelly       Senior Javascript Developer Edinburgh       26    29.01.2019
=============================================================================
```

2) Перенесіть інформацію з файлу **`input.txt`** до масиву структур і виведіть її на екран за допомогою окремої функції. Зверніть увагу на те, що під час виводу першим слід додати стовпець з номерами рядків (нумерація починається з 1), а стовпець `Start date` необхідно розмістити після стовпця `Position`.

3) Напишіть функцію, яка б передавала до функції `main()` в якості результату та виводила на екран сумарний вік осіб, що зазначені в стовпці `Age`.

4) Напишіть функцію, яка б видаляла інформацію про будь-якого працівника. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Результат видалення рядка виведіть на екран.

5) Напишіть функцію, яка б додавала інформацію про нового працівника в заданому рядку. Номер рядка та інформацію про працівника слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за зменшенням віку (`Age`) працівників. Результат сортування занесіть до файлу **`output.txt`**, а також виведіть його на екран.

Варіант №19

Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього інформацію у вигляді таблиці:

Продовження додатка 1

First name	Last name	Birthday	Company	Country	Balance	Points	Rating
Lawrence	Cain	15.16.1999	IBM	United States	51.81	6	4.5
Lunea	Valenzuela	01.11.1978	Cisco	Germany	74.09	7	3.3
Travis	Banks	05.04.1965	Apple	Italy	51.17	5	6.1
Quynn	Burgess	19.24.1984	Mozilla	France	37.15	2	2.1
Dennis	Middleton	11.10.1996	Google	Spain	24.83	4	1.2

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **`input.txt`**, та створіть масив структур визначеного типу. Зверніть увагу на те, що поле `Birthday` являє собою вкладену структуру.

2) Перенесіть інформацію з файлу **`input.txt`** до масиву структур і виведіть її за допомогою окремої функції на екран. Під час виводу інформації про структуру додайте стовпець з поточними номерами (нумерація рядків починається з 1), а стовпці `Company` та `Birthday` поміняйте місцями.

3) Напишіть функцію, яка б передавала до `main()` в якості результату та виводила на екран сумарний баланс працівників усіх компаній. Дані для розрахунку суми зазначені в стовпці `Balance`.

4) Напишіть функцію, яка б видаляла інформацію про будь-яку особу. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Після видалення рядка необхідно знову застосувати функцію виводу структури.

5) Напишіть функцію, яка б додавала інформацію про нову особу в заданому рядку. Номер рядка та інформацію про нову особу слід вводити з клавіатури. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за збільшенням рейтингу (`Rating`). Результат сортування занесіть до файлу **`output.txt`**, а також виведіть його на екран.

Варіант №20

Створіть в директорії з *C*-програмою текстовий файл **`input.txt`** та занесіть до нього інформацію у вигляді таблиці:

Продовження додатка 1

Name	ISO Code	Population	Size (km²)	Density
Australia	AU	25475400	7692024	3.31
United States	US	327167434	9833520	33.27
Germany	DE	83019200	357578	232.17
India	IN	1324171354	3287263	402.82
Canada	CA	37602103	9984670	3.77
China	CN	1403500365	9596961	146.24
Italy	IT	60483973	301340	200.72

1) В *C*-програмі опишіть шаблон структури, яка відповідає вмісту файлу **`input.txt`**, та створіть масив структур визначеного типу.

2) Перенесіть інформацію з файлу **`input.txt`** до масиву структур і виведіть її за допомогою окремої функції на екран. Зверніть увагу на те, що під час виводу першим слід додати стовпець з номерами рядків (нумерація починається з 1), а стовпець `Population` необхідно розмістити після стовпця `Name`.

3) Напишіть функцію, яка б передавала до `main()` в якості результату та виводила на екран сумарну кількість населення (`Population`) в Італії та Канаді.

4) Напишіть функцію, яка б видаляла рядок з інформацією про будь-яку країну. Номер рядка, який необхідно видалити, треба вводити з клавіатури. Результат видалення рядка виведіть на екран.

5) Напишіть функцію, яка б додавала інформацію про нову країну в заданому рядку. Номер рядка та інформацію про нову країну слід вводити з клавіатури. Інформацію з рядка, на місце якого будуть вводитися нові дані, та всю подальшу інформацію з наступних рядків треба перемістити на один рядок нижче. Результат додавання рядка виведіть на екран.

6) Напишіть функцію, яка сортує масив структур за збільшенням кількості населення (`Population`). Результат сортування занесіть до файлу **`output.txt`**, а також виведіть його на екран.

Додаток 2

Варіанти завдань до практичної роботи №2

Варіант №1. Побітові логічні операції.

Напишіть функцію, яка після вводу з клавіатури числа типу **`unsigned int`** в діапазоні від 2147483648 до 4294967295 визначає кількість одиниць у двійковому представленні числа та кількість одиниць, що стоять на непарних місцях (біти 1, 3, 5, …).

Результати виконання програми для трьох довільних чисел з зазначеного діапазону необхідно представити в файлі **`output.txt`** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №1: Побітові логічні операції (32 біти)
================================================================
Введіть число від 2147483648 до 4294967295:     3157413737
================================================================
Число: 3157413737 = 0xBC324F69 = 10111100 00110010 01001111 01101001
================================================================
Кількість одиниць у числі:                        17
Кількість одиниць, що стоять на непарних місцях:   9
================================================================
```

Варіант №2. Переставляння бітів в межах байту.

Напишіть функцію, яка після вводу з клавіатури числа типу **`unsigned char`** в діапазоні від 128 до 255, повертає число, у двійковому представленні якого у порівняні з двійковим представленням введеного числа переставлені місцями біти: нульовий біт – з сьомим, перший – з шостим, другий – з п’ятим, третій – з четвертим.

Результати виконання програми для трьох довільних чисел з зазначеного діапазону необхідно представити в файлі **`output.txt`** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №2: Переставляння бітів (8 бітів)
========================================
Введіть число:   163
========================================
Введено:   163 = 0xA3 = 10100011
Результат: 197 = 0xC5 = 11000101
========================================
```

Продовження додатка 2

Варіант №3. Переставляння пар бітів в межах байту.

Напишіть функцію, яка після вводу з клавіатури числа типу **`unsigned char`** в діапазоні від 128 до 255, повертає число, в двійковому представленні якого у порівняні з двійковим представленням введеного числа переставлені місцями пари бітів: пара наймолодших бітів (1 і 0) з парою найстарших бітів (7 і 6), передостання пара бітів з правого боку (3 і 2) з передостанньою парою бітів з лівого боку (5 і 4).

Результати виконання програми для трьох довільних чисел з зазначеного діапазону необхідно представити в файлі **`output.txt`** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №3: Переставляння пар бітів (8 бітів)
==============================================
Введіть число:   163
==============================================
Введено:   163 = 0xA3 = 10100011
Результат: 202 = 0xCA = 11001010
==============================================
```

Варіант №4. Побітові логічні операції.

Напишіть функцію, яка після вводу з клавіатури числа типу **`unsigned int`** в діапазоні від 2147483648 до 4294967295, визначає кількість одиниць в числі та кількість одиниць, що стоять на парних місцях (біти 0, 2, 4, …).

Результати виконання програми для трьох довільних чисел з зазначеного діапазону необхідно представити в файлі **`output.txt`** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №4: Побітові логічні операції (32 біти)
================================================================
Введіть число від 2147483648 до 4294967295:    2157113638
================================================================
Число: 2157113638 = 0x8092F126 = 10000000 10010010 11110001 00100110
================================================================
Кількість одиниць у числі:                    12
Кількість одиниць, що стоять на парних місцях:     5
================================================================
```

Продовження додатка 2

Варіант №5. Переставляння байтів.

Напишіть функцію, яка після вводу з клавіатури числа типу **unsigned short** в діапазоні від 19256 до 52768, повертає інше число, в двійковому представленні якого у порівняні з двійковим представленням введеного числа переставлені місцями байти: молодший байт зі старшим байтом.

Результати виконання програми для трьох довільних чисел з зазначеного діапазону необхідно представити в файлі **output.txt** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №5. Переставляння байтів (16 бітів)
================================================
Введіть число:    41906
================================================
Введено:    41906 = 0xA3B2 = 10100011 10110010
Результат:  45731 = 0xB2A3 = 10110010 10100011
================================================
```

Варіант №6. Переставляння четвірок бітів.

Напишіть функцію, яка після вводу з клавіатури числа типу **unsigned char** в діапазоні від 148 до 215, повертає число, в двійковому представленні якого у порівняні з двійковим представленням введеного числа переставлені місцями четвірки бітів: четвірка наймолодших бітів переставлена місцем з четвіркою найстарших бітів.

Результати виконання програми для трьох довільних чисел необхідно представити в файлі **output.txt** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №6. Переставляння четвірок (8 бітів)
============================================
Введіть число:   163
============================================
Введено:   163 = 0xA3 = 10100011
Результат:  58 = 0x3A = 00111010
============================================
```

Продовження додатка 2

Варіант №7. Обмін сусідніми бітами.

Напишіть функцію, яка після вводу з клавіатури числа типу **unsigned char** в діапазоні від 138 до 225, повертає число, в двійковому представленні якого у порівняні з двійковим представленням введеного числа переставлені місцями сусідні біти: нульовий з першим, другий з третім і т. д.

Результати виконання програми для трьох довільних чисел з зазначеного діапазону необхідно представити в файлі **output.txt** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №7. Обмін сусідніми бітами (8 бітів)
============================================
Введіть число:   163
============================================
Введено:   163 = 0xA3 = 10100011
Результат:  83 = 0x53 = 01010011
============================================
```

Варіант №8. Циклічний зсув вліво в межах кожного байту.

Напишіть функцію, яка після вводу з клавіатури числа типу **unsigned short** в діапазоні від 19256 до 52768 і цілого числа в діапазоні від 1 до 7, повертає результат циклічного зсуву вліво в межах кожного байту на кількість бітів, яка визначається другим числом.

Результати виконання програми для трьох довільних пар чисел з зазначених діапазонів необхідно представити в файлі **output.txt** та на екрані.

Приклад результату роботи програми для однієї пари чисел:

```
Варіант №8: Циклічний зсув вліво в межах байтів (16 бітів)
============================================================
Введіть два числа:    23456 3
============================================================
Введено:    23456 = 0x5BA0 = 01011011 10100000
Результат:  55813 = 0xDA05 = 11011010 00000101
============================================================
```

Продовження додатка 2

Варіант №9. Переставляння бітів в четвірках.

Напишіть функцію, яка після вводу з клавіатури числа типу **unsigned char** в діапазоні від 131 до 235, повертає число, в двійковому представленні якого у порівняні з двійковим представленням введеного числа переставлені місцями біти в четвірках: нульовий біт кожної четвірки з третім бітом тієї ж самої четвірки, перший біт кожної четвірки з другим.

Результати виконання програми для трьох довільних чисел з зазначеного діапазону необхідно представити в файлі **output.txt** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №9: Переставляння бітів в четвірках (8 бітів)
=====================================================
Введіть число:   163
=====================================================
Введено:   163 = 0xA3 = 10100011
Результат:  92 = 0x5C = 01011100
=====================================================
```

Варіант №10. Переставляння бітів в межах кожного з байтів.

Напишіть функцію, яка після вводу з клавіатури числа типу **unsigned short** в діапазоні від 19256 до 52768, повертає число, в двійковому представленні якого у порівняні з двійковим представленням введеного числа переставлені місцями біти в байтах: нульовий біт байта з сьомим бітом того ж самого байта, перший біт байта – з шостим бітом байта і т. д.

Результати виконання програми для трьох довільних чисел з зазначеного діапазону необхідно представити в файлі **output.txt** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №10: Переставляння бітів в байтах (16 бітів)
=====================================================
Введіть число:    23456
=====================================================
Введено:    23456 = 0x5BA0 = 01011011 10100000
Результат:  55813 = 0xDA05 = 11011010 00000101
=====================================================
```

Продовження додатка 2

Варіант №11. Дзеркальне відображення бітів в межах кожного з байтів.

Напишіть функцію, яка після вводу з клавіатури числа типу **unsigned int** в діапазоні від 2147483648 до 4294967295, повертає результат дзеркального відображення бітів в межах кожного з байтів: нульовий біт поточного байту відображається на сьомий, перший – на шостий, другий – на п’ятий, третій – на четвертий.

Результати виконання програми для трьох довільних чисел з зазначеного діапазону необхідно представити в файлі **output.txt** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №11: Дзеркальне відображення бітів в межах байтів (32 біти)
========================================================================
Введіть число:    3975853717
========================================================================
Введено:  3975853717 = 0xECFAB695 = 11101100 11111010 10110110 10010101
Результат: 929000873 = 0x375F6DA9 = 00110111 01011111 01101101 10101001
========================================================================
```

Варіант №12. Циклічний зсув вліво.

Напишіть функцію, яка після вводу з клавіатури числа типу **unsigned short** в діапазоні від 19256 до 52768 і цілого числа в діапазоні від 1 до 15, повертає результат циклічного зсуву вліво першого числа на кількість бітів, яка визначається другим числом.

Результати виконання програми для трьох довільних пар чисел з зазначених діапазонів необхідно представити в файлі **output.txt** та на екрані.

Приклад результату роботи програми для однієї пари чисел:

```
Варіант №12: Циклічний зсув вліво (16 бітів)
=============================================
Введіть два числа:    23456 3
=============================================
Введено:    23456 = 0x5BA0 = 01011011 10100000
Результат:  56578 = 0xDD02 = 11011101 00000010
=============================================
```

Продовження додатка 2

Варіант №13. Циклічний зсув вправо.

Напишіть функцію, яка після вводу з клавіатури числа типу **unsigned short** в діапазоні від 19256 до 52768 і цілого числа в діапазоні від 1 до 15, повертає результат циклічного зсуву вправо першого числа на кількість бітів, яка визначається другим числом.

Результати виконання програми для трьох довільних пар чисел з зазначених діапазонів необхідно представити в файлі **output.txt** та на екрані.

Приклад результату роботи програми для однієї пари чисел:

```
Варіант №13: Циклічний зсув вправо (16 бітів)
================================================
Введіть два числа:    23456 3
================================================
Введено:    23456 = 0x5BA0 = 01011011 10100000
Результат:   2932 = 0x0B74 = 00001011 01110100
================================================
```

Варіант №14. Побітові логічні операції.

Напишіть функцію, яка після вводу двох чисел **a** і **b** типу **unsigned int** в діапазоні від 2147483648 до 4294967295 формує третє число **c** за наступним принципом: 0-й байт – результат побітової операції "АБО" (|) нульових байтів операндів; 1-й байт – результат побітової операції "І" (&) перших байтів операндів; 2-й байт – результат побітової операції "виключне АБО" (^) других байтів операндів; 3-й байт – результат побітового заперечення (~) для 3-го байту операнда **a.** Результати виконання програми для трьох довільних пар чисел з зазначеного діапазону необхідно представити в файлі **output.txt** та на екрані.

Приклад результату роботи програми для однієї пари чисел:

```
Варіант №14: Побітові логічні операції (32 біти)
================================================================
Введіть число a:    2157113638
Введіть число b:    3294967295
================================================================
a: 2157113638 = 0x8092F126 = 10000000 10010010 11110001 00100110
b: 3294967295 = 0xC46535FF = 11000100 01100101 00110101 11111111
================================================================
c: 2146906623 = 0x7FF731FF = 01111111 11110111 00110001 11111111
================================================================
```

Продовження додатка 2

Варіант №15. Циклічний зсув вправо в межах кожного з байтів.

Напишіть функцію, яка після вводу з клавіатури числа типу **unsigned short** в діапазоні від 19256 до 52768 і цілого числа в діапазоні від 1 до 7, повертає результат циклічного зсуву вправо в межах кожного з байтів на кількість бітів, яка визначається другим числом.

Результати виконання програми для трьох довільних пар чисел з зазначених діапазонів необхідно представити в файлі **output.txt** та на екрані.

Приклад результату роботи програми для однієї пари чисел:

```
Варіант №15: Циклічний зсув вправо в межах байтів (16 бітів)
============================================================
Введіть два числа:    23456 3
============================================================
Введено:    23456 = 0x5BA0 = 01011011 10100000
Результат:  27412 = 0x6B14 = 01101011 00010100
============================================================
```

Варіант №16. Дзеркальне відображення байтів.

Напишіть функцію, яка після вводу з клавіатури числа типу **unsigned short** в діапазоні від 19256 до 52768, повертає результат дзеркального відображення байтів: всі біти нульового байту дзеркально записуються до першого байту, а всі біти першого байту – до нульового байту. Вісь дзеркальної симетрії знаходиться на межі байтів.

Результати виконання програми для трьох довільних чисел з зазначеного діапазону необхідно представити в файлі **output.txt** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №16: Дзеркальне відображення бітів (16 бітів)
=================================================
Введіть число:    23456
=================================================
Введено:    23456 = 0x5BA0 = 01011011 10100000
Результат:   1498 = 0x05DA = 00000101 11011010
=================================================
```

Продовження додатка 2

Варіант №17. Побітові логічні операції.

Напишіть функцію, яка після вводу з клавіатури числа типу **`unsigned int`** в діапазоні від 2147483648 до 4294967295 визначає кількість одиниць у двійковому представленні числа та кількість одиниць, що стоять на парних місцях (біти 0, 2, 4, …).

Результати виконання програми для трьох довільних чисел з зазначеного діапазону необхідно представити в файлі **`output.txt`** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №17: Побітові логічні операції (32 біти)
=================================================================
Введіть число від 2147483648 до 4294967295:     3157413737
=================================================================
Число: 3157413737 = 0xBC324F69 = 10111100 00110010 01001111 01101001
=================================================================
Кількість одиниць у числі:                        17
Кількість одиниць, що стоять на парних місцях:     8
=================================================================
```

Варіант №18. Переставляння бітів в межах байту.

Напишіть функцію, яка після вводу з клавіатури числа типу **`unsigned char`** в діапазоні від 128 до 255, повертає число, у двійковому представленні якого у порівняні з двійковим представленням введеного числа переставлені місцями біти: нульовий біт – з сьомим, перший – з шостим, другий – з п’ятим, третій – з четвертим.

Результати виконання програми для трьох довільних чисел з зазначеного діапазону необхідно представити в файлі **`output.txt`** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №18: Переставляння бітів (8 бітів)
=========================================
Введіть число:   208
=========================================
Введено:   208 = 0xD0 = 11010000
Результат:  11 = 0x0B = 00001011
=========================================
```

Продовження додатка 2

Варіант №19. Переставляння пар бітів в межах байту.

Напишіть функцію, яка після вводу з клавіатури числа типу **`unsigned char`** в діапазоні від 128 до 255, повертає число, в двійковому представленні якого у порівняні з двійковим представленням введеного числа переставлені місцями пари бітів: пара наймолодших бітів (1 і 0) з парою найстарших бітів (7 і 6), передостання пара бітів з правого боку (3 і 2) з передостанньою парою бітів з лівого боку (5 і 4).

Результати виконання програми для трьох довільних чисел з зазначеного діапазону необхідно представити в файлі **`output.txt`** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №19: Переставляння пар бітів (8 бітів)
===============================================
Введіть число:   178
===============================================
Введено:   178 = 0xB2 = 10110010
Результат: 142 = 0x8E = 10001110
===============================================
```

Варіант №20. Побітові логічні операції.

Напишіть функцію, яка після вводу з клавіатури числа типу **`unsigned int`** в діапазоні від 2147483648 до 4294967295, визначає кількість одиниць в числі та кількість одиниць, що стоять на непарних місцях (біти 1, 3, 5, …).

Результати виконання програми для трьох довільних чисел з зазначеного діапазону необхідно представити в файлі **`output.txt`** та на екрані.

Приклад результату роботи програми для одного числа наведено нижче:

```
Варіант №20: Побітові логічні операції (32 біти)
=================================================================
Введіть число від 2147483648 до 4294967295:    2157113638
=================================================================
Число: 2157113638 = 0x8092F126 = 10000000 10010010 11110001 00100110
=================================================================
Кількість одиниць у числі:                      12
Кількість одиниць, що стоять на непарних місцях:   7
=================================================================
```

Додаток 3

Приклад програми для роботи зі списком

```
#include <stdio.h>
#include <windows.h>

typedef struct node
{
    int value;
    struct node *next;
} Node;
void  push(Node **head, int data);
int   pop(Node **head);
void  pushBack(Node *head, int value);
int   popBack(Node **head);
void  insert(Node *head, unsigned n, int val);
void  deleteList(Node **head);
int   deleteNth(Node **head, int n);
Node *getLast(Node *head);
Node *getNth(Node* head, int n);
void  fromArray(Node **head, int *arr, size_t size);
void  printLinkedList(const Node *head);

int main(void)
{
    Node *head = NULL;
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

    SetConsoleOutputCP(1251);
    // Створюємо список з масиву
    fromArray(&head, arr, 10);
    printf("================================================\n");
    printf("Зв'язний список                                 \n");
    printf("================================================\n");
    printLinkedList(head);
    // Додаємо вузол зі значенням 55 після 5-го елемента
    // (стане шостим: нумерація з 0)
    printf("\n================================================\n");
    printf("Додаємо елемент 55 до списку після п'ятого      \n");
    printf("================================================\n");
    insert(head, 4, 55);
    printLinkedList(head);
    printf("\n================================================\n");
    printf("Додаємо в кінець списку елементи 11, 12, 13, 14  \n");
    printf("================================================\n");
    pushBack(head, 11);
    pushBack(head, 12);
    pushBack(head, 13);
    pushBack(head, 14);
    printLinkedList(head);
    printf("\n================================================\n");
    printf("Голова списку                                   \n");
```

```
    printf("================================================\n");
    printf("%4d\n", pop(&head));
    printf("\n================================================\n");
    printf("Хвіст списку                                    \n");
    printf("================================================\n");
    printf("%4d\n", popBack(&head));
    printf("\n================================================\n");
    printf("Зв'язний список після видалення голови та хвоста \n");
    printf("================================================\n");
    printLinkedList(head);
    // Видаляємо п’ятий елемент
    printf("\n================================================\n");
    printf("Видаляємо п'ятий елемент                        \n");
    printf("================================================\n");
    deleteNth(&head, 4);
    printLinkedList(head);
    deleteList(&head);
    return 0;
}
void push(Node **head, int data)
{
    Node *tmp = (Node *) malloc(sizeof(Node));
    tmp->value = data;
    tmp->next = (*head);
    (*head) = tmp;
}
int pop(Node **head)
{
    Node* prev = NULL;
    int val;
    if(head == NULL)
        exit(-1);
    prev = (*head);
    val = prev->value;
    (*head) = (*head)->next;
    free(prev);
    return val;
}
void pushBack(Node *head, int value)
{
    Node *last = getLast(head);
    Node *tmp = (Node *) malloc(sizeof(Node));
    tmp->value = value;
    tmp->next = NULL;
    last->next = tmp;
}
void insert(Node *head, unsigned n, int val)
{
    unsigned i = 0;
    Node *tmp = NULL;
```

```
    // Знаходимо потрібний елемент.
    // Якщо вийшли за межі списку, то виходимо з циклу
    while(i < n && head->next)
    {
        head = head->next;
        i++;
    }
    tmp = (Node *) malloc(sizeof(Node));
    tmp->value = val;
    // Якщо це не останній елемент,
    // то next спрямовуємо на наступний вузол
    if(head->next)
        tmp->next = head->next;
    else   // інакше на NULL
        tmp->next = NULL;
    head->next = tmp;
}
int deleteNth(Node **head, int n)
{
    if(n == 0)
        return pop(head);
    else
    {
        Node *prev = getNth(*head, n-1);
        Node *elm  = prev->next;
        int val = elm->value;
        prev->next = elm->next;
        free(elm);
        return val;
    }
}
void fromArray(Node **head, int *arr, size_t size)
{
    size_t i = size - 1;
    if(arr == NULL || size == 0)
        return;
    do
    {
        push(head, arr[i]);
    } while(i-- != 0);
}
void printLinkedList(const Node *head)
{
    while(head)
    {
        printf("%4d", head->value);
        head = head->next;
    }
    printf("\n");
}
```

Продовження додатка 3

```
Node *getLast(Node *head)
{
    if(head == NULL)
        return NULL;
    while(head->next)
        head = head->next;
    return head;
}
Node *getNth(Node* head, int n)
{
    int counter = 0;
    while(counter < n && head)
    {
        head = head->next;
        counter++;
    }
    return head;
}
int popBack(Node **head)
{
    Node *pFwd = NULL;        // поточний вузол
    Node *pBwd = NULL;        // попередній вузол
    // Отримали NULL
    if(!head)
        exit(-1);
    // Список пустий
    if(!(*head))
        exit(-1);
    pFwd = *head;
    while(pFwd->next)
    {
        pBwd = pFwd;
        pFwd = pFwd->next;
    }
    if(pBwd == NULL)
    {
        free(*head);
        *head = NULL;
    }
    else
    {
        free(pFwd->next);
        pBwd->next = NULL;
    }
    return pFwd->value;
}

void deleteList(Node **head)
{
    Node *prev = NULL;
```

```
    while((*head)->next)
    {
        prev = (*head);
        (*head) = (*head)->next;
        free(prev);
    }
    free(*head);
}
```

Результат виконання програми наведено на рис. Д 3.1.

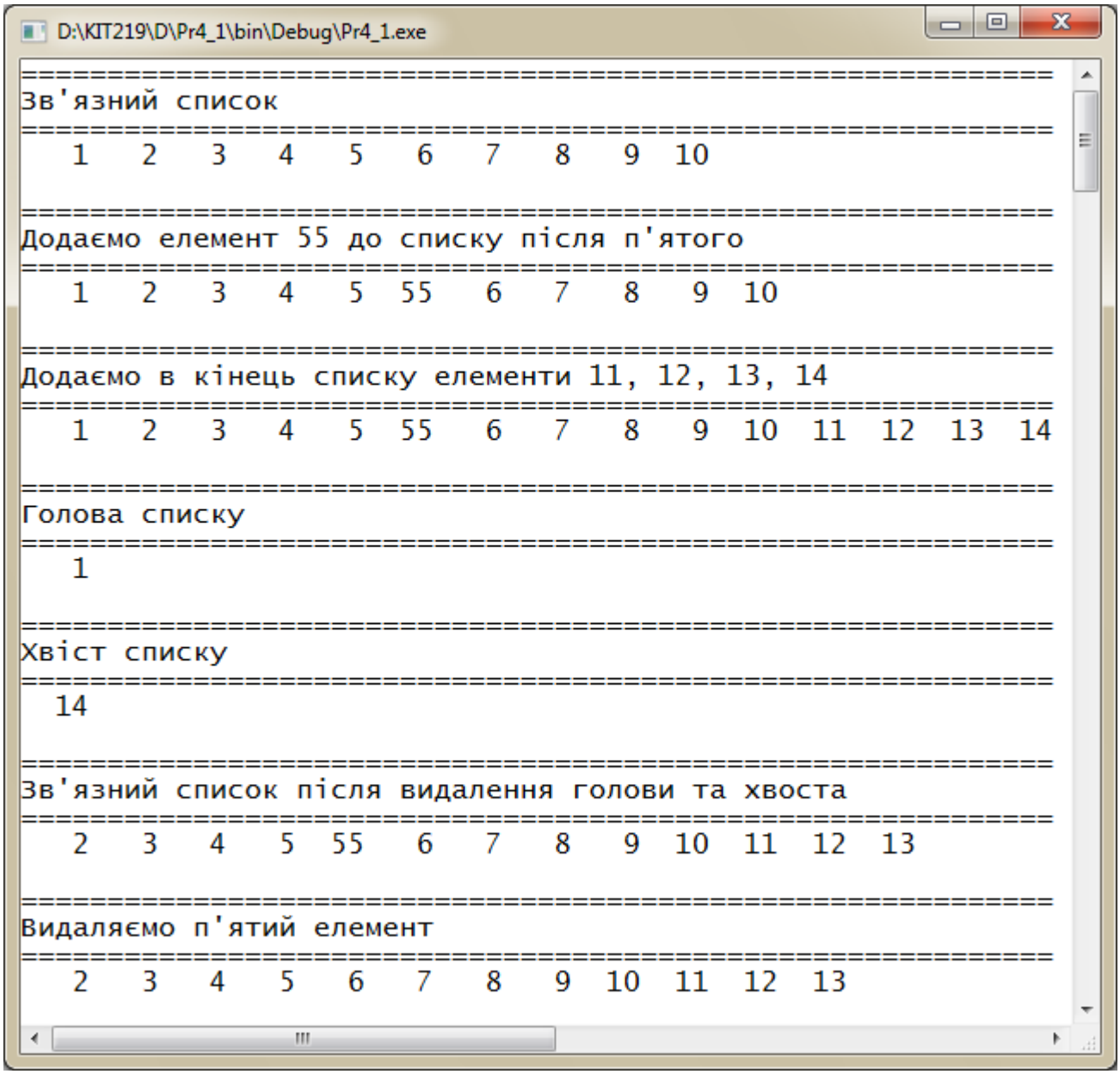


Рисунок Д3.1 – Результат виконання програми для роботи зі зв’язним списком

Додаток 4

Варіанти завдань до практичної роботи №7

Варіант №1

1) Створіть за допомогою *C*-програми ДДП, що складається з наступних вузлів: 41, 78, 15, 48, 98, 11, 13, 27, 47, 91, 14, 33, 55, 50, 9. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 27, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 50, 11, 41.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Варіант №2

1) Створіть за допомогою *C*-програми ДДП, що складається з наступних вузлів: 14, 25, 37, 22, 62, 87, 45, 9, 33, 77, 39, 20, 13, 18, 67. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 77, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 22, 33, 14.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Варіант №3

1) Створіть за допомогою *C*-програми ДДП, що складається з наступних вузлів: 32, 2, 16, 24, 67, 56, 39, 47, 19, 8, 10, 29, 49, 99, 85. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 47, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 49, 16, 32.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Продовження додатка 4

Варіант №4

1) Створіть за допомогою *С*-програми ДДП, що складається з наступних вузлів: 64, 27, 42, 14, 78, 69, 89, 4, 18, 55, 90, 71, 66, 35, 99. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 18, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 99, 69, 64.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Варіант №5

1) Створіть за допомогою *С*-програми ДДП, що складається з наступних вузлів: 27, 19, 10, 7, 29, 77, 55, 13, 58, 22, 63, 71, 18, 44, 65. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 18, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 65, 55, 27.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Варіант №6

1) Створіть за допомогою *С*-програми ДДП, що складається з наступних вузлів: 43, 65, 8, 32, 11, 19, 40, 68, 88, 77, 17, 34, 18, 70, 67. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 68, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 18, 32, 43.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Продовження додатка 4

Варіант №7

1) Створіть за допомогою *С*-програми ДДП, що складається з наступних вузлів: 33, 13, 41, 17, 84, 7, 14, 37, 69, 21, 60, 27, 83, 6, 85. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 7, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 33, 60, 84.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Варіант №8

1) Створіть за допомогою *С*-програми ДДП, що складається з наступних вузлів: 50, 32, 11, 77, 84, 15, 64, 7, 99, 38, 67, 56, 43, 22, 41. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 11, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 22, 64, 50.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Варіант №9

1) Створіть за допомогою *С*-програми ДДП, що складається з наступних вузлів: 74, 6, 37, 62, 17, 85, 90, 22, 79, 11, 77, 44, 95, 88, 55. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 22, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 55, 85, 74.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Продовження додатка 4

Варіант №10

1) Створіть за допомогою *С*-програми ДДП, що складається з наступних вузлів: 38, 12, 6, 9, 91, 44, 27, 57, 39, 21, 88, 76, 33, 51, 30. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 21, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 76, 44, 38.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Варіант №11

1) Створіть за допомогою *С*-програми ДДП, що складається з наступних вузлів: 45, 65, 22, 36, 98, 7, 13, 88, 47, 79, 31, 44, 81, 9, 27. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 88, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 36, 81, 45.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Варіант №12

1) Створіть за допомогою *С*-програми ДДП, що складається з наступних вузлів: 58, 25, 37, 17, 62, 87, 92, 9, 24, 77, 39, 20, 13, 95, 72. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 24, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 72, 87, 58.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Продовження додатка 4

Варіант №13

1) Створіть за допомогою *С*-програми ДДП, що складається з наступних вузлів: 25, 2, 16, 22, 28, 4, 36, 54, 19, 78, 17, 55, 24, 72, 53. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 16, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 17, 54, 25.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Варіант №14

1) Створіть за допомогою *С*-програми ДДП, що складається з наступних вузлів: 38, 2, 26, 98, 33, 22, 15, 75, 14, 68, 18, 27, 29, 35, 88. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 14, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 29, 15, 38.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Варіант №15

1) Створіть за допомогою *С*-програми ДДП, що складається з наступних вузлів: 37, 11, 45, 43, 9, 1, 58, 76, 48, 27, 32, 98, 29, 35, 81. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 48, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 81, 32, 37.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Продовження додатка 4

Варіант №16

1) Створіть за допомогою *C*-програми ДДП, що складається з наступних вузлів: 44, 12, 51, 25, 33, 75, 54, 8, 19, 58, 77, 16, 49, 7, 63. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 75, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 63, 25, 44.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Варіант №17

1) Створіть за допомогою *C*-програми ДДП, що складається з наступних вузлів: 45, 35, 67, 83, 12, 33, 29, 55, 17, 23, 87, 71, 50, 13, 8. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 55, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 23, 33, 45.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Варіант №18

1) Створіть за допомогою *C*-програми ДДП, що складається з наступних вузлів: 41, 34, 76, 87, 22, 77, 11, 71, 36, 54, 13, 98, 2, 40, 15. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 71, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 15, 22, 41.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Продовження додатка 4

Варіант №19

1) Створіть за допомогою *С*-програми ДДП, що складається з наступних вузлів: 67, 22, 36, 78, 21, 65, 72, 98, 44, 1, 17, 87, 70, 34, 64. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 72, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 64, 36, 67.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Варіант №20

1) Створіть за допомогою *С*-програми ДДП, що складається з наступних вузлів: 51, 33, 98, 37, 87, 38, 91, 8, 4, 32, 86, 64, 34, 69, 11. Порядок додавання вузлів повинен зберігатися, перший елемент – корінь.

2) Визначте в дереві (програмно) максимальний і мінімальний елементи.

3) Нарисуйте в робочому зошиті двійкове дерево пошуку з зазначенням в ньому максимального і мінімального вузлів та позначте відповідні рівні ДДП.

4) Знайдіть програмно батьківський вузол для вузла 37, та позначте його на двійковому дереві в зошиті.

5) Видаліть за допомогою відповідної функції вузли у визначеному порядку: 69, 87, 51.

6) Нарисуйте в зошиті вигляд ДДП після кожного видаленого вузла.

Навчальне видання

Методичні вказівки
до виконання практичних робіт з дисципліни

«Програмування»

для студентів денної та заочної форм навчання за спеціальністю
123 «Комп'ютерна інженерія»
Частина 2

Укладачі: ОНИЩЕНКО Валерій Валентинович
СТАТКУС Андрій Віталійович
НОСИК Андрій Михайлович
КОРОЛЬОВА Яна Юріївна

Відповідальний за випуск проф. Порошин С. М.
Роботу рекомендував до друку проф. Заполовський М. Й.

В авторській редакції

План 2021 р., поз. 262

Підп. до друку 6.12.2021 р. Формат 60×84 1/16. Папір офсетний.
RISO-друк. Гарнітура Тайм
Ум. друк. арк. 9.8
Наклад 50 прим. Замовлення № . Ціна договірна

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
61002, Харків, вул. Кирпичова, 2

Видавництво та друк ФО-П Єфименко С.А.
61166, Україна, м. Харків, вул. Коломенська, 27
Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру видавців – виготовлювачів і розповсюджувачів видавничої продукції
ДК № 6869 від 08.08.2019 р.