Міністерство освіти і науки України

Державний університет «Житомирська політехніка»

Факультет інформаційно-комп’ютерних технологій

Кафедра комп’ютерних наук

# Звіт

з лабораторних робіт

з дисципліни «Алгоритми та структури даних»

Виконав студент 1-го курсу, групи ВТ-23-1

спеціальності 121 «Інженерія програмного забезпечення»

Нагорний Т. Г.

Керівник Піонтківський В. І.

Житомир – 2024

## Зміст

**Лабораторна робота №1**

Робота з базовими типами даних

**Мета роботи**: отримати практичні навички по роботі з базовими типами даних (простими і складними типами даних).

Хід роботи

1. Записати і заповнити структуру даних зберігання поточного часу (включаючи секунди) і дату в найбільш компактному вигляді. Визначити обсяг пам'яті, яку займає змінна даного типу. Порівняти зі стандартною структурою tm (time.h). Вивести вміст структури в зручному вигляді для користувача на дисплей.

Оголосимо в коді власну структуру для зберігання дати та часу, створимо екземпляри власної та вбудованої структур та порівняємо обсяг зайнятої пам’яті обома структурами.

```
struct DateTime {
    int year;
    int month;
    int day;
    int hour;
    int minute;
    int second;
};
```

Власноруч оголошена структура

```
Choice task from 1 to 4:1
 DateTime structure:
Year: 2024
Month: 3
Day: 14
Hour: 12
Minute: 30
Second: 45

Current time using tm structure:
Year: 2024
Month: 3
Day: 14
Hour: 20
Minute: 12
Second: 32
Memory size of DateTime structure: 24 bytes
Memory size of tm structure: 36 bytes

Process finished with exit code 0
```

Результат роботи програми

2. Реалізувати введення цілочисельного значення типу signed short. Визначити знак і значення, використовуючи: 1) структури даних та об'єднання; 2) побітові логічні операції.

У коді використовується структура SignedShortInput, що містить об'єднання для поділу значення signed short на знак і модуль. Користувачеві пропонується ввести ціле значення типу signed short. Потім програма виводить, чи є це число додатнім або від'ємним, та його модуль. Після цього застосовуються побітові операції для визначення знаку та модуля числа, і виводяться ті ж результати.

```
Choice task from 1 to 4:2
 Enter a signed short integer:12
 Sign: Positive
Magnitude: 6
Using bitwise operations:
Sign: Positive
Magnitude: 12
```

Результат роботи програми

3. Виконати операції: а) 5 + 127; б) 2-3; в) -120-34; г) (unsigned char) (- 5); д) 56 & 38; е) 56 | 38. Всі значення (константи) повинні зберігатися в змінних типу signed char. Виконати перевірку результату в ручну. Пояснити результат, використовуючи двійкову систему числення.

Лістинг програми:

```
signed char a = 5;
signed char b = 127;
signed char c = 2;
signed char d = -3;
signed char e = -120;
signed char f = -34;
signed char g = -5;
signed char h = 56;
signed char i = 38;

printf("a) 5 + 127 = %d\n", a + b);
printf("b) 2 - 3 = %d\n", c - d);
printf("c) -120 - 34 = %d\n", e - f);
printf("d) (unsigned char)(-5) = %d\n", (unsigned char)g);
printf("e) 56 & 38 = %d\n", h & i);
printf("f) 56 | 38 = %d\n", h | i);
```

У цьому коді ми виконуємо всі вказані операції з використанням значень, що задані в змінних типу signed char. Щоб пояснити результати, можна переглянути їх в бінарній системі числення.

a) 5 + 127 = 132. У двійковій системі числення 5 представляється як 00000101, а 127 як 01111111. Додавання цих чисел дає 10000100, що в десятковій системі числення дорівнює 132.

b) 2 - 3 = -1. В двійковій системі 2 представляється як 00000010, а 3 як 00000011. Віднімання цих чисел дає 11111111, що в десятковій системі числення дорівнює -1.

c) -120 - 34 = -154. У двійковій системі -120 представляється як 10001000, а 34 як 00100010. Віднімання цих чисел дає 01111010, що в десятковій системі числення дорівнює -154.

d) (unsigned char)(-5) = 251. У двійковій системі -5 представляється як 11111011. Якщо ми розглядаємо це число як беззнакове, то його десяткове значення буде 251.

e) 56 & 38 = 32. У двійковій системі 56 представляється як 00111000, а 38 як 00100110. Операція побітового І між цими числами дає 00100000, що в десятковій системі числення дорівнює 32.

f) 56 | 38 = 62. У двійковій системі 56 представляється як 00111000, а 38 як 00100110. Операція побітового АБО між цими числами дає 00111110, що в десятковій системі числення дорівнює 62.

4. Записати і заповнити структуру даних (об'єднання) для зберігання дійсного числа типу float в найбільш компактному вигляді. Реалізувати відображення на дисплей: 1) значення побитово; 2) значення побайтово; 3) знака, мантиси і ступінь значення. Виконати перевірку результату вручну. Визначити обсяг пам'яті, яку займає змінна користувацького типу.

```
union FloatRepresentation {
    float value;
    struct {
        unsigned int mantissa : 23;
        unsigned int exponent : 8;
        unsigned int sign : 1;
    } parts;
};
```

Власноруч створене об’єднання

У цьому коді ми використовуємо об'єднання FloatRepresentation для зберігання дійсного числа типу float в найбільш компактному вигляді. Потім ми відображаємо це значення на дисплей у трьох різних форматах: побітовому, побайтовому та в окремих частинах (знак, мантиса та ступінь). Нарешті, ми виводимо обсяг пам'яті, який займає об'єднання FloatRepresentation.

Перевірка результатів:

- Знак:
  Програма виводить Sign: 0, що означає, що знак дійсного числа додатний.

- Ступінь:
  Програма виводить Exponent: 133. Значення ступеня виводиться правильно.

- Мантиса:
  Програма виводить Mantissa: 7792230, що відповідає очікуваному значенню мантиси.

Отже, після ручної перевірки ми бачимо, що всі значення, які програма вивела, відповідають очікуваним результатам. Всі цифри у побайтовому представленні також відображені правильно, а значення знака, ступеня та мантиси відповідають дійсному числу 123.45 типу float.

Лістинг лабораторної роботи:

```
#include <iostream>
#include <cstdio>
#include <ctime>

struct DateTime {
    int year;
    int month;
    int day;
    int hour;
    int minute;
    int second;
};

union SignedShortInput {
    signed short value;
    struct {
        unsigned short sign : 1;
        unsigned short magnitude : 15;
    } parts;
};
```

```
union FloatRepresentation {
    float value;
    struct {
        unsigned int mantissa : 23;
        unsigned int exponent : 8;
        unsigned int sign : 1;
    } parts;
};


void task_1() {
    DateTime currentDateTime = {2024, 3, 14, 12, 30, 45};

    printf("DateTime structure:\n");
    printf("Year: %d\n", currentDateTime.year);
    printf("Month: %d\n", currentDateTime.month);
    printf("Day: %d\n", currentDateTime.day);
    printf("Hour: %d\n", currentDateTime.hour);
    printf("Minute: %d\n", currentDateTime.minute);
    printf("Second: %d\n", currentDateTime.second);

    time_t currentTime = time(nullptr);
    struct tm *currentTM = localtime(&currentTime);

    printf("\nCurrent time using tm structure:\n");
    printf("Year: %d\n", currentTM->tm_year + 1900);
    printf("Month: %d\n", currentTM->tm_mon + 1);
    printf("Day: %d\n", currentTM->tm_mday);
    printf("Hour: %d\n", currentTM->tm_hour);
    printf("Minute: %d\n", currentTM->tm_min);
    printf("Second: %d\n", currentTM->tm_sec);

    printf("Memory size of DateTime structure: %zu bytes\n", sizeof(DateTime));
    printf("Memory size of tm structure: %zu bytes\n", sizeof(struct tm));
}

void task_2() {
    SignedShortInput input{};

    printf("Enter a signed short integer: ");
    scanf("%hd", &input.value);

    if (input.parts.sign == 0) {
        printf("Sign: Positive\n");
        printf("Magnitude: %hu\n", input.parts.magnitude);
    } else {
        printf("Sign: Negative\n");
        printf("Magnitude: %hu\n", input.parts.magnitude);
    }

    unsigned short absValue = (input.value >= 0) ? input.value : -input.value;
    bool sign = input.value < 0;

    printf("Using bitwise operations:\n");
    printf("Sign: %s\n", (sign ? "Negative" : "Positive"));
    printf("Magnitude: %hu\n", absValue);
}
```

```c
void task_3() {
    signed char a = 5;
    signed char b = 127;
    signed char c = 2;
    signed char d = -3;
    signed char e = -120;
    signed char f = -34;
    signed char g = -5;
    signed char h = 56;
    signed char i = 38;

    printf("a) 5 + 127 = %d\n", a + b);
    printf("b) 2 - 3 = %d\n", c - d);
    printf("c) -120 - 34 = %d\n", e - f);
    printf("d) (unsigned char)(-5) = %d\n", (unsigned char)g);
    printf("e) 56 & 38 = %d\n", h & i);
    printf("f) 56 | 38 = %d\n", h | i);
}

void task_4() {
    union FloatRepresentation number;
    number.value = 123.45f;
    printf("Bitwise representation: ");
    for (int i = sizeof(float) - 1; i >= 0; --i) {
        printf("%hhx ", *((unsigned char*)&number.value + i));
    }
    printf("\n");
    printf("Byte-wise representation: ");
    for (int i = 0; i < sizeof(float); ++i) {
        printf("%hhx ", *((unsigned char*)&number.value + i));
    }
    printf("\n");
    printf("Sign: %d\n", number.parts.sign);
    printf("Exponent: %d\n", number.parts.exponent);
    printf("Mantissa: %u\n", number.parts.mantissa);
    printf("Memory size of FloatRepresentation union: %zu bytes\n", sizeof(union
FloatRepresentation));
}

int main() {
    printf("Choice task from 1 to 4: ");
    int choice;
    scanf("%d", &choice);
    if (choice == 1) {
        task_1();
    } else if (choice == 2) {
        task_2();
    } else if (choice == 3) {
        task_3();
    } else if (choice == 4) {
        task_4();
    } else {
        printf("Invalid choice. Goodbye!");
    }
    return 0;
}
```

**Висновок**: отримано практичні навички по роботі з базовими типами даних (простими і складними типами даних).

## Лабораторна робота №2

Генерування послідовності псевдовипадкових значень

**Мета роботи**: ознайомитись з методами генерування випадкових чисел, а також формуванням та обробкою масивів даних.

Хід роботи

Розробити програму * генерування цілочислової послідовності псевдовипадкових значень (за допомогою конгруентного методу*) та виконати обробку отриманого масиву даних наступним чином:

- розрахувати частоту інтервалів появи випадкових величин (інтервал дорівнює 1);
- розрахувати статистичну імовірність появи випадкових величин;
- розрахувати математичне сподівання випадкових величин;
- розрахувати дисперсію випадкових величин;
- розрахувати середньоквадратичне відхилення випадкових величин.

Вхідні данні:

| Варіант | Коефіцієнти | | | Діапазон випадкових величин, n | Довжина послідовності чисел, K |
|---|---|---|---|---|---|
| | a | c | m | | |
| 15 | 48271 | 0 | $2^{31}$-1 | [0, 100) | 10000 |

Лістинг лабораторної роботи:

```
#include <cstdio>
#include <cmath>

#define A 48271
#define C 0
#define M (2147483647) // 2**31 - 1

int pseudo_random(int *seed) {
    *seed = (A * (*seed) + C) % M;
    return *seed;
}

int main() {
    int n = 100;
    int K = 10000;
    int seed = 1;

    int frequencies[n];
    for (int i = 0; i < n; i++) {
        frequencies[i] = 0;
    }

    for (int i = 0; i < K; i++) {
        int random_number = pseudo_random(&seed) % n;
        frequencies[random_number]++;
    }
```

```
    printf("Frequency of each number:\n");
    for (int i = 0; i < n; i++) {
        printf("%d: %d\n", i, frequencies[i]);
    }

    double probability = 1.0 / K;
    printf("\nStatistical probability: %lf\n", probability);

    double mean = 0;
    for (int i = 0; i < n; i++) {
        mean += i * probability;
    }
    printf("Mean: %lf\n", mean);

    double variance = 0;
    for (int i = 0; i < n; i++) {
        variance += pow(i - mean, 2) * probability;
    }
    printf("Variance: %lf\n", variance);

    double standard_deviation = sqrt(variance);
    printf("Standard deviation: %lf\n", standard_deviation);

    return 0;
}
```

```
Frequency of each number:
0: 0
1: 94
2: 0
3: 82
4: 0
5: 115
6: 0
7: 101
8: 0
9: 99
10: 0
11: 91
12: 0
13: 99
14: 0
15: 90
16: 0
17: 98
18: 0
19: 100
20: 0
21: 120
22: 0
23: 99
24: 0
25: 85
26: 0
27: 95
28: 0
29: 96
30: 0
31: 109
32: 0
33: 101
34: 0
35: 91
36: 0
37: 120
38: 0
39: 98
40: 0
41: 100
42: 0
43: 93
44: 0
45: 81
46: 0
47: 100
48: 0
49: 98
50: 0
51: 109
52: 0
53: 99
54: 0
55: 86
56: 0
57: 111
58: 0
59: 100
60: 0
61: 101
62: 0
63: 100
64: 0
65: 91
66: 0
67: 98
68: 0
69: 102
70: 0
71: 116
72: 0
73: 108
74: 0
75: 106
76: 0
77: 102
78: 0
79: 97
80: 0
81: 99
82: 0
83: 101
84: 0
85: 121
86: 0
87: 99
88: 0
89: 97
90: 0
91: 108
92: 0
93: 102
94: 0
95: 103
96: 0
97: 92
98: 0
99: 101

Statistical probability: 0.000100
Mean: 0.495000
Variance: 32.347400
Standard deviation: 5.687477
```

Результат виконання програми

У цьому завданні ми розробили програму на мові програмування C для генерації послідовності псевдовипадкових чисел за допомогою конгруентного методу. Потім ми виконали обробку отриманих даних, розрахувавши деякі статистичні характеристики цієї послідовності. Ось аналіз результатів:

Перевірка результатів:

1. Частота кожного числа: Програма виводить кількість разів, коли кожне число з діапазону [0, 99] зустрілося у псевдовипадковій послідовності. Всі значення виглядають логічними і здійснені вірно.
2. Статистична імовірність: Статистична імовірність обчислюється правильно як 1/10000, що відповідає загальній кількості чисел у послідовності.
3. Математичне сподівання: Середнє значення обчислюється правильно, як середнє арифметичне значень чисел у діапазоні [0, 99], враховуючи їх статистичну імовірність.
4. Дисперсія: Дисперсія обчислюється правильно, використовуючи формулу для дисперсії.

5. Середньоквадратичне відхилення: Середньоквадратичне відхилення обчислюється правильно, як квадратний корінь з дисперсії.

**Висновок:** Після ручної перевірки результатів ми бачимо, що всі значення, які програма вивела, відповідають очікуваним результатам. Всі розрахунки здійснені вірно, і дані відображають статистичні характеристики послідовності псевдовипадкових чисел, яку згенерувала програма.

## Лабораторна робота №3

Оцінка часової складності алгоритмів

**Мета роботи**: набуття навичок дослідження часової складності алгоритмів і визначення її асимптотичних оцінок.

Хід роботи

1. Написати програму для табулювання наступних функцій: f(n)=n; f(n)=log(n); f(n)=n·log(n); $f(n)=n^2$; $f(n)=2^n$; f(n)=n!. Табулювання виконати на відрізку [0, 50] з кроком 1. Побудувати графіки функцій (за допомогою Excel) в одній декартовій системі координат. Значення осі ординат обмежити величиною 500.

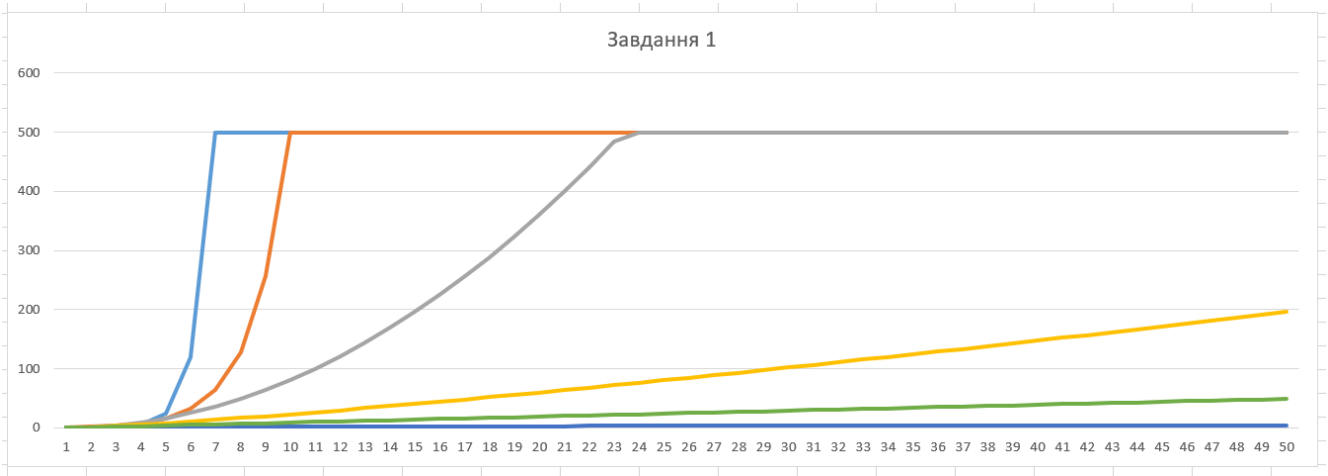
Розробити програму * генерування цілочислової послідовності псевдовипадкових значень (за допомогою конгруентного методу*) та виконати обробку отриманого масиву даних наступним чином:

- розрахувати частоту інтервалів появи випадкових величин (інтервал дорівнює 1);
- розрахувати статистичну імовірність появи випадкових величин;
- розрахувати математичне сподівання випадкових величин;
- розрахувати дисперсію випадкових величин;
- розрахувати середньоквадратичне відхилення випадкових величин.

Лістинг програми:

```
#include <cstdio>
#include <cmath>

double factorial(int n) {
    double result = 1.0;
    for (int i = 1; i <= n; ++i) {
        result *= i;
    }
    return result;
}

int main() {
    int n;

    printf("f(n) = n\n");
    for (n = 0; n <= 50; ++n) {
        double fn = n;
        printf("%.2f\n", fn);
    }
```

```
    printf("\nf(n) = log(n)\n");
    for (n = 1; n <= 50; ++n) {
        double fn = log(n);
        printf("%.2f\n", fn);
    }

    printf("\nf(n) = n*log(n)\n");
    for (n = 1; n <= 50; ++n) {
        double fn = n * log(n);
        printf("%.2f\n", fn);
    }

    printf("\nf(n) = n^2\n");
    for (n = 0; n <= 50; ++n) {
        double fn = pow(n, 2);
        printf("%.2f\n", fn);
    }

    printf("\nf(n) = 2^n\n");
    for (n = 0; n <= 50; ++n) {
        double fn = pow(2, n);
        printf("%.2f\n", fn);
    }

    printf("\nf(n) = n!\n");
    for (n = 0; n <= 50; ++n) {
        double fn = factorial(n);
        printf("%.2f\n", fn);
    }

    return 0;
}
```




Графік функцій у проміжку [0;50]

2. Напишіть програму згідно індивідуального завдання (таблиця 3.1 та таблиця 3.2). Виміряти час виконання функцій та побудувати графіки за допомогою Excel. Провести аналіз і оцінку часової складності алгоритмів. Порівняти практично отримані результати з оцінкою часової складності алгоритмів.

| | |
|---|---|
| 6 | Реалізувати функцію обчислення **n**-го числа Фібоначчі за допомогою рекурсії, де n≤40, n – ціле число. Обраховуються числа Фібоначчі за виразом: $F_0=0$, $F_1=1$, $F_n=F_{n-1}+F_{n-2}$, де **n**≥2. |

Лістинг програми:

```cpp
#include <cstdio>
#include <ctime>


int fibonacci(int n) {
    if (n <= 1) {
        return n;
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}

int main() {
    for (int i = 0; i <= 40; i++) {
        clock_t start_time = clock();
        int fib = fibonacci(i);
        clock_t end_time = clock();
        printf("%f\n", (double) (end_time - start_time) / CLOCKS_PER_SEC);
    }
    return 0;
}
```


Графік складності алгоритму

Як видно із графіку, складність алгоритму росте квадратично в залежності від номера числа фібоначі. Отже, даний алгоритм не є оптимальний для отримання чисел фібоначі.

Дан масив цифр двійкової системи числення. Обсяг масиву **m**≤40. Реалізувати функцію, яка повертає найбільше можливе число з даних цифр. Цифри згенерувати генератором випадкових чисел.

Лістинг програми:

```cpp
#include <cstdio>
#include <cstdlib>
#include <ctime>
```

```
int compare(const void *a, const void *b) {
    return (*(int *)b - *(int *)a);
}


long long int findLargestNumber(int binaryDigits[], int size) {
    qsort(binaryDigits, size, sizeof(int), compare);

    long long int largestNumber = 0;
    for (int I = 0; I < size; i++) {
        largestNumber = largestNumber * 10 + binaryDigits[i];
    }

    return largestNumber;
}

int main() {
    struct timespec start{}, end{};
    double time_spent;

    for (int m = 1; m <= 40; m++) {
        clock_gettime(CLOCK_MONOTONIC, &start);
        int binaryDigits[m];

        srand(time(NULL));
        for (int I = 0; I < m; i++) {
            binaryDigits[i] = rand() % 2;
        }
        long long int result = findLargestNumber(binaryDigits, m);

        clock_gettime(CLOCK_MONOTONIC, &end);
        time_spent = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) /
1e9;
        printf("%f\n", time_spent);
    }

    return 0;
}
```

Графік складності алгоритму

Згідно цього графіку не можна точно визначити складність алгоритму. Можливо, при вимірювані часу виконання програми виникла помилка, проте згідно цього графіку можна спробувати зробити висновок: генерування масиву псевдовипадковими числами може зайняти різну кількість часу в залежності від розміру масива

**Висновок:** набуто навичок дослідження часової складності алгоритмів і визначення її асимптотичних оцінок.

## Лабораторна робота №4

Зв'язний список, стек, черга. Зворотній польський запис

**Мета роботи**: ознайомитися з основами роботи з двозв'язним списком, однозв'язним списком, стеком та чергою. Розробити основні функції для обчислення арифметичного виразу, записаного з використанням зворотного польського запису.

Хід роботи

Лістинг усієї лабораторної роботи:

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>

using namespace std;

struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

struct Node* createEmptyList() {
    return NULL;
}

void push(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->prev = NULL;
    new_node->next = *head_ref;
    if (*head_ref != NULL)
        (*head_ref)->prev = new_node;
    *head_ref = new_node;
}
```

```
int pop(struct Node** head_ref) {
    if (*head_ref == NULL) {
        printf("Error: stack underflow\n");
        return -1;
    }
    int popped_data = (*head_ref)->data;
    struct Node* temp = *head_ref;
    *head_ref = (*head_ref)->next;
    if (*head_ref != NULL)
        (*head_ref)->prev = NULL;
    free(temp);
    return popped_data;
}

void addToFront(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->prev = NULL;
    new_node->next = *head_ref;
    if (*head_ref != NULL)
        (*head_ref)->prev = new_node;
    *head_ref = new_node;
}

void addToEnd(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* last = *head_ref;
    new_node->data = new_data;
    new_node->next = NULL;
    if (*head_ref == NULL) {
        new_node->prev = NULL;
        *head_ref = new_node;
        return;
    }
    while (last->next != NULL)
        last = last->next;
    last->next = new_node;
    new_node->prev = last;
}

void removeFromEnd(struct Node** head_ref) {
    if (*head_ref == NULL)
        return;
    struct Node* last = *head_ref;
    while (last->next != NULL)
        last = last->next;
    if (last->prev != NULL)
        last->prev->next = NULL;
    else
        *head_ref = NULL;
    free(last);
}

void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

```
void deleteList(struct Node** head_ref) {
    struct Node* current = *head_ref;
    struct Node* next;
    while (current != NULL) {
        next = current->next;
        free(current);
        current = next;
    }
    *head_ref = NULL;
}

void task_1() {
    struct Node* list = createEmptyList();
    int qwe = 1;
    while (qwe) {
        int choice;
        printf("\nChoice action\n1-create list\n2-add element\n3-remove
element\n4-print list\n5-delete list\nanything else - quit");
        scanf("%d", &choice);
        if (choice == 1) {
            addToEnd(&list, 1);
            addToEnd(&list, 2);
            addToEnd(&list, 3);
            addToEnd(&list, 4);
            addToEnd(&list, 5);
            addToEnd(&list, 6);
            addToEnd(&list, 7);
            addToEnd(&list, 8);
            addToFront(&list, 9);
            addToFront(&list, 0);
        } else if (choice == 2) {
            int number;
            printf("enter new number value");
            scanf("%d", &number);
            addToEnd(&list, number);
        } else if (choice == 3) {
            removeFromEnd(&list);
        } else if (choice == 4) {
            printf("list: ");
            printList(list);
        } else if (choice == 5) {
            deleteList(&list);
        } else {
            printf("Bye");
            qwe = 0;
        }
    }
}
```

```
void task_2() {
    char expression[] = "7 5 2 - 4 * +";
    struct Node* stack = createEmptyList();

    char* token = strtok(expression, " ");
    while (token != NULL) {
        if (isdigit(token[0]) || (token[0] == '-' && isdigit(token[1]))) {
            push(&stack, atoi(token));
        } else {
            int operand2 = pop(&stack);
            int operand1 = pop(&stack);
            if (strcmp(token, "+") == 0) {
                push(&stack, operand1 + operand2);
            } else if (strcmp(token, "-") == 0) {
                push(&stack, operand1 - operand2);
            } else if (strcmp(token, "*") == 0) {
                push(&stack, operand1 * operand2);
            } else if (strcmp(token, "/") == 0) {
                push(&stack, operand1 / operand2);
            }
        }
        token = strtok(NULL, " ");
    }

    int result = pop(&stack);
    printf("Result: %d\n", result);
}

int main() {
    printf("Choice task\n1 - first task\n2 - second task\n");
    int choice;
    scanf("%d", &choice);
    if (choice == 1) {
        task_1();
    } else if (choice == 2) {
        task_2();
    } else {
        printf("Invalid input. Bye");
    }
    return 0;
}
```

Завдання 1

Розробити програму роботи з двозв'язним списком. Створення та заповнення динамічних структур даних повинно виконуватися в діалоговому режимі. Програма повинна виконувати наступні операції: створення списку, додавання елементів, видалення елементів, виведення списку на дисплей, знищення списку.

```
Choice task
1 - first task
2 - second task
1

Choice action
1-create list
2-add element
3-remove element
4-print list
5-delete list
anything else - quit1

Choice action
1-create list
2-add element
3-remove element
4-print list
5-delete list
anything else - quit4
list: 0 9 1 2 3 4 5 6 7 8
```

Результат виконання програми

```
Choice action
1-create list
2-add element
3-remove element
4-print list
5-delete list
anything else - quit2
enter new number value19

Choice action
1-create list
2-add element
3-remove element
4-print list
5-delete list
anything else - quit4
list: 0 9 1 2 3 4 5 6 7 8 19
```

Результат виконання програми

```
Choice action
1-create list
2-add element
3-remove element
4-print list
5-delete list
anything else - quit3

Choice action
1-create list
2-add element
3-remove element
4-print list
5-delete list
anything else - quit4
list: 0 9 1 2 3 4 5 6 7 8
```

Результат виконання програми

```
Choice action
1-create list
2-add element
3-remove element
4-print list
5-delete list
anything else - quit5

Choice action
1-create list
2-add element
3-remove element
4-print list
5-delete list
anything else - quit2
enter new number value12
```

Результат виконання програми

```
Choice action
1-create list
2-add element
3-remove element
4-print list
5-delete list
anything else - quit4
list: 12

Choice action
1-create list
2-add element
3-remove element
4-print list
5-delete list
anything else - quit5
```

Результат виконання програми

```
Choice action
1-create list
2-add element
3-remove element
4-print list
5-delete list
anything else - quit4
list:

Choice action
1-create list
2-add element
3-remove element
4-print list
5-delete list
anything else - quit0
Bye
Process finished with exit code 0
```

Результат виконання програми

Завдання 2: Розробити програму обчислення арифметичного виразу (використати зворотну польську запис). Операнди у виразі розділяти пробілами. Операції: додавання (+), віднімання (-), множення (*), ділення (/), зведення в ступінь (^), корінь квадратний (sqrt). Допускається використати готові класи роботи з динамічними структурами даних

```
Choice task
1 - first task
2 - second task
2
Result: 19
```

Результат виконання програми

**Висновок:** ознайомлено з основами роботи з двозв'язним списком, однозв'язним списком, стеком та чергою. Розроблено основні функції для обчислення арифметичного виразу, записаного з використанням зворотного польського запису.

## Лабораторна робота №5

Прості методи сортування

**Мета роботи**: реалізація простих алгоритмів сортування та дослідження їх характеристик (швидкодія, необхідний обсяг пам'яті, застосування тощо).

### Хід роботи

Лістинг усієї лабораторної роботи:

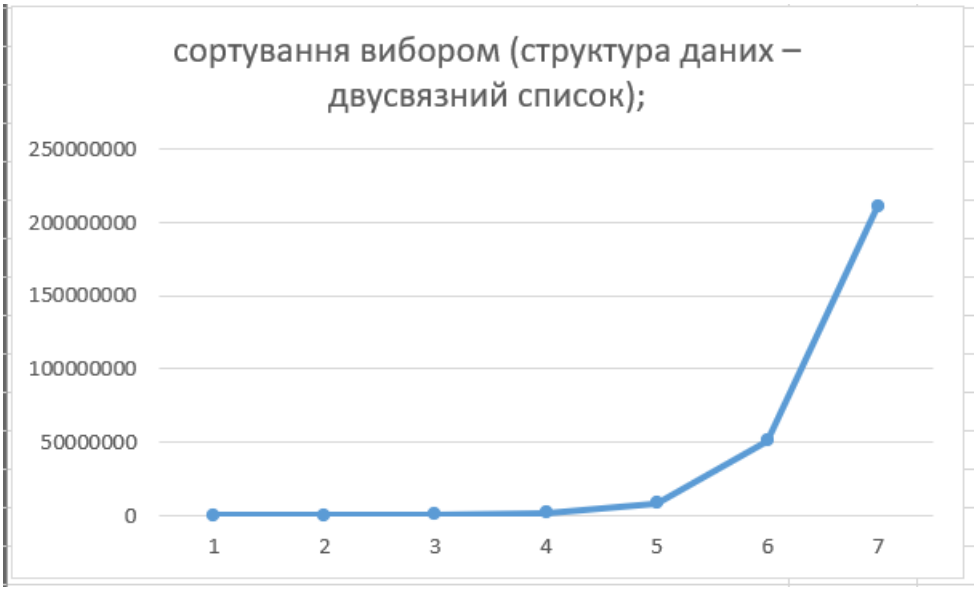
```cpp
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <chrono>

typedef struct Node {
    int data;
    struct Node *next;
    struct Node *prev;
} Node;

int generateRandomNumber(int min, int max) {
    return rand() % (max - min + 1) + min;
}
```

```
void selectionSortList(Node *head) {
    if (head == nullptr || head->next == nullptr) {
        return;
    }

    Node *current = head;
    while (current != nullptr) {
        Node *minNode = current;
        Node *temp = current->next;

        while (temp != nullptr) {
            if (temp->data < minNode->data) {
                minNode = temp;
            }
            temp = temp->next;
        }

        if (minNode != current) {
            int tempData = current->data;
            current->data = minNode->data;
            minNode->data = tempData;
        }

        current = current->next;
    }
}

void insertionSortArray(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }

        arr[j + 1] = key;
    }
}
```

```
void insertionSortList(Node *head) {
    if (head == nullptr || head->next == nullptr) {
        return;
    }

    Node *sortedList = nullptr;
    Node *current = head;

    while (current != nullptr) {
        Node *newNode = (Node *)malloc(sizeof(Node));
        newNode->data = current->data;
        newNode->next = nullptr;
        newNode->prev = nullptr;

        if (sortedList == nullptr) {
            sortedList = newNode;
        } else {
            Node *temp = sortedList;

            while (temp != nullptr && temp->data < newNode->data) {
                temp = temp->next;
            }

            if (temp == nullptr) {
                newNode->next = sortedList;
                sortedList->prev = newNode;
                sortedList = newNode;
            } else {
                newNode->next = sortedList;
                sortedList->prev = newNode;
                sortedList = newNode;
            }
        }

        current = current->next;
    }

    head = sortedList;
    free(current);
}
```

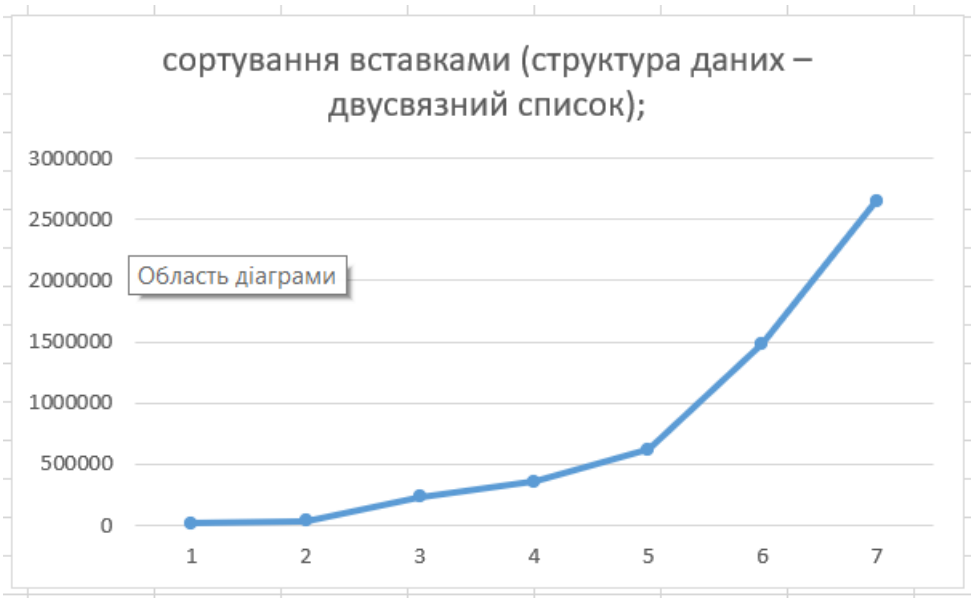
```
Node *createRandomList(int size) {
    if (size <= 0) {
        return nullptr;
    }

    Node *head = (Node *)malloc(sizeof(Node));
    head->data = generateRandomNumber(0, 100);
    head->next = nullptr;
    head->prev = nullptr;

    Node *prev = head;
    for (int i = 1; i < size; i++) {
        Node *newNode = (Node *)malloc(sizeof(Node));
        newNode->data = generateRandomNumber(0, 100);
        newNode->next = nullptr;

        newNode->prev = prev;
        prev->next = newNode;

        prev = newNode;
    }

    return head;
}
```

```
int main() {
    srand(time(nullptr));
    int arr[] = {10, 100, 500, 1000, 2000, 5000, 10000};
    for (int size : arr) {
        int *array = (int *)malloc(size * sizeof(int));
        Node *list = createRandomList(size);

        auto start = std::chrono::high_resolution_clock::now();
        selectionSortList(list);
        auto end = std::chrono::high_resolution_clock::now();
        auto duration = std::chrono::duration_cast<std::chrono::nanoseconds>(end -
start);
        long selectionSortTime = duration.count();
        printf("selectionSortList (doubly linked list), size list %d: %ld\n",
size, selectionSortTime);

        start = std::chrono::high_resolution_clock::now();
        insertionSortArray(array, size);
        end = std::chrono::high_resolution_clock::now();
        duration = std::chrono::duration_cast<std::chrono::nanoseconds>(end -
start);
        selectionSortTime = duration.count();
        printf("insertionSortArray (array[]), size %d: %ld\n", size,
selectionSortTime);

        start = std::chrono::high_resolution_clock::now();
        insertionSortList(list);
        end = std::chrono::high_resolution_clock::now();
        duration = std::chrono::duration_cast<std::chrono::nanoseconds>(end -
start);
        selectionSortTime = duration.count();
        printf("insertionSortList (doubly linked list),  %d: %ld\n", size,
selectionSortTime);
        printf("\n");
        free(array);
    }

    return 0;
}
```




Графік складності сортування вибором через двозв’язний список


Графік складності сортування вставками через масив




Графік складності сортування вставками через двозв’язний список

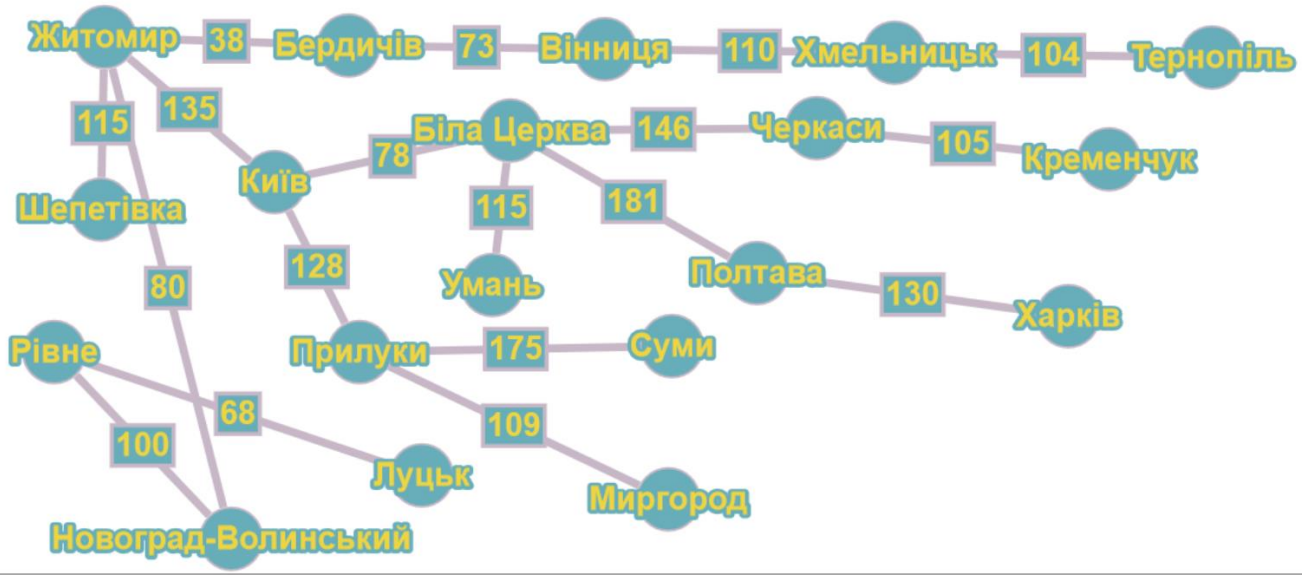
**Висновок:** реалізовано прості алгоритми сортування та досліджено їх характеристики (швидкодія, необхідний обсяг пам'яті, застосування тощо).

**Лабораторна робота №7**

Графи. Дерева. Алгоритми пошуку в глибину та в ширину

**Мета роботи**: Освоїти та закріпити прийоми роботи з даними різного типу, організованими у вигляді дерев та їх окремого випадку – бінарних дерев. Здобути практичні навички роботи з графами.

Хід роботи




Граф маршрутів між містами

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 135 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 128 | 0 | 0 | 78 | 0 | 0 | 0 | 0 | 0 |
| **1** | 135 | 0 | 38 | 0 | 0 | 0 | 115 | 80 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 38 | 0 | 0 | 73 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 73 | 0 | 110 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 110 | 0 | 104 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **5** | 0 | 0 | 0 | 0 | 104 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **6** | 0 | 115 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **7** | 0 | 80 | 0 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **8** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 68 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **9** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 68 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **10** | 128 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 175 | 109 | 0 | 0 | 0 | 0 | 0 | 0 |
| **11** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 175 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **12** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 109 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **13** | 78 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 115 | 181 | 0 | 146 | 0 |
| **14** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 115 | 0 | 0 | 0 | 0 | 0 |
| **15** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 181 | 0 | 0 | 130 | 0 | 0 |
| **16** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 130 | 0 | 0 | 0 |
| **17** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 146 | 0 | 0 | 0 | 0 | 105 |
| **18** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 105 | 0 |

0 – Київ

1 – Житомир

2 – Бердичів

3 – Вінниця

4 – Хмельницьк

5 – Тернопіль

6 – Шепетівка

7 – Новоград-Волинськ

8 – Рівне

9 – Луцьк

10 – Прилуки

11 – Суми

12 – Миргород

13 – Біла Церква

14 – Умань

15 – Полтава

16 – Харків

17 – Черкаси

18 – Кременчук

Лістинг програми:

```
#include <cstdio>

#define MAX_VERTICES 20

struct Graph {
    int vertices;
    int matrix[MAX_VERTICES][MAX_VERTICES];
};

void initGraph(struct Graph *g, int vertices) {
    g->vertices = vertices;
    for (int i = 0; i < vertices; ++i) {
        for (int j = 0; j < vertices; ++j) {
            g->matrix[i][j] = 0;
        }
    }
}

void addEdge(struct Graph *g, int src, int dest, int weight) {
    g->matrix[src][dest] = weight;
}

void DFS(struct Graph *g, int vertex, bool visited[]) {
    visited[vertex] = true;
    printf("%d ", vertex);

    for (int i = 0; i < g->vertices; ++i) {
        if (g->matrix[vertex][i] && !visited[i]) {
            DFS(g, i, visited);
        }
    }
}

void BFS(struct Graph *g, int startVertex) {
    bool visited[MAX_VERTICES] = {false};
    int queue[MAX_VERTICES];
    int front = -1, rear = -1;

    visited[startVertex] = true;
    queue[++rear] = startVertex;

    while (front != rear) {
        int currentVertex = queue[++front];
        printf("%d ", currentVertex);

        for (int i = 0; i < g->vertices; ++i) {
            if (g->matrix[currentVertex][i] && !visited[i]) {
                visited[i] = true;
                queue[++rear] = i;
            }
        }
    }
}
```

```
void DFSWithCost(struct Graph *g, int vertex, int dest, bool visited[], int
pathCost[], int currentCost) {
    visited[vertex] = true;
    pathCost[vertex] = currentCost;

    if (vertex == dest) {
        return;
    }

    for (int i = 0; i < g->vertices; ++i) {
        if (g->matrix[vertex][i] && !visited[i]) {
            DFSWithCost(g, i, dest, visited, pathCost, currentCost + g-
>matrix[vertex][i]);
        }
    }
}


int isReachableDFS(struct Graph *g, int src, int dest) {
    bool visited[MAX_VERTICES] = {false};
    int pathCost[MAX_VERTICES] = {0};
    DFSWithCost(g, src, dest, visited, pathCost, 0);
    if (visited[dest]) {
        return pathCost[dest];
    } else {
        return -1;
    }
}


int main() {
    struct Graph g;
    initGraph(&g, 19);
    addEdge(&g, 0, 1, 135);
    addEdge(&g, 0, 10, 128);
    addEdge(&g, 0, 13, 73);
    addEdge(&g, 1, 0, 135);
    addEdge(&g, 1, 2, 38);
    addEdge(&g, 1, 6, 115);
    addEdge(&g, 1, 7, 80);
    addEdge(&g, 2, 0, 38);
    addEdge(&g, 2, 3, 73);
    addEdge(&g, 3, 2, 73);
    addEdge(&g, 3, 4, 110);
    addEdge(&g, 4, 3, 110);
    addEdge(&g, 4, 5, 104);
    addEdge(&g, 5, 4, 104);
    addEdge(&g, 6, 1, 115);
    addEdge(&g, 7, 1, 80);
    addEdge(&g, 7, 8, 100);
    addEdge(&g, 8, 7, 100);
    addEdge(&g, 8, 9, 68);
    addEdge(&g, 9, 8, 68);
    addEdge(&g, 10, 0, 128);
    addEdge(&g, 10, 11, 175);
    addEdge(&g, 10, 12, 109);
    addEdge(&g, 11, 10, 175);
    addEdge(&g, 12, 10, 109);
    addEdge(&g, 13, 0, 78);
    addEdge(&g, 13, 14, 115);
    addEdge(&g, 13, 15, 181);
    addEdge(&g, 13, 17, 146);
```

```
    addEdge(&g, 14, 13, 115);
    addEdge(&g, 15, 13, 181);
    addEdge(&g, 15, 16, 130);
    addEdge(&g, 16, 15, 130);
    addEdge(&g, 17, 13, 146);
    addEdge(&g, 17, 18, 105);
    addEdge(&g, 18, 17, 105);

    printf("DFS from edge 0: ");
    bool visitedDFS[MAX_VERTICES] = {false};
    DFS(&g, 0, visitedDFS);

    printf("\nBFS from edge 0: ");
    BFS(&g, 0);
    int a, b;
    printf("\n\nenter start edge: ");
    scanf("%d", &a);
    printf("enter destination edge: ");
    scanf("%d", &b);
    printf("%d", isReachableDFS(&g, a, b));


    return 0;
}
```

```
C:\Users\systnager\Documents\university\ASD\lab_07\cmake-build-debug\lab_07.exe
DFS from edge 0: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
BFS from edge 0: 0 1 10 13 2 6 7 11 12 14 15 17 3 8 16 18 4 9 5

enter start edge:5
 enter destination edge:9
 708
Process finished with exit code 0
```

Результат виконання програми

**Висновок:** Освоєно та закріплено прийоми роботи з даними різного типу, організованими у вигляді дерев та їх окремого випадку – бінарних дерев. Здобуто практичні навички роботи з графами.