МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Алгоритми та структури даних. Частина 2

Методичні вказівки
до виконання лабораторних робіт
для здобувачів ступеня бакалавра
за освітньою програмою «Інженерія програмного забезпечення комп'ютерних систем»
спеціальності 121 Інженерія програмного забезпечення

Уклали: д. т. н., проф. кафедри ОТ Анатолій Михайлович Сергієнко
к. т. н., доц. кафедри СП і СКС Олександр Іванович Марченко
Ph. D., ас. кафедри ОТ Анастасія Анатоліївна Молчанова

Електронне мережеве видання

Київ
КПІ ім. Ігоря Сікорського
2024

Зміст

1 Програмування графічного вікна **3**

1.1 Принципи функціонування вікон 3

1.2 Програмування графічного вікна 5

1.2.1 Програмування графічного вікна мовою C для ОС Windows 5

1.2.2 Графічні кнопки у вікні 10

1.2.3 Генерація випадкових чисел 12

1.2.4 Програмування графічного вікна мовою C для ОС Linux 12

1.3 Вирішення можливих помилок компіляції 16

1.3.1 Помилка `undefined reference to `__imp_MoveToEx'` . . . 16

1.3.2 Помилка 0xC000013A при закритті програми 17

2 Лабораторні роботи **19**

Лабораторна робота 1.
Рекурсивні алгоритми 19

Лабораторна робота 2.
Зв'язані динамічні структури даних. Списки 28

Лабораторна робота 3.
Графічне представлення графів 38

Лабораторна робота 4.
Характеристики та зв'язність графа 42

Лабораторна робота 5.
Обхід графа 46

1 Програмування графічного вікна

1.1 Принципи функціонування вікон

В операційній системі (ОС) Windows багатозадачність реалізується як паралельні процеси та потоки. Будь-який застосунок Windows після запуску реалізується як процес. Тут процес представляє собою програмний код, що виконується разом з виділеними системними ресурсами.

Під час запуску процесу система створює основний потік (thread), який виконує код програми з даними в адресному просторі процесу. З основного потоку можуть бути запущені вторинні потоки, які виконуються одночасно з основним потоком. Вікно, що відображається на дисплеї комп'ютера, є результатом виконання певного потоку.

У програми, яка написана для Windows, немає прямого доступу до апаратної частини такого пристрою відображення, як екран. Замість цього вона викликає функції графічної підсистеми, яка називається графічним інтерфейсом пристрою (Graphics Device Interface, GDI).

Функції GDI реалізують графічні операції за допомогою виклику програмних драйверів апаратних пристроїв. Реалізація певної операції є різною для пристроїв різного типу. Але вона прихована від програміста, що спрощує розробку застосунку. Отже, застосунки, які написані з викликами функцій GDI, будуть правильно виконуватись з будь-яким типом дисплея, для якого встановлений драйвер Windows.

Інтерфейси функцій GDI перелічені у заголовкових файлах Windows, головним серед яких є windows.h. В ньому є посилання на інші заголовкові файли.

Функції GDI викликаються з DLL-бібліотеки. Вони реалізуються так само, як функції, що програмуються користувачем, але відміна полягає в тому, що зв'язування коду з функціями відкладається під час компіляції та реалізується під час виконання програми (динамічне зв'язування). Більша частина цих бібліотек розміщена в підкаталозі System32.

Вікно — це базовий об'єкт, з яким мають справу як ОС, так і програміст, а також користувач застосунку. Вікно одержує інформацію від клавіатури чи миші користувача та виводить графічну інформацію на екран.

З вікном взаємодіють різноманітні графічні об'єкти, які застосовуються для малювання, такі як пера, пензлі, шрифти, палітри та інші. Незалежно від свого типу, кожен об'єкт у Windows ідентифікується своїм дескриптором

(handle). Дескриптор — це посилання на об'єкт, яке, крім його адреси, містить додаткову інформацію, яка необхідна для керування ним. Усі взаємодії програмного коду з об'єктом виконуються лише через його дескриптор.

Програма користувача взаємодіє з ОС через повідомлення. Повідомлення повідомляє ОС, що сталась певна подія, на яку має зреагувати ОС особливим чином. Такою подією є, наприклад, сигнал від миші чи клавіатури.

Як правило, повідомлення оформлене як структура даних, яка вміщує дескриптор вікна, якому воно адресоване, код повідомлення, додаткову інформацію, як, наприклад, координати пікселів вікна.

Повідомлення від кількох джерел, що стосуються одного вікна, направляються у системну чергу повідомлень ОС. Windows періодично опитує цю чергу і направляє чергове повідомлення відповідному адресату, який заданий у дескрипторі (рис. 1).

У кожного віконного застосунку завжди є головна функція WinMain(), яка виконує роль функції main(). В ній програмуються функції для ініціалізації та створення вікна, а також цикл обробки повідомлень і, насамкінець, код для завершення роботи застосунку.

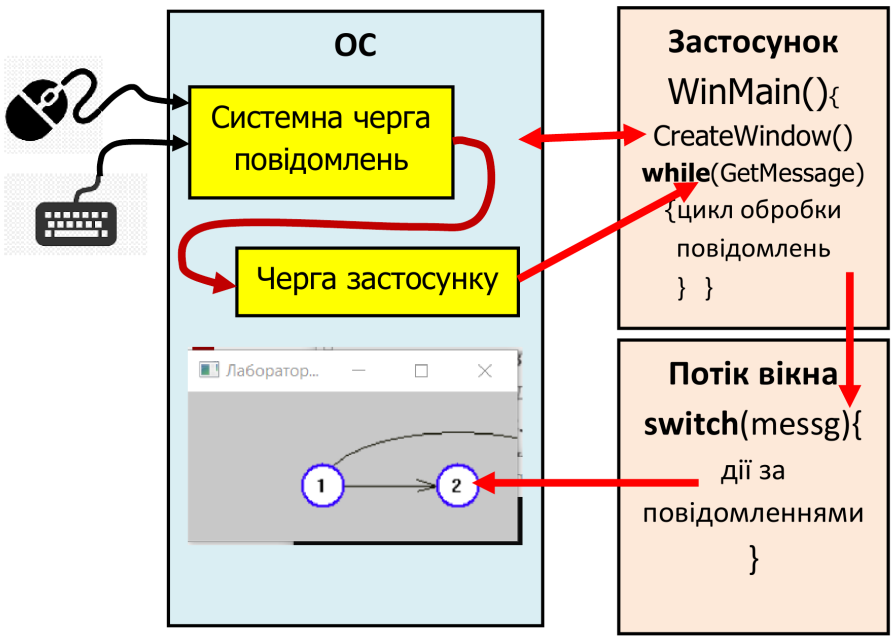


Рис. 1. Взаємодія ОС і застосунку під час графічних операцій

Як правило, в програмі є цикл одержання повідомлення з черги за допомогою функції GetMessage(). Повідомлення, які одержані цією функцією, передаються до циклу обробки повідомлень, який реалізується у окремому потоці, який викликається як функція.

Якщо чергове повідомлення має код WM_DESTROY, то за ним з потоку в ОС передається синхронізуюче повідомлення про завершення роботи з вікном функцією PostQuitMessage(0), виконується вихід з циклу і завершення програми.

1.2 Програмування графічного вікна

1.2.1 Програмування графічного вікна мовою С для ОС Windows

Розлянемо простий приклад програмування креслення графа у графічному вікні. Під час опису прикладу наводяться рядки програми з їхнім описом у тому самому порядку, в якому вони стоять у тексті програми. Про подробиці реалізації структур та опції їх параметрів можна довідатися у керівних матеріалах, довідниках та підручниках з інтерфейсу Win32 API.

Спочатку підключається бібліотека для роботи з застосунками Windows та бібліотека математичних функцій.

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

Створюється прототип функції потоку вікна, яка буде визначена в кінці програми і буде неявно запускатися з головної функції.

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

Оголошується рядок — ім'я програми.

```
char ProgName[]="Лабораторна робота 3";
```

Викликається головна функція — повертає структуру типу `WINAPI`.

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
   LPSTR lpszCmdLine, int nCmdShow)
{
```

Спочатку створюється екземпляр `w` структури `WNDCLASS`.

```
    WNDCLASS w;
```

Ця структура містить наступні атрибути вікна.

```
    w.lpszClassName=ProgName; // ім'я програми
    w.hInstance=hInstance;    // ідентифікатор застосунку
    w.lpfnWndProc=WndProc;  //вказівник на функцію вікна
    w.hCursor=LoadCursor(NULL, IDC_ARROW); // завантажений курсор
    w.hIcon=0;   //піктограми не буде
    w.lpszMenuName=0;   // меню не буде
    w.hbrBackground = WHITE_BRUSH;  //колір фона вікна
    w.style=CS_HREDRAW|CS_VREDRAW;   //стиль: можна перемальовувати
    w.cbClsExtra=0;     //к-ть додаткових байтів для цього класу
    w.cbWndExtra=0;     //
```

Якщо структуру вікна ОС не зареєструвала, то програма завершується.

```
18      if(!RegisterClass(&w))
19          return 0;
```

Створюється екземпляр вікна в пам'яті та змінна типу повідомлення.

```
20      HWND hWnd;
21      MSG lpMsg;
```

Вікно заповнюється параметрами та даними.

```
22      hWnd=CreateWindow(ProgName, // Ім'я програми
23      "Лабораторна робота 3. Виконав А.М.Сергієнко", //Заголовок
24      WS_OVERLAPPEDWINDOW, //Стиль вікна - комплексний
25      100,        //положення верхнього кута вікна на екрані по x
26      100,        // положення верхнього кута вікна на екрані по y
27      460,                //ширина вікна
28      240,                //висота вікна
29      (HWND)NULL,             // ідентифікатор породжуючого вікна
30      (HMENU)NULL,            //ідентифікатор меню (немає)
31      (HINSTANCE)hInstance, //ідентифікатор екземпляра вікна
32      (HINSTANCE)NULL);     // додаткові параметри відсутні
```

Вікно виводиться на екран.

```
33      ShowWindow(hWnd, nCmdShow);
```

Організується цикл читання повідомлень з черги.

```
34      int b;
35
36      while((b = GetMessage(&lpMsg, hWnd, 0, 0))!= 0) {
37                                      //повідомлення з черги
38          if(b == -1) {
39              return lpMsg.wParam;
40          }
41          TranslateMessage(&lpMsg);   //перетворення повідомлення
42                                       // у рядок
43          DispatchMessage(&lpMsg); //Передача повідомлення
44                                  //до функції вікна
45      }
46      return(lpMsg.wParam); // кінець основної функції
47  }
```

Перевірка `b == -1` потрібна для випадку, якщо функція `GetMessage` звертається до недійсного вікна (наприклад, такого, що вже було зруйноване). Оскільки -1, як ненульове значення, сприймається як true, потрібно запобігти нескінченному виконанню циклу і завершити виконання основної функції.

Опис функції вікна — потоку, який власне креслить граф, — є наступний.

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT messg,
                         WPARAM wParam, LPARAM lParam)
{
```

Створюється контекст вікна та екземпляр структури графічного виводу.

```
    HDC hdc;              // контекст
    PAINTSTRUCT ps;       // екземпляр структури
```

Описується допоміжна функція, яка виводить стрілочку ребра графа.

```
    void arrow(float fi, int px,int py){
        fi = 3.1416*(180.0 - fi)/180.0;
        int lx,ly,rx,ry;
        lx = px+15*cos(fi+0.3);
        rx = px+15*cos(fi-0.3);
        ly = py+15*sin(fi+0.3);
        ry = py+15*sin(fi-0.3);
        MoveToEx(hdc, lx, ly, NULL);
        LineTo(hdc, px, py);
        LineTo(hdc, rx, ry);
        return 0;
    }
```

Ця функція вимальовує два вектори так, як показано на рис. 2.

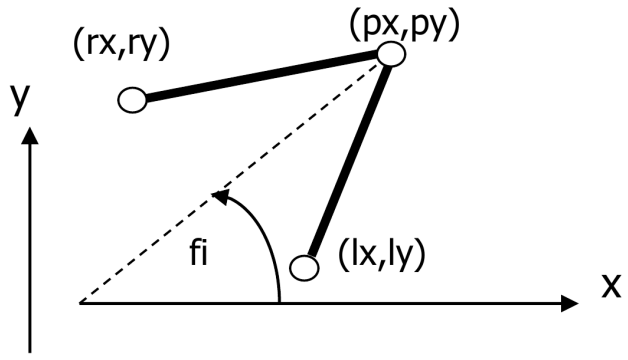


Рис. 2. Графічне представлення кінця стрілки за функцією arrow()

Кут напрямленості стрілки `fi` задається в градусах і відкладається у напрямку, показаному на рис. 2. Параметри `px`, `py` задають координати кінця стрілки.

Тут функція `MoveToEx(hdc, lx, ly, NULL);` переставляє перо у точку вікна з дескриптором `hdc` з координатами `lx`, `ly` і повертає попередні координати пера, якщо останній параметр не `NULL`.

Функція `LineTo(hdc, px, py);` креслить у вікні з дескриптором `hdc` пряму лінію пером від точки з координатами пера до точки зі вказаними координатами `px`, `py`.

Цикл обробки повідомлень відкривається так:

```
17    switch(messg){
18        case WM_PAINT :
```

У ньому за першою альтернативою — повідомленням `WM_PAINT` — програмують усі дії з креслення задуманого графічного образу у вікні з дескриптором `hdc`. Спочатку одержується дескриптор вікна.

```
19            hdc=BeginPaint(hWnd, &ps);
```

Потім задаються мітки та координати вершин графа, який слід накреслити, та декларуються допоміжні змінні. Для простоти й наочності прикладу тут координати не обчислюються, а задаються явно.

```
19            hdc=BeginPaint(hWnd, &ps);
20            char *nn[3] = {"1", "2", "3"};
21            int nx[3] = {100,200,300};
22            int ny[3] = {100,100,100};
23            int dx = 16, dy = 16, dtx = 5;
24            int i;
```

Тут `dx`, `dy`, `dtx` — це параметри радіуса кола вершини та зміщення для друку в колі мітки вершини у кількості пікселів. Далі задаються пера з дескрипторами `BPen`, `KPen`, які креслять лінію типу `PS_SOLID`, тобто, безперервну пряму.

```
25            HPEN BPen = CreatePen(PS_SOLID, 2, RGB(50, 0, 255));
26            HPEN KPen = CreatePen(PS_SOLID, 1, RGB(20, 20, 5));
```

Тут цифри 2, 1 означають товщину лінії у пікселах. Параметри `RGB(50, 0, 255)`, `RGB(20, 20, 5)` задають синій та чорний колір пера, відповідно.

Задається активне перо `KPen`.

```
27            SelectObject(hdc, KPen);
```

Креслиться перша напрямлена дуга графа, використовуючи координати 0-ї та 1-ї вершин.

```
28            MoveToEx(hdc, nx[0], ny[0], NULL);
29            LineTo(hdc, nx[1], ny[1]);
30            arrow(0,nx[1]-dx,ny[1]);
```

Креслиться друга напрямлена дуга графа, використовуючи координати 0-ї та 2-ї вершин.

```
            Arc(hdc, nx[0], ny[0]-40, nx[2], ny[2]+40,
                        nx[2], ny[2], nx[0], ny[0]);
            arrow(-45.0,nx[2]-dx*0.5,ny[2]-dy*0.8);
```

Функція `Arc` креслить дугу у вікні `hdc`. Чотири наступні її параметри задають координати верхнього лівого кута та нижнього правого кута прямокутника, в який вписаний еліпс, частиною якого є дуга. Решта параметрів задають координати початкової та кінцевої точок дуги.

Далі у циклі кресляться три кола вершин пером `BPen`.

```
            SelectObject(hdc, BPen);
            for(i=0; i<=2; i++){
                Ellipse(hdc, nx[i]-dx,ny[i]-dy,nx[i]+dx,ny[i]+dy);
                TextOut(hdc, nx[i]-dtx,ny[i]-dy/2,  nn[i],1);
            }
```

Тут функція `Ellipse` креслить коло (еліпс). Її параметри задають координати верхнього лівого кута та нижнього правого кута прямокутника, в який вписаний еліпс. Функція `TextOut` виводить текст `nn[i]` у рядок із заданими початковими координатами. Її останній параметр (1) дорівнює кількості символів, які виводяться.

Нарешті, процес малювання закінчується.

```
            EndPaint(hWnd, &ps);
            break;
```

Іншою альтернативою є видалення вікна, про що ОС повідомляє сигналом `WM_DESTROY`. Видалення виконується ОС, коли користувач активує кнопку закриття вікна.

```
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
```

При цьому програма повертає повідомлення 0 про нормальне завершення.

У решті випадків у черзі повідомлень залишаються нерозпізнані повідомлення, які просто видаляються з черги.

```
        default:
            return(DefWindowProc(hWnd, messg, wParam, lParam));
    }
}
```

У результаті виконання цієї програми одержується зображення, як на рис. 3.

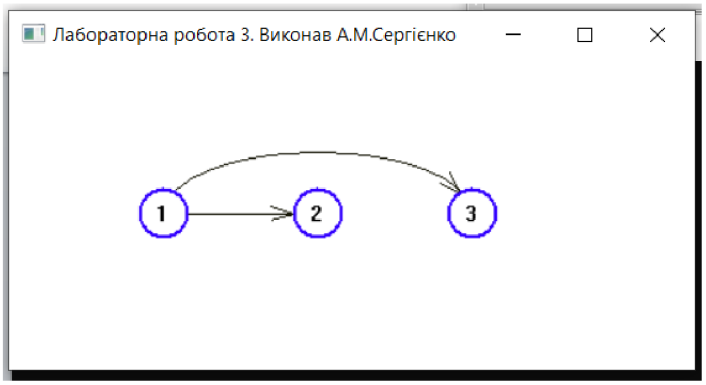


Рис. 3. Зображення графа побудоване функціями з пакету Windows

1.2.2 Графічні кнопки у вікні

Якщо є потреба додати у вікно графічні кнопки, потрібно у головну функцію вікна додати створення кнопок як елементів вікна, а в цикл обробки повідомлень необхідно додати обробку натискань на кнопки.

У цьому прикладі до вікна, розглянутого вище, додамо дві кнопки.

```
int WINAPI WinMain (....)
{
    ...

    HWND button1 = CreateWindow("BUTTON",   // перша кнопка
        "Перша кнопка",                    // текст на кнопці
        WS_VISIBLE | WS_CHILD | WS_BORDER,
        150, 100,       // координати кнопки (150, 100)
        110, 30,            // ширина 110, висота 30
        hWnd, (HMENU)1, NULL, NULL);     // (HMENU)1 – це її ідентифікатор

    HWND button2 = CreateWindow("BUTTON",   // друга кнопка
        "Друга кнопка",
        WS_VISIBLE | WS_CHILD | WS_BORDER,
        150, 135,       // має координати (150, 135)
        110, 30,
        hWnd, (HMENU)2, NULL, NULL);        // і номер 2

    ShowWindow(hWnd, nCmdShow);
    ...
}
```

Ці кнопки будуть розташовані нижче за граф, одна під одною. Розташування кнопки визначається координатами її верхнього кута ((150, 100) для першої кнопки і (150, 135) для другої). Розміри кнопок будуть однакові — 110x30. На кнопках будуть написи «Перша кнопка» та «Друга кнопка». hWnd — це ідентифікатор породжуючого вікна. Також кнопки мають свої

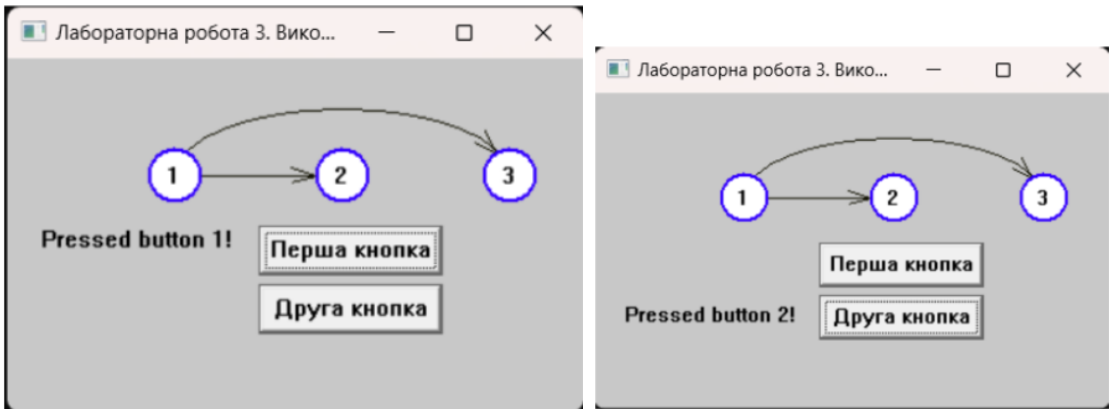
ідентифікатори в меню: 1 і 2 відповідно. За цими номерами їх можна буде розрізняти при обробці повідомлень про натискання на кнопку.

А в цикл обробки повідомлень треба додати обробку натискань.

```
switch(messg)
{
    ...
    case WM_COMMAND:
    {
        RECT rect;                              // очищуємо екран, інакше
        GetClientRect(hWnd, &rect);             // при перемиканні буде
        InvalidateRect(hWnd, &rect, TRUE);  // одразу обидва написи
        hdc=GetDC(hWnd);
        UpdateWindow(hWnd);
        SetBkMode( hdc, TRANSPARENT );  // тло прозоре, щоб тло напису
                                    // не відрізнялося від решти

        switch (LOWORD(wParam))     // перевіряємо ідентифікатор
        {                           // ( (HMENU) кнопки)
            case 1:
                TextOut(hdc, 20, 100, "Pressed button 1!", 17); // якась дія
                break;
            case 2:
                TextOut(hdc, 20, 140, "Pressed button 2!", 17); // інша дія
                break;
            default:
                printf("other\n");
                break;
        }
    }
    ...
}
```

У цьому прикладі за натисканням кнопки на екран виводиться відповідний напис (рис. 4). У загальному випадку, для обслуговування натискання кнопки краще написати функцію, яка викликається у відповідному випадку, й інкапсулювати в цій функції всі необхідні дії.



(а) Перша кнопка (б) Друга кнопка

Рис. 4. Виконання дій при натисканні кнопок

1.2.3 Генерація випадкових чисел

У лабораторних роботах задіяні випадкові числа. Для генерації таких чисел використовуються функції `rand()` та `srand()`. Функція `rand()` генерує псевдовипадкові цілі числа в діапазоні від 0 до `RAND_MAX`, де `RAND_MAX` — це константа, яка задана в бібліотеці `<cstdlib>`.

Для одержання початкового випадкового значення функції `rand()` слід перед нею викликати функцію `srand(seed)`, де `seed` — натуральне число, яке необхідне для запуску генератора випадкових чисел у функції `rand()`. Генератор випадкових чисел насправді повертає псевдовипадкові числа, і `seed` є параметром для обчислення цих чисел. Зазвичай `seed` це випадкове число, наприклад, поточне значення системного лічильника часу. Якщо ж `seed` задати як фіксоване значення, то псевдовипадкові числа при кожному запуску програми будуть одні й ті ж. Таким чином, у ЛР № 3, 4, 5, 6 завдяки встановленню `seed`, рівному номеру варіанту, при кожному запуску програми одержується одна й та ж матриця напрямленого графа, унікальна для конкретного варіанту.

Для одержання випадкового числа у діапазоні (0.0–1.0) результат функції `rand()` слід поділити на системну константу `RAND_MAX`.

1.2.4 Програмування графічного вікна мовою C для ОС Linux

Наведена нижче програма є прикладом того, як запрограмувати для ОС Linux графічне вікно, у якому малюється граф, таке ж саме, як розглядалося в розділі 1.2.1 для ОС Windows.

Текст програми мовою С	Пояснення

`#include <X11/Xlib.h>`
`#include <X11/Xutil.h>`
`#include <X11/Xos.h>`
	Підключення X-бібліотек

`#include <stdio.h>`
`#include <stdlib.h>`
`#include <math.h>`
	Підключення стандартних бібліотек та Math

`/* here are our X variables */`
`Display *dis;`
`int screen;`
`Window win;`
`GC gc;`
	dis — підключення до X-сервера
	screen — екран, який використовуємо
	win — наше вікно
	gc — графічний контекст (він визначає атрибути графічних операцій, такі як стиль лінії, тло, стиль заливки, шрифт тощо)

`/* Program name */` `char ProgName[] = "Laboratory work 3";`	Назва програми, яку потім підставимо при створенні вікна

```
/* here are our X routines declared! */
void init_x();
void close_x();
void redraw();
```

Оголошення функцій для роботи з вікном

```
/* graph drawing routines declared */
void arrow(float fi,int px,int py);
void draw_graph();
```

Оголошення функцій для малювання графа

```
main () {
    XEvent event;
    KeySym key;
    char text[255];
```

event — змінна типу подія, в яку будемо зберігати повідомлення про події
key — змінна, потрібна для виклику функції трансляції натискання клавіш у текст
text — буфер, у який буде зберігатися текст натиснутих клавіш

```
    init_x();
```

Створили вікно

```
    /* look for events forever... */
    while(1) {
        XNextEvent(dis, &event);
```

Нескінченний цикл обробки подій.
Функція **XNextEvent** читає повідомлення з голови черги і записує в змінну **event**.
Зверніть увагу, що повідмолення будуть лише про ті події, які було задано маскою.

```
        if (event.type==Expose &&
                        event.xexpose.count==0) {
            redraw();
        }
```

Тип події **Expose** — це коли частина вікна, яку затулило інше вікно, стала знову видимою. У такому випадку треба його перемалювати, викликаємо **redraw**()

```
        if (event.type==KeyPress &&
            XLookupString(&event.xkey,
                    text,255,&key,0)==1) {
```

Якщо сталася подія — натискання клавіші (**KeyPress**):
XLookupString транслює подію в буфер-рядок **text** та значення **key** типу **KeySym** (останнє нам в даному випадку не потрібне)

```
            if (text[0]=='q') {
                close_x();
            }
            else {
                draw_graph();
            }
        }
    }
}
```

Якщо було натиснуто клавішу **q** — закрити вікно,
якщо іншу клавішу — виводимо зображення графа
(обидві функції описані нижче)

```
/* Create window */
void init_x() {
    unsigned long black,white;
```

Функція для створення вікна

```
    dis=XOpenDisplay((char *)0);
    screen=DefaultScreen(dis);
```

Підключення до X-сервера:
XOpenDisplay повертає вказівник на структуру типу Display, яка слугує як підключення до X-сервера
DefaultScreen вибирає поточний екран

```
    black=BlackPixel(dis, screen),
    white=WhitePixel(dis, screen);
```

одержали чорний колір
одержали білий колір

```
    win=XCreateSimpleWindow(dis,
                    DefaultRootWindow(dis),
                    0,
                    0,
                    460,
                    240,
                    1,
                    black,
                    white);
```

Створюємо просте вікно
• **dis** — підключення до X-сервера
• **DefaultRootWindow(dis)** — як батьківське вікно призначили кореневе вікно для екрану за замовчуванням
• **0, 0** — координати x, y верхнього лівого кута відносно батьківського вікна
• ширина вікна **460**,
• висота вікна **240**,
• контур вікна завширшки **1** піксель
• колір контуру вікна: чорний
• колір тла: білий

```
    XSetStandardProperties(dis,
                    win,
                    ProgName,
                    ProgName,
                    None,
                    NULL,
                    0,
                    NULL);
```

Задаємо властивості вікна
• **dis** — підключення до X-сервера
• **win** — наше вікно
• назва вікна та • назва мінімізованого вікна — наша назва програми, задана вище
• без іконки
• без списку аргументів
• к-ть аргументів 0
• без size hints для вікна

```
    XSelectInput(dis, win,
            ExposureMask|KeyPressMask);
```

Задаємо, про які типи подій X-сервер буде повідомляти (інакше він про них не повідомлятиме).
ExposureMask|KeyPressMask — маска, що вказує, що розглядатися будуть події Expose та KeyPress

```
    gc=XCreateGC(dis, win, 0,0);
```

Створюємо графічний контекст

```
    XMapRaised(dis, win);
};
```

XMapRaised робить вікно видимим і показує його поверх усіх інших вікон

```
/* Close window */
void close_x() {
    XFreeGC(dis, gc);
    XDestroyWindow(dis,win);
    XCloseDisplay(dis);
    exit(0);
};
```

Коректно закриваємо вікно, віддаючи системі системні ресурси.
XFreeGC — видаляє графічний контекст і звільняє відповідну пам'ять
XDestroyWindow — видаляє вікно
XCloseDisplay — закриває з'єднання з X-сервером
exit(0) — завершити програму

```
/* Redraw window */
void redraw() {
    XClearWindow(dis, win);
};
```

Перемалювати вікно

```
/* Draw an arrowhead */
void arrow(float fi, int px,int py){
            fi = 3.1416*(180.0 - fi)/180.0;
            int lx,ly,rx,ry; //px,py,
           // px=150; py=60;
            lx = px+15*cos(fi+0.3);
            rx = px+15*cos(fi-0.3);
            ly = py+15*sin(fi+0.3);
            ry = py+15*sin(fi-0.3);
            XDrawLine(dis,win,gc, lx, ly, px, py)
    ;
            XDrawLine(dis,win,gc, px, py, rx, ry)
    ;
         //  return 0;
}
```

Функція для малювання вістря стрілки — як було в прикладі для Windows

```
/* Draw graph */
void draw_graph(){
    char *nn[3] = {"1", "2", "3"};
    int nx[3] = {100,200,300};
    int ny[3] = {100,100,100};
    int dx = 16, dy = 16, dtx = 5;
    int i;
```

Функція для малювання графа — як у прикладі для Windows.
У лабораторних роботах слід обчислювати всі координати автоматично в циклі, а тут для простоти вони задані явно:
nn[3] — мітки вершин
nx[3], ny[3] — координати вершин x та y
dx, dy, dtx — радіус кола вершини та зміщення для друку в колі мітки вершини в пікселах

```
    unsigned long bpen = 0x3200FF;
    unsigned long kpen = 0x141405;
```

Синій та чорний колір (як у прикладі для Windows), задані шістнадцятковим числом в RGB: RRGGBB

```
    XSetForeground(dis,gc,kpen);
```

Вибираємо чорний колір

```
    /* Draw straight arrow */
    XDrawLine(dis,win,gc,nx[0],ny[0],nx[1],ny[1])
    ;
    arrow(0,nx[1]-dx,ny[1]);
```

Креслимо горизонтальну лінію з точки (nx[0], ny[0]) в точку (nx[1], ny[1]);
на кінці креслимо вістря стрілки зі зміщенням на dx

```
    /* Draw arc */
    XDrawArc(dis,win,gc,
            nx[0], ny[0]-40,
            nx[2]-nx[0], 80,
            0, 180*64);
```

Креслимо дугу:
• **nx[0], ny[0]-40** — x, y верхнього лівого кута прямокутника, в який вписується дуга еліпса
• **nx[2]-nx[0], 80** — ширина та висота (головні вісі еліпса)
• **0, 180*64** — початок та кінець дуги (задані кутом, одиниці вимірювання 1/64 градуса; 0 — праворуч від центра). 11520 = 180*64 — тобто, кінець дуги 180 градусів

```
    arrow(-45.0,nx[2]-dx*0.5,ny[2]-dy*0.8);
```

На кінці креслимо вістря стрілки

```
    /* Draw circles with numbers */
    XSetForeground(dis,gc,white);
```

Вибираємо білий колір

```
    for(i=0; i<=2; i++){
        XFillArc(dis,win, gc,
                nx[i]-dx, ny[i]-dy,
                2*dx,2*dy,
                0, 360*64);
    }
```

Заливаємо кружечки білим кольором:
• **nx[i]-dx, ny[i]-dy** — x, y верхнього лівого кута прямокутника, в який вписується дуга еліпса
• **2*dx,2*dy** — ширина та висота (головні вісі дуги)
• **0, 360*64** — кути початку та кінця дуги

```
    XSetForeground(dis,gc,bpen);

    XSetLineAttributes(dis, gc,
                2,
                LineSolid,
                CapButt, JoinMiter);

    for(i=0; i<=2; i++){
        XDrawArc(dis,win,gc, nx[i]-dx,ny[i]-dy,
                2*dx,2*dy, 0, 360*64);
        XDrawString(dis,win,gc,
                nx[i],ny[i],
                nn[i],
                1);
    }
}
```

Вибираємо синій колір

Задаємо більшу товщину лінії:
- товщина лінії 2 пікселя
- стиль лінії: суцільний (LineSolid)
- стиль кінця лінії та стиль з'єднання ліній

Малюємо сині кола і номери вершин всередині:
креслимо коло;

за координатами (nx[i],ny[i]) друкуємо на екрані номер вершини nn[i],
1 — довжина тексту

1.3 Вирішення можливих помилок компіляції

1.3.1 Помилка `undefined reference to `__imp_MoveToEx'`

У такому випадку в налаштуваннях лінкера треба підключити бібліотеку `libgdi32`.

При компіляції через ***консоль*** ця бібліотека підключається за допомогою опції -lgdi32:
`gcc main.c -o lab.exe -lm -lgdi32`

У ***CodeBlocks*** це робиться так. Треба відкрити в меню Project->Build options->Linker settings, натиснути Add та вибрати файл libgdi32.a Сам файл libgdi32.a має бути десь у папці з MinGW (наприклад, C:\MinGW\lib\libgdi32.a). Якщо цього виявилося недостатньо, у сусідній панелі «Other linker options:» вписати `-lgdi32` (рис. 5 та рис. 6).

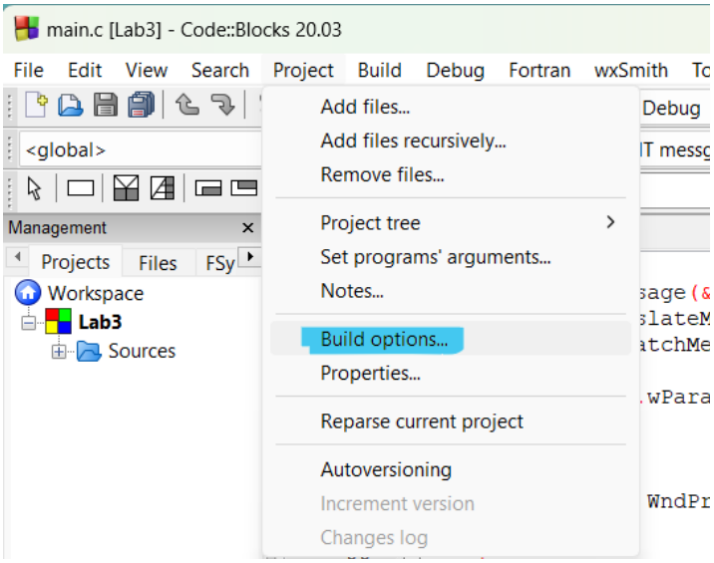


Рис. 5. Розташування підменю «Build options» в середовищі CodeBlocks

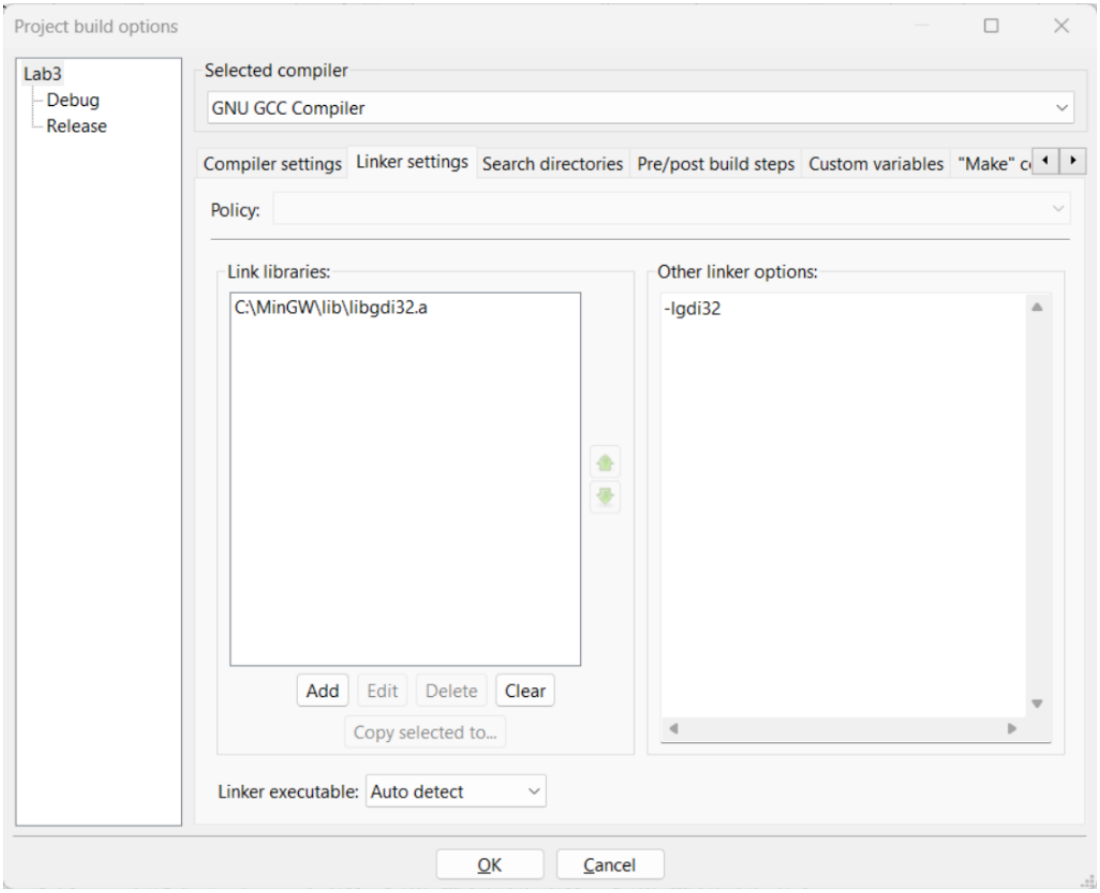


Рис. 6. Вказівка лінкеру про під'єднання бібліотеки libgdi32

1.3.2 Помилка 0xC000013A при закритті програми

Ця помилка виникає, якщо програма була завершена не коректно, а шляхом припинення виконання процесу — коли закрито вікно консолі, або натиснуто Ctrl+C. Однією з причин може бути те, що цикл одержання повідомлень має вигляд: `while(GetMessage(&lpMsg, hWnd, 0, 0)) ...`

Згідно з документацією Microsoft[1], функція `GetMessage` може повертати -1, у випадку, якщо звертається до недійсного вікна (наприклад, такого, що вже було зруйноване). У такому разі цикл одержання повідомлень `while(GetMessage(&lpMsg, hWnd, 0, 0)) ...` продовжить нескінченно виконуватися, оскільки -1 це ненульове значення (а отже, спрацьовує як true).

Рекомендується змінити цикл обробки повідомлень таким чином:

```
    int b;
    while((b = GetMessage(&lpMsg, hWnd, 0, 0))!= 0) {
            if(b == -1) {
            return lpMsg.wParam;
        }
        else {
            TranslateMessage(&lpMsg);
            DispatchMessage(&lpMsg);
        }
    }
```

[1] GetMessage function (winuser.h) – Win32 apps | Microsoft Learn. URL: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getmessage#return-value

2 Лабораторні роботи

Лабораторна робота 1. Рекурсивні алгоритми

Мета лабораторної роботи

Метою лабораторної роботи №1 «Рекурсивні алгоритми» є засвоєння теоретичного матеріалу та набуття практичного досвіду створення рекурсивних алгоритмів та написання відповідних їм програм.

Постановка задачі

Дане натуральне число n. Знайти суму перших n членів ряду чисел, заданого рекурентною формулою. Розв'язати задачу ***трьома способами***:

1) у програмі використати рекурсивну функцію, яка виконує обчислення і членів ряду, і суми на рекурсивному спуску;

2) у програмі використати рекурсивну функцію, яка виконує обчислення і членів ряду, і суми на рекурсивному поверненні;

3) у програмі використати рекурсивну функцію, яка виконує обчислення членів ряду на рекурсивному спуску, а обчислення суми на рекурсивному поверненні.

При проєктуванні програм ***слід врахувати наступне***:

1) програми повинні працювати коректно для довільного цілого додатного n включно з $n = 1$;

2) видимість змінних має обмежуватися тими ділянками, де вони потрібні;

3) функції повинні мати властивість модульності;

4) у кожному з трьох способів рекурсивна функція має бути одна (за потреби, можна також використати додаткову функцію-обгортку (wrapper function));

5) у другому способі можна використати запис (struct) з двома полями (але в інших способах у цьому немає потреби і це вважатиметься надлишковим);

6) програми мають бути написані мовою програмування C.

Результати виконання роботи

Як результат виконання лабораторної роботи надіслати:

1) звіт до лабораторної роботи у форматі .PDF;

2) файл (або файли) з текстом програм (із розширенням .C).

Зміст звіту

1. Титульна сторінка.

2. Загальна постановка завдання.

3. Завдання для конкретного варіанту.

4. Текст усіх програм.

5. Скриншоти результатів тестування програм.

6. Графік залежності похибки обчислення заданої функції від значення x при фіксованому значенні n.

7. Висновки.

Тестування програм

1. З метою тестування потрібно написати циклічний варіант рішення задачі, а також перевірити правильність обчислень елементів ряду та їх суми за допомогою калькулятора.

2. Як результат, роздрукувати дані тестування усіма трьома рекурсивними функціями та циклічною програмою, а також навести скриншот обчислень на калькуляторі. Для тестування прийняти $n = 5$, а значення x вибрати самостійно (у межах області визначення функції). Результати обчислень усіма способами повинні співпадати.

Графік похибки обчислення функції

Алгоритм кожного варіанту обчислює певну задану функцію з деякою похибкою. А саме, у другому рядку завдання за варіантом вказано, що сума членів ряду дорівнює значенню функції. У звіті має бути наведено графік залежності цієї похибки від x (тобто, по горизонтальній вісі відкладаються значення x, а по вертикальній — значення похибки), при сталому значенні n. Вісі мають бути підписані.

Деякі з функцій, що розглядаються, є гіперболічними та оберненими гіперболічними функціями, як-от: sh (гіперболічний синус), ch (гіперболічний косинус), arsh (гіперболічний ареасинус), arth (гіперболічний ареатангенс).

Похибка обчислюється як різниця за модулем між апроксимованим (приблизним) значенням, обчисленим програмою ($\sum F_i$), та фактичним значенням функції. Наприклад, якщо $\sum F_i = \sqrt{x}$, то $\sum F_i$ це сума елементів ряду, обчислених програмою, а $\sqrt{x}$ — фактичне значення функції (еталонне), яке треба обчислити за допомогою калькулятора, MS Excel, Google Таблиць тощо. Тоді значення похибки для певного значення x буде дорівнювати $|\sum F_i - \sqrt{x}|$.

Шкала графіка повинна мати фіксовану ціну поділки — щоби графік мав правильну форму. Для цього слід або обирати значення x із рівномірним інтервалом, або встановити крок шкали в налаштуваннях діаграми. Наприклад, у Google Таблицях (sheets.google.com) це виконується так. Правим кліком по діаграмі відкривається контекстне меню, там треба вибрати: Стиль діаграми → Оформлення → Сітки та позначки. У цьому меню слід вибрати потрібну вісь і задати величину кроку (рис. 7).

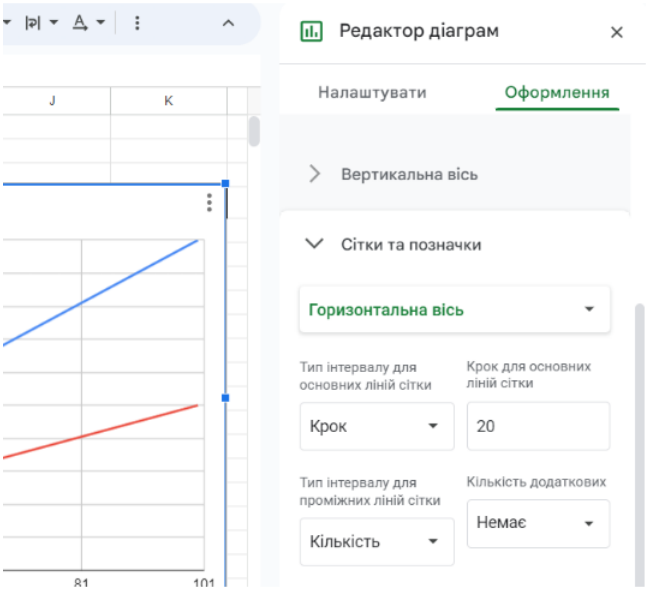


Рис. 7. Налаштування кроку шкали

Контрольні питання

1. Визначення рекурсивного об'єкта.
2. Визначення глибини та поточного рівня рекурсії.
3. Форма виконання рекурсивних дій на рекурсивному спуску.
4. Форма виконання рекурсивних дій на рекурсивному поверненні.
5. Форма виконання рекурсивних дій на як рекурсивному спуску, так і на рекурсивному поверненні.

Варіанти індивідуальних завдань

Варіант № 1

$$F_1 = 1; \quad F_2 = -x/2; \quad F_i = F_{i-1} \cdot x \cdot (2i-3)/(2i), \quad i > 2;$$
$$\sum_{i=1}^{n} F_i = \sqrt{1-x}, \quad |x| < 1.$$

Варіант № 2

$$F_1 = 1; \quad F_2 = -x/3; \quad F_i = -F_{i-1} \cdot x \cdot (3i-7)/(3i-3), \quad i > 2;$$
$$\sum_{i=1}^{n} F_i = \sqrt[3]{1-x}, \quad |x| < 1.$$

Варіант № 3

$$F_1 = 1; \quad F_i = -F_{i-1} \cdot x \cdot (2i-3)/(2i-2), \quad i > 1;$$
$$\sum_{i=1}^{n} F_i = 1/\sqrt{1+x}, \quad |x| < 1.$$

Варіант № 4

$$F_1 = 1; \quad F_i = -F_{i-1} \cdot x \cdot (3i-5)/(3i-3), \quad i > 1;$$
$$\sum_{i=1}^{n} F_i = 1/\sqrt[3]{1+x}, \quad |x| < 1.$$

Варіант № 5

$$F_1 = 1; \quad F_i = -F_{i-1} \cdot (2x/3-1), \quad i > 1;$$
$$\sum_{i=1}^{n} F_i = 1.5/x, \quad 1 < x < 2.$$

Варіант № 6

$$F_1 = 1; \quad F_{i+1} = F_i \cdot x \cdot (\ln 2)/(i-1), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = 2^x.$$

Варіант № 7

$$F_1 = (x-1)/(x+1); \quad F_{i+1} = F_i \cdot (2i-1)(x-1)^2/((2i+1)\cdot(x+1)^2), \quad i > 1;$$
$$\sum_{i=1}^{n} F_i = 0.5 \ln x, \quad x > 0.$$

Варіант № 8

$$F_1 = x - 1; \quad F_{i+1} = -F_i \cdot (x-1) \cdot i/(i+1), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \ln x, \quad 0 < x < 2;$$

Варіант № 9

$$F_1 = (x-1)/x; \quad F_{i+1} = F_i \cdot i \cdot (x-1)/(i \cdot x + x), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \ln x, \quad x > 0.5.$$

Варіант № 10

$$F_1 = x; \quad F_2 = x^2/2; \quad F_i = -F_{i-1} \cdot x \cdot (i-1) \cdot (i-2)/(i^2 - i), \quad i > 2;$$
$$(\sum_{i=1}^{n} F_i)/(1+x) = \ln(1+x), \quad -1 < x < 2.$$

Варіант № 11

$$F_1 = x; \quad F_{i+1} = -F_i \cdot x^2/(4i^2 + 2i), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \sin x.$$

Варіант № 12

$$F_1 = 1; \quad F_{i+1} = -F_i \cdot x^2/(4i^2 - 2i), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \cos x.$$

Варіант № 13

$$F_1 = x; \quad F_{i+1} = F_i \cdot x^2(2i-1)^2/(4i^2 + 2i), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \arcsin x, \quad -1 < x < 1.$$

Варіант № 14

$$F_0 = x; \quad F_i = F_{i-1} \cdot (2i-1)^2 \cdot x^2/(2i(2i+1)), \quad i > 0;$$
$$\pi/2 - \sum_{i=0}^{n} F_i = \arccos x.$$

Варіант № 15

$$F_1 = x; \quad F_{i+1} = -F_i \cdot x^2(2i-1)/(2i+1), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \operatorname{arctg} x, \quad |x| < 1.$$

Варіант № 16

$$F_1 = x; \quad F_{i+1} = F_i \cdot (2i-1)^2 \cdot x^2/(4i^2+2i), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \arcsin x, \quad |x| < 1.$$

Варіант № 17

$$F_1 = x; \quad F_{i+1} = F_i \cdot x^2/(4i^2+2i), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \operatorname{sh} x, \quad |x| < 10^6.$$

Варіант № 18

$$F_1 = 1; \quad F_{i+1} = F_i \cdot x^2/(4i^2-2i), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \operatorname{ch} x, \quad |x| < 10^6.$$

Варіант № 19

$$F_1 = x; \quad F_{i+1} = -F_i \cdot x^2(2i-1)^2/(4i^2+2i), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \operatorname{arsh} x, \quad |x| < 1.$$

Варіант № 20

$$F_1 = x; \quad F_{i+1} = F_i \cdot x^2(2i-1)/(2i+1), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \operatorname{arth} x, \quad |x| < 1.$$

Варіант № 21

$$F_0 = 1; \quad F_1 = x/2; \quad F_{i+1} = -F_i \cdot x(2i-1)/(2(i+1)), \quad i > 0;$$
$$\sum_{i=0}^{n} F_i = \sqrt{1+x}, \quad |x| < 1.$$

Варіант № 22

$$F_1 = x - 1; \quad F_{i+1} = -F_i \cdot i(x-1)/(i+1), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \ln x, \quad 0 < x < 2.$$

Варіант № 23

$$F_1 = (x-1)/x; \quad F_i = F_{i-1} \cdot (i-1)(x-1)/(ix), \quad i > 1;$$
$$\sum_{i=1}^{n} F_i = \ln x, \quad 0.5 < x.$$

Варіант № 24

$$F_1 = 1; \quad F_{i+1} = -F_i \cdot x^2/i, \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = e^{-x \cdot x}.$$

Варіант № 25

$$F_1 = x; \quad F_i = -F_{i-1} \cdot x(i-1)/i, \quad i > 1;$$
$$\sum_{i=1}^{n} Fi = \ln(1+x), \quad -1 < x < 1.$$

Варіант № 26

$$F_1 = x; \quad F_i = F_{i-1} \cdot x^2(2i-3)/(2i-1), \quad i > 1;$$
$$\sum_{i=1}^{n} F_i = 0.5 \ln((1+x)/(1-x)), \quad -1 < x < 1.$$

Варіант № 27

$$F_1 = 4/3; F_{i+1} = F_i \cdot (1 - 4x/3), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = 1/x, \quad 0.5 < x < 1.$$

Варіант № 28

$$F_1 = 1; \quad F_{i+1} = F_i \cdot x^2/(4i^2 - 2i), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \operatorname{ch} x.$$

Варіант № 29

$$F_1 = x/(0.525 + 0.5x)^2 - 1; \quad F_{i+1} = F_i \cdot F_1(3-2i)/(2i), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \sqrt{x}, \quad 0.5 < x < 1.$$

Варіант № 30

$$F_1 = x/(0.418 + 0.5x)^3 - 1; \quad F_{i+1} = F_i \cdot F_1(4-3i)/(3i), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \sqrt[3]{x}, \quad 0.5 < x < 1.$$

Варіант № 31

$$F_1 = x; \quad F_{i+1} = F_i \cdot x^2/(4i^2 + 2i), \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = \operatorname{sh} x.$$

Варіант № 32

$$F_1 = 1.951 - x; \quad F_{i+1} = 0.5F_i \cdot (1 - x \cdot F_1^2)(1 + i)/i, \quad i > 0;$$
$$\sum_{i=1}^{n} F_i = 1/\sqrt{x}, \quad 0.5 < x < 1.$$

Варіант № 33

$$F_1 = 1; \quad F_i = F_{i-1} \cdot x \cdot (\ln 2)/i, \quad i > 1;$$
$$\sum_{i=1}^{n} F_i = 2^x.$$

Лабораторна робота 2.
Зв'язані динамічні структури даних. Списки

Мета лабораторної роботи

Метою лабораторної роботи №2 «Зв'язані динамічні структури даних. Списки» є засвоєння теоретичного матеріалу та набуття практичного досвіду використання зв'язаних динамічних структур даних у вигляді одно- та двозв'язних списків при складанні різних алгоритмів.

Постановка задачі

1. Створити список з n ($n > 0$) елементів (n вводиться з клавіатури), якщо інша кількість елементів не вказана у конкретному завданні за варіантом.

2. Тип ключів (інформаційних полів) задано за варіантом.

3. Вид списку (черга, стек, дек, прямий однозв'язний лінійний список, обернений однозв'язний лінійний список, двозв'язний лінійний список, однозв'язний кільцевий список, двозв'язний кільцевий список) вибрати самостійно з метою найбільш доцільного розв'язку поставленої за варіантом задачі.

4. Створити функції (або процедури) для роботи зі списком (для створення, обробки, додавання чи видалення елементів, виводу даних зі списку в консоль, звільнення пам'яті тощо).

5. Значення елементів списку взяти самостійно такими, щоб можна було продемонструвати коректність роботи алгоритму програми. Введення значень елементів списку можна виконати довільним способом (випадкові числа, формування значень за формулою, введення з файлу чи з клавіатури).

6. Виконати над створеним списком дії, вказані за варіантом, та коректне звільнення пам'яті списку.

7. **При виконанні заданих дій, виводі значень елементів та звільненні пам'яті списку вважати, що довжина списку (кількість елементів) *невідома* на момент виконання цих дій**. Тобто, не дозволяється зберігати довжину списку як константу, змінну чи додаткове поле.

   При проєктуванні програм ***слід врахувати наступне***:

1) при виконанні завдання кількість операцій (зокрема, операцій читання й запису) має бути мінімізованою, а також максимально мають використовуватися властивості списків;

2) повторювані частини алгоритму необхідно оформити у вигляді процедур або функцій (для створення, обробки, виведення та звільнення пам'яті списків) з передачею списку за допомогою параметра(ів).

3) у таких видів списків, як черга, стек, дек функції для роботи зі списком мають забезпечувати роботу зі списком, відповідну тому чи іншому виду списку (наприклад, не можна додавати нові елементи всередину черги);

4) програми мають бути написані мовою програмування C.

Результати виконання роботи

Як результат виконання лабораторної роботи надіслати:

1) звіт до лабораторної роботи у форматі .PDF;

2) файл (або файли) з текстом програм (із розширенням .C).

Зміст звіту

1. Титульна сторінка.
2. Загальна постановка завдання.
3. Завдання для конкретного варіанту.
4. Текст усіх програм.
5. Скриншоти результатів тестування програм. На скриншотах має бути видно початкові дані та відповідні одержані результати.
6. Висновки.

Контрольні питання

1. Чи існують обмеження на кількість елементів у списку, якщо так, то які?
2. Які є види спискових структур даних з точки зору їх логічного використання (стек, черга, тощо). Поясніть їх особливості та відмінності між ними.
3. Які є вимоги до структури елемента зв'язних динамічних даних?
4. В чому полягає особливість опису типів для створення зв'язних динамічних даних?
5. Якою повинна бути структура елемента лінійного двозв'язного списку?
6. Скільки вказівників і якого призначення необхідно для роботи з чергою?
7. Скільки вказівників і якого призначення необхідно для роботи зі стеком?

8. Скільки вказівників і якого призначення необхідно для роботи з деком?

9. Скільки вказівників і якого призначення необхідно для роботи з лінійними однозв'язними списками?

10. Скільки вказівників і якого призначення необхідно для роботи з лінійними двозв'язними списками?

Варіанти індивідуальних завдань

Варіант № 1

Ключами елементів списку є рядки довжиною не більше 10-ти символів, що складаються з латинських літер. Відсортувати елементи списку у лексикографічному порядку, не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»), методом вибору.

Варіант № 2

Ключами елементів списку є дійсні числа. Кількість елементів списку повинна дорівнювати $\boldsymbol{2n}$. Обчислити значення виразу: $(a_1 - a_{2n})(a_3 - a_{2n-2}) \ldots (a_{2n-1} - a_2)$, де $\boldsymbol{a_i}$ — $\boldsymbol{i}$-й елемент списку.

Варіант № 3

Ключами елементів списку є символи з множини латинських літер та цифр. Перекомпонувати список таким чином, щоб усі цифри стояли на початку списку, не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»).

Варіант № 4

Заданий список є *чергою*. Ключами елементів списку є символи. Переписати елементи цієї черги до іншої черги у оберненому порядку, *не підраховуючи кількості елементів у черзі*.

Варіант № 5

Ключами елементів списку є дійсні числа. Виконати циклічний зсув елементів списку на $\boldsymbol{k}$ позицій вліво ($\boldsymbol{k}$ — натуральне і не перевищує кількості елементів списку). При необхідності дозволяється використати ще один список, інші структури даних, крім простих змінних, використовувати не дозволяється.

Варіант № 6

Ключами елементів списку є дійсні числа. Відсортувати елементи списку за незбільшенням, не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»), методом обміну («бульбашки») з використанням «прапорця».

Варіант № 7

Ключами елементів списку є цілі числа. Кількість елементів списку повинна дорівнювати $\boldsymbol{2n}$. Обчислити значення виразу: $a_1 \cdot a_{2n} + a_2 \cdot a_{2n-1} + \ldots + a_n \cdot a_{n+1}$, де $\boldsymbol{a_i}$ — $\boldsymbol{i}$-й елемент списку.

Варіант № 8

Ключами елементів списку є цілі ненульові числа, причому кількість від'ємних чисел дорівнює кількості додатних. Перекомпонувати список так, щоб отримати послідовність чисел із чергуванням знаків, не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»).

Варіант № 9

Ключами елементів списку є різні дійсні числа. Знайти максимальний та мінімальний елементи списку. Вставити до списку два нові елементи: після елемента з максимальним значенням вставити елемент з мінімальним значенням, а після елемента з мінімальним значенням вставити елемент з максимальним значенням.

Варіант № 10

Ключами елементів списку є цілі числа. Визначити кількість $\boldsymbol{p}$ елементів списку, значення яких більше за задане ціле число $\boldsymbol{M}$, та вставити $\boldsymbol{p}$ нових елементів після $\boldsymbol{k}$-го ($\boldsymbol{k}$ — натуральне і не перевищує кількості елементів списку) елементу списку.

Варіант № 11

Ключами елементів списку є латинські літери. Відсортувати елементи списку у лексикографічному порядку, не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»), методом вставки.

Варіант № 12

Ключами елементів списку є цілі числа. Обчислити значення виразу:
$(a_1 + a_2 + 2a_n)(a_2 + a_3 + 2a_{n-1}) \ldots (a_{n-1} + a_n + 2a_2)$, де $\boldsymbol{a_i}$ — $\boldsymbol{i}$-й елемент списку.

Варіант № 13

Ключами елементів списку є цілі числа. Переставити елементи списку так, щоб спочатку розташовувались додатні, потім нульові, а за ними від'ємні елементи, не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»).

Варіант № 14

Ключами елементів списку є рядки довжиною не більше 5-ти символів. Перекомпонувати список так, щоб елементи списку були розташовані в оберненому порядку (виконати «дзеркальне відображення» списку), не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»).

Варіант № 15

Ключами елементів списку є дійсні числа. Перекомпонувати елементи списку таким чином, щоб його елементи розташовувались у такому порядку: $a_1, a_n, a_2, a_{n-1}, ..., a_{[\frac{n+1}{2}]}$, де $\boldsymbol{a_i}$ — $\boldsymbol{i}$-й елемент списку. При необхідності дозволяється використати ще один список, інші структури даних, крім простих змінних, використовувати не дозволяється.

Варіант № 16

Ключами елементів списку є дійсні числа. Розширити список, дописавши в його кінець свої ж елементи, але у оберненому порядку, не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»).

Варіант № 17

Ключами елементів списку є дійсні числа. Обчислити значення виразу:
$a_1 \cdot a_n + a_2 \cdot a_{n-1} + \ldots + a_n \cdot a_1$, де $\boldsymbol{a_i}$ — $\boldsymbol{i}$-й елемент списку.

Варіант № 18

Ключами елементів списку є цілі ненульові числа, які розташовуються у наступному порядку: 10 додатних, 10 від'ємних і т. д. Кількість елементів списку $\boldsymbol{n}$ повинна бути кратною 20. Перекомпонувати список так, щоб розташування елементів було наступним: 5 додатних, 5 від'ємних і т. д., не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»).

Варіант № 19

Задано два списки, список $\boldsymbol{S1}$ довжиною $\boldsymbol{2n}$ елементів і список $\boldsymbol{S2}$ довжиною $\boldsymbol{n}$ елементів. Ключами елементів обох списків є натуральні числа. Вставити список $\boldsymbol{S2}$ у середину списку $\boldsymbol{S1}$, не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»).

Варіант № 20

Ключами елементів списку є цілі числа. Кількість елементів списку повинна дорівнювати $\boldsymbol{2n}$. Перекомпонувати елементи списку так, розташування елементів було наступним: $a_1, a_{n+1}, a_2, a_{n+2}, a_3, \ldots, a_n, a_{2n}$, де $\boldsymbol{a_i}$ — $\boldsymbol{i}$-й компонент списку, не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»).

Варіант № 21

Ключами елементів списку є дійсні числа. Виконати наступні дії: якщо елементи списку впорядковані за незбільшенням, то залишити його без змін, інакше перезаписати елементи списку у оберненому порядку. При необхідності дозволяється використати ще один список, інші структури даних, крім простих змінних, використовувати не дозволяється.

Варіант № 22

Ключами елементів списку є цілі ненульові числа, які розташовуються в наступному порядку: 5 від'ємних, 5 додатних і т. д. Кількість елементів списку $\boldsymbol{n}$ повинна бути кратною 20-ти. Перекомпонувати елементи списку так, щоб розташування елементів було наступним: 10 від'ємних, 10 додатних і т. д., не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»).

Варіант № 23

Ключами елементів списку є цілі числа. Виконати циклічний зсув елементів списку на $\boldsymbol{k}$ позицій вправо ($\boldsymbol{k}$ — натуральне і не перевищує кількості елементів списку). При необхідності дозволяється використати ще один список, інші структури даних, крім простих змінних, використовувати не дозволяється.

Варіант № 24

Ключами елементів списку є цілі ненульові числа. Кількість елементів списку $\boldsymbol{n}$ повинна бути кратною 4-ом, причому кількість від'ємних чисел дорівнює кількості додатних. Перекомпонувати елементи списку так, щоб розташування елементів було наступним: два додатні, два від'ємні і т. д., не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»).

Варіант № 25

Ключами елементів списку є латинські літери. Перекомпонувати список так, щоб спочатку розташовувались елементи з голосними латинськими літерами, а потім елементи з приголосними латинськими літерами, не змінюючи початкового взаємного розташування літер. Наприклад:

початковий список: university

результат: uieiynvrst

При необхідності дозволяється використати ще один список, інші структури даних, крім простих змінних, використовувати не дозволяється.

Варіант № 26

Ключами елементів списку є натуральні числа. Перекомпонувати список так, щоб спочатку йшли елементи з ключами, що діляться на 3 без залишку, потім елементи з ключами, що діляться на 3 із залишком 1, і нарешті ті, що діляться на 3 із залишком 2, не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»).

Варіант № 27

Ключами елементів списку є цілі числа. Перекомпонувати список так, щоб спочатку розташовувались додатні, потім нульові, а за ними від'ємні елементи, не змінюючи початкового взаємного розташування елементів. Наприклад:

початковий список: -1 0 5 -9 8 -3 5 0 -7 4

результат: 5 8 5 4 0 0 -1 -9 -3 -7.

При необхідності дозволяється використати ще один список, інші структури даних, крім простих змінних, використовувати не дозволяється.

Варіант № 28

Ключами елементів списку є цілі ненульові числа. Кількість елементів списку $\boldsymbol{n}$ повинна бути кратною 10-ти, а елементи у початковому списку розташовуватись із чергуванням знаків. Перекомпонувати список, змінюючи порядок чисел всередині кожного десятка елементів так, щоб спочатку йшли від'ємні числа цього десятка елементів, а за ними — додатні, не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»).

Варіант № 29

Ключами елементів списку є рядки довжиною не більше 25-ти символів. Кількість елементів списку $\boldsymbol{n}$ повинна бути кратною 20-ти. Перекомпонувати список всередині кожних 20-ти елементів, розташувавши їх у наступному порядку: $a_1, a_{11}, a_2, a_{12}, \ldots, a_{21}, a_{31}, a_{22}, a_{32}, \ldots$, де $\boldsymbol{a_i}$ — $\boldsymbol{i}$ елемент списку. При необхідності дозволяється використати ще один список, інші структури даних, крім простих змінних, використовувати не дозволяється.

Варіант № 30

Ключами елементів списку є цілі числа. Відсортувати елементи списку за незменшенням, не використовуючи додаткових структур даних, крім простих змінних (тобто «на тому ж місці»), методом шейкерного сортування.

Лабораторна робота 3.
Графічне представлення графів

Мета лабораторної роботи

Метою лабораторної роботи №3 «Графічне представлення графів» є набуття практичних навичок представлення графів у комп'ютері та ознайомлення з принципами роботи ОС.

Постановка задачі

1. Представити у програмі напрямлений і ненапрямлений графи з заданими параметрами:
   - кількість вершин n;
   - розміщення вершин;
   - матриця суміжності A.
2. Створити програму для формування зображення напрямленого і ненапрямленого графів у графічному вікні.

Згадані вище параметри графа задаються на основі чотиризначного номера варіанту $n_1n_2n_3n_4$, де n_1n_2 це десяткові цифри номера групи, а n_3n_4 — десяткові цифри номера варіанту, який був у студента для двох попередніх робіт (див. таблицю з поточними оцінками з АСД, надану викладачем на початку поточного семестру).

Кількість вершин n дорівнює $10 + n_3$.

Розміщення вершин:

- колом при $n_4 = 0, 1$;
- квадратом (прямокутником) при $n_4 = 2, 3$;
- трикутником при $n_4 = 4, 5$;

- колом з вершиною в центрі при $n_4 = 6, 7$;
- квадратом (прямокутником) з вершиною в центрі при $n_4 = 8, 9$.

Наприклад, при $n_4 = 9$ розміщення вершин прямокутником з вершиною в центрі повинно виглядати так, як на прикладі графа на рис. 8.

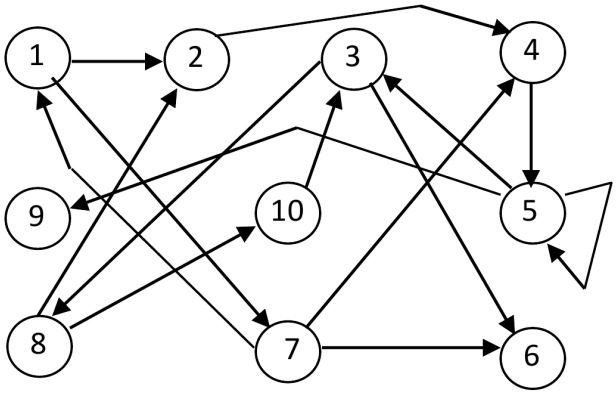


Рис. 8. Приклад зображення графа

Матриця суміжності A_{dir} напрямленого графа за варіантом формується таким чином:

1) встановлюється параметр (seed) генератора випадкових чисел, рівний номеру варіанту $n_1n_2n_3n_4$ — детальніше див. с. 12;
2) матриця розміром $n \cdot n$ заповнюється згенерованими випадковими числами в діапазоні [0, 2.0);
3) обчислюється коефіцієнт $k = 1.0 - n_3 * 0.02 - n_4 * 0.005 - 0.25$;
4) кожен елемент матриці множиться на коефіцієнт k;
5) елементи матриці округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент більший або дорівнює 1.0.

Матриця суміжності A_{undir} ненапрямленого графа одержується з матриці A_{dir}:

$$a_{dir_{i,j}} = 1 \Rightarrow \ a_{undir_{i,j}} = 1, a_{undir_{j,i}} = 1.$$

Наприклад, якщо запрограмувати додаткові функції `randm` та `mulmr`, у програмі на мові C генерація матриці A_{dir} напрямленого графа може виглядати так:

```
srand(n1n2n3n4);
T = randm(n);
k = 1.0 - n3*0.02 - n4*0.005 - 0.25;
A = mulmr(T, k);
```

Тут `randm(n)` — розроблена функція, яка формує матрицю розміром $n \cdot n$, що складається з випадкових чисел у діапазоні (0, 2.0);
`mulmr(T, k)` — розроблена функція множення матриці на коефіцієнт та округлення результату до 0 чи 1.

При проєктуванні програм ***слід врахувати наступне***:

1) мова програмування обирається студентом самостійно;

2) графічне зображення графа має формуватися на основі графічних примітивів з графічної бібліотеки (таких як еліпс, пряма, дуга, текст тощо);

3) використання готових бібліотек для роботи з графами не дозволяється;

4) вивід графа має бути реалізований універсальним чином: вершини і ребра мають виводитися в циклі, а не окремими командами для кожного графічного елемента;

5) типи та структури даних для внутрішнього представлення графа у програмі слід вибрати самостійно;

6) матриці суміжності графів можна виводити в графічне вікно або консоль — на розсуд студента;

7) матриці суміжності мають виводитися як матриці: в квадратному вигляді, з 1 та 0.

Результати виконання роботи

Як результат виконання лабораторної роботи надіслати:

1) звіт до лабораторної роботи у форматі .PDF;

2) файли з текстом програм:

   - або в архіві формату .zip або .7zip (***без файлів .exe***);
   - або як посилання на корінь підкаталогу репозиторію (github, bitbucket etc.) з проєктом.

Зміст звіту

1. Титульна сторінка.
2. Загальна постановка завдання та завдання для конкретного варіанту.
3. Текст програм.
4. Згенеровані за варіантом матриці суміжності напрямленого і ненапрямленого графів.
5. Скриншоти напрямленого і ненапрямленого графів, які побудовані за варіантом.
6. Висновки.

Лабораторна робота 4.
Характеристики та зв'язність графа

Мета лабораторної роботи

Мета лабораторної роботи №4 «Характеристики та зв'язність графа» — дослідити характеристики графів та навчитись визначати їх на конкретних прикладах, вивчення методу транзитивного замикання.

Постановка задачі

1. Представити напрямлений та ненапрямлений графи із заданими параметрами так само, як у лабораторній роботі №3.

   Відмінність: коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.01 - 0.3$.

   Отже, матриця суміжності A_{dir} напрямленого графа за варіантом формується таким чином:

   1) встановлюється параметр (seed) генератора випадкових чисел, рівне номеру варіанту $n_1 n_2 n_3 n_4$;
   2) матриця розміром $n \cdot n$ заповнюється згенерованими випадковими числами в діапазоні [0, 2.0);
   3) обчислюється коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.01 - 0.3$, кожен елемент матриці множиться на коефіцієнт k;
   4) елементи матриці округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент більший або дорівнює 1.0.

2. Обчислити:

   1) степені вершин напрямленого і ненапрямленого графів;
   2) напівстепені виходу та заходу напрямленого графа;
   3) чи є граф однорідним (регулярним), і якщо так, вказати степінь однорідності графа;

4) перелік висячих та ізольованих вершин.

Результати вивести у графічне вікно, консоль або файл.

3. ***Змінити матрицю*** A_{dir}, коефіцієнт $k = 1.0 - n_3 * 0.005 - n_4 * 0.005 - 0.27$.

4. Для нового орграфа обчислити:

   1) півстепені вершин;
   2) всі шляхи довжини 2 і 3;
   3) матрицю досяжності;
   4) матрицю сильної зв'язності;
   5) перелік компонент сильної зв'язності;
   6) граф конденсації.

   Результати вивести у графічне вікно, в консоль або файл.

   Шляхи довжиною 2 і 3 слід шукати за матрицями A^2 і A^3, відповідно. Як результат вивести перелік шляхів, включно з усіма проміжними вершинами, через які проходить шлях.

   Матрицю досяжності та компоненти сильної зв'язності слід шукати за допомогою операції транзитивного замикання. У переліку компонент слід вказати, які вершини належать до кожної компоненти.

   Граф конденсації вивести у графічне вікно.

При проєктуванні програми ***слід врахувати наступне***:

1) мова програмування обирається студентом самостійно;

2) графічне зображення усіх графів має формуватися програмою з тими ж вимогами, як у ЛР №3;

3) всі графи, включно із графом конденсації, обов'язково зображувати у графічному вікні;

4) типи та структури даних для внутрішнього представлення всіх даних у програмі слід вибрати самостійно;

5) обчислення перелічених у завданні результатів має виконуватися розробленою програмою (не вручну і не сторонніми засобами);

6) матриці, переліки степенів та маршрутів тощо можна виводити в графічне вікно або консоль — на розсуд студента;

7) у переліку знайдених шляхів треба вказувати не лише початок та кінець шляху, але й усі проміжні вершини, через які він проходить (наприклад, 1 – 5 – 3 – 2).

Результати виконання роботи

Як результат виконання лабораторної роботи надіслати:

1) звіт до лабораторної роботи у форматі .PDF;

2) файли з текстом програм:

   - або в архіві формату .zip або .7zip (***без файлів .exe***);
   - або як посилання на корінь підкаталогу репозиторію (github, bitbucket etc.) з проєктом.

Зміст звіту

1. Титульна сторінка.

2. Загальна постановка завдання та завдання для конкретного варіанту.

3. Текст програм.

4. За п. 1 завдання: згенеровані матриці суміжності напрямленого та ненапрямленого графів.

5. За п. 2: перелік степенів, півстепенів, результат перевірки на однорідність, переліки висячих та ізольованих вершин.

6. За п. 3: матриця другого орграфа.

7. За п. 4: переліки півстепенів, шляхів, матриці досяжності та сильної зв'язності, перелік компонент сильної зв'язності, граф конденсації.

8. Скриншоти заданих орієнтованого і неорієнтованого графів, модифікованого графа та його графа конденсації.

9. Висновки.

Оскільки завдання має велику кількість підпунктів, при оформленні звіту можна чергувати результати з пунктами завдання. Деякі результати (наприклад, переліки шляхів) мають велику довжину, тож їх можна навести не у вигляді скриншотів, а у вигляді тексту, скопійованого з консолі, вікна чи файлу, куди вони виводились.

Лабораторна робота 5.
Обхід графа

Мета лабораторної роботи

Метою лабораторної роботи №5 «Обхід графа» є вивчення методу дослідження графа за допомогою обходу його вершин в глибину та в ширину.

Постановка задачі

1. Представити напрямлений граф із заданими параметрами так само, як у лабораторній роботі №3.

   Відмінність: коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.005 - 0.15$.

   Отже, матриця суміжності A_{dir} напрямленого графа за варіантом формується таким чином:

   1) встановлюється параметр (seed) генератора випадкових чисел, рівне номеру варіанту $n_1 n_2 n_3 n_4$;
   2) матриця розміром $n \cdot n$ заповнюється згенерованими випадковими числами в діапазоні [0, 2.0);
   3) обчислюється коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.005 - 0.15$, кожен елемент матриці множиться на коефіцієнт k;
   4) елементи матриці округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент більший або дорівнює 1.0.

2. Створити програму, яка виконує обхід напрямленого графа вшир (BFS) та вглиб (DFS).

   - обхід починати з вершини із найменшим номером, яка має щонайменше одну вихідну дугу;
   - при обході враховувати порядок нумерації;

- у програмі виконання обходу відображати покроково, черговий крок виконувати за натисканням кнопки у вікні або на клавіатурі.

3. Під час обходу графа побудувати дерево обходу. У програмі дерево обходу виводити покроково у процесі виконання обходу графа. Це можна виконати одним із двох способів:

   - або виділяти іншим кольором ребра графа;
   - або будувати дерево обходу поряд із графом.

4. Зміну статусів вершин у процесі обходу продемонструвати зміною кольорів вершин, графічними позначками тощо, або ж у процесі обходу виводити протокол обходу у графічне вікно або в консоль.

5. Якщо після обходу графа лишилися невідвідані вершини, продовжувати обхід з невідвіданої вершини з найменшим номером, яка має щонайменше одну вихідну дугу.

При проєктуванні програми також ***слід врахувати наступне***:

1) мова програмування обирається студентом самостійно;

2) графічне зображення усіх графів має формуватися програмою з тими ж вимогами, як у ЛР №3;

3) всі графи обов'язково зображувати у графічному вікні;

4) типи та структури даних для внутрішнього представлення всіх даних у програмі слід вибрати самостійно.

Результати виконання роботи

Як результат виконання лабораторної роботи надіслати:

1) звіт до лабораторної роботи у форматі .PDF;

2) файли з текстом програм:

   - або в архіві формату .zip або .7zip (***без файлів .exe***);
   - або як посилання на корінь підкаталогу репозиторію (github, bitbucket etc.) з проєктом.

Зміст звіту

1. Титульна сторінка.
2. Загальна постановка завдання та завдання для конкретного варіанту.
3. Текст програми.
4. Згенерована матриця суміжності напрямленого графа.
5. Матриця суміжності дерева обходу.
6. Список (вектор) відповідності номерів вершин і їх нової нумерації, набутої в процесі обходу.
7. Скриншоти зображення графа та дерева обходу: 1) на початку обходу, 2) у процесі обходу та 3) після завершення обходу.
8. Висновки.