

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторного практикуму
та самостійної роботи
з дисципліни
„ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ“
(частина друга)
для студентів спеціальності
152 Метрологія та інформаційно-вимірювальна техніка
Галузь знань:
15 Автоматизація та приладобудування
денної й заочної форм навчання

Методичні вказівки до лабораторного практикуму та самостійної роботи з дисципліни „Обчислювальна техніка та програмування“ (частина друга) для студентів спеціальності 152 „Метрологія та інформаційно-вимірювальна техніка“ Галузь знань: 15 "Автоматизація та приладобудування" денної й заочної форм навчання /Укл.: В.І.Рева. – Запоріжжя: НУ «Запорізька політехніка», 2019. – 85 с.

Укладачі: В. І. Рева, доц., канд. фіз.-мат. наук

Рецензент: О.В. Василенко, доц., канд. техн. наук

Відповідальний за випуск: А. В. Коротун, доц., канд. фіз.-мат. наук

Затверджено
на засіданні кафедри
мікро- та наноелектроніки

Протокол №2
від “ 30 “ вересня 2019 р.

Рекомендовано до видання
НМК ФРЕТ
Протокол №3
від “ 03 “ жовтня 2019 р.

ЗМІСТ

Вступ	111
Підготовка до проведення лабораторної роботи та оформлення звіту	113
4 Лабораторна робота № 4 «Створення інтерфейсу користувача у текстовому режимі»	114
4.1 Бібліотека PDCurses	114
4.2 Функції введення і виведення PDCurses	116
4.2.1 Висновок на екран	116
4.2.2 Робота з кольором	117
4.2.3 Введення з клавіатури	119
4.2.4 Управління курсором	122
4.3 Вікна та панелі PDCurses	124
4.3.1 Вікна	124
4.3.2 Панелі	128
4.4 Робота з мишею в PDCurses	130
4.5 Практичне завдання	138
4.5.1 Варіанти завдань	139
5 Лабораторна робота № 5 «Інтерполяція алгебраїчними многочленами»	141
5.1 Введення в завдання інтерполяції	141
5.2 Графічна бібліотека WinBGI	145
5.2.1 Графічний курсор, вказівник	147
5.2.2 Малювання точок і ліній	148
5.2.3 Малювання прямокутників	150
5.2.4 Малювання кіл і еліпсів	151
5.2.5 Виведення тексту в графічному режимі	152
5.3 Побудова графіків функцій	154
5.4 Практичне завдання	159
6 Лабораторна робота № 6 «Чисельне інтегрування»	161
6.1 Основні методи чисельного інтегрування	161
6.2 Практичне завдання	164
6.2.1 Варіанти завдань	164
7 Лабораторна робота № 7 «Наближене рішення нелінійних рівнянь»	166
7.1 Основні методи чисельного рішення рівнянь	166
7.1.1 Метод половинного ділення	167

7.1.2 Метод хорд	168
7.1.3 Метод ітерацій	169
7.1.4 Метод дотичних (Ньютона)	169
7.2 Практичне завдання	171
7.2.1 Варіанти завдань	172
Додаток А Приклад оформлення титульної сторінки	173
Додаток Б «Створення графічного інтерфейсу користувача засобами бібліотеки Qt»	174

ВСТУП

У лабораторному практикумі розглядаються теоретичні і практичні відомості, необхідні для оволодіння навичками програмування на мовах C і C ++. Практикум складається з 7 лабораторних робіт. В описі кожної лабораторної роботи приведені теоретичні відомості та індивідуальні завдання.

Ні для кого не секрет, що значна частина сучасних радіотехнічних пристроїв побудована на базі мікропроцесорної техніки. Найяскравішими прикладами є мобільні телефони, пейджери, МР3-плеєри, автомобільні сигналізації, пристрої управління інжекторами і т.д. Розробляються численні пристрої, що працюють спільно з міцно ввійшли в повсякденне життя персональними комп'ютерами. Тому сучасному радіоінженеру життєво необхідно володіти навичками програмування.

Серед розробників радіоапаратури загальноприйнятим стандартом при написанні програм є застосування мов C і C ++. Існує багато книг про C / C ++. Однак структура викладу матеріалу в них, як правило, відрізняються від прийнятої при вивченні програмування у ВНЗ. Мабуть, це пов'язано з протиріччям між необхідністю досить суворого викладу матеріалу в лекції.

Його розумілого роз'яснення (найчастіше з випередженням) на практичних заняттях. Помітний вплив на процес навчання програмування надає специфіка самого мови C ++. З одного боку, це мова, що вивчається поступово. З іншого боку, оволодіння практичними навичками програмування вимагає випереджаючого вивчення, а так-же повернення і переосмислення вже вивченого.

Найважливішу роль при навчанні грає лабораторні практикум. Вважається, що навчитися програмувати можна лише в результаті практичної діяльності. По-цьому написання цієї книги продиктовано необхідністю структурувати виклад матеріалу оптимальним для виконання лабораторних робіт способом. Книга складається зі вступу, дванадцяти розділів і висновку. Глави 1, 3, 4, 6, 10 є теоретичними. Їх мета доведення до читача мінімальних відомостей, необхідних для виконання лабораторних робіт.

Разом з їх вивченням необхідно користуватися книгами з повним систематичним викладом мови C / C ++. Глави 2, 5, 7, 8, 9, 11, 12

крім теоретичних відомостей містять описи семи лабораторних робіт, приклади виконання завдань і самі завдання.

Хочеться звернути особливу увагу читачів на те, що в даному посібнику намічена канва вивчення, є посилання на більш об'ємні джерела, сформульовані завдання і намічені шляхи їх вирішення. Використання тільки цієї книги недостатньо для формування стійкого досвіду програмування. Слід обов'язково розширювати теоретичні знання, використовуючи книги з бібліографічного списку.

Навчальний посібник призначений для студентів спеціальності 210302 "Радіотехніка" очної форми навчання (в тому числі за скороченою програмою), які вивчають дисципліну "Інформатика".

ПІДГОТОВКА ДО ПРОВЕДЕННЯ ЛАБОРАТОРНОЇ РОБОТИ ТА ОФОРМЛЕННЯ ЗВІТУ

Лабораторна робота включає самостійне опрацювання теоретичного матеріалу, вивчення документації з програмних засобів, методик проведення розрахунків, програмування і моделювання, обробку й інтерпретацію результатів. При проведенні лабораторного практикуму необхідно:

- підготуватися до експрес-опитування з теоретичного матеріалу, необхідного для виконання роботи (щоб отримати допуск до лабораторної роботи);
- підготувати і оформити план виконання лабораторної роботи;
- після виконання лабораторної роботи провести необхідні розрахунки і оформити звіт, який повинен містити:
 - 1) титульний аркуш (див. Додаток А);
 - 2) мету роботи;
 - 3) опис словами;
 - 4) необхідні розрахунки;
 - 5) текст розробленої програми;
 - 6) основні результати моделювання;
 - 7) стислі висновки по роботі, в яких:
 - вказати, що досліджувалось;
 - вказати методику (чи метод) дослідження;
 - вказати особливості алгоритму роботи;
 - визначити характер роботи;
 - відзначити подібність і відмінність результатів моделювання від теоретичних даних;
- здати звіт з виконаної лабораторної роботи та відповісти на запитання викладача (отримати залік за виконану роботу).

4 ЛАБОРАТОРНА РОБОТА № 4 «СТВОРЕННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА У ТЕКСТОВОМУ РЕЖИМІ»

Мета роботи: ознайомлення з принципами побудови інтерактивних програм; ознайомлення з бібліотекою PDCurses; освоєння принципів програмування меню; освоєння прийомів роботи з вікнами в текстовому режимі.

Для взаємодії користувача з програмою програміст повинен передбачити будь-якої інтерфейс. Консольні додатки, які виконуються в командному рядку, використовують текстовий інтерфейс. Для створення текстового інтерфейсу прийнято використовувати різні бібліотеки, що містять функції управління режимами екрану, роботи з розширеними кодами клавіатури, кольором, вікнами і звуком.

4.1 Бібліотека PDCurses

Серед Unix-систем як правило використовується бібліотека ncurses. Вільним аналогом цієї бібліотеки в середовищі Windows є бібліотека PDCurses. Встановлювати її необхідно в ту ж папку, в яку встановлений компілятор GCC. Пакет програм, що реалізують функції компілятора GCC для Windows називається MinGW і знаходиться звичайно в одній з папок: `c:\MinGW`, `c:\Program Files\MinGW` або `c:\Program Files\CodeBlocks\MinGW`. Для використання бібліотеки PDCurses, необхідно включити в програму заголовки бібліотеки

```
#include <curses.h>
```

Тоді стають доступними наступні типи даних, визначені в бібліотеці:

- `chtype` тип символів, з якими працює PDCurses (він включає в себе код символу, колір і додаткові атрибути);
- `bool` логічний тип (значення цього типу може бути помилковим `FALSE` або істинним `TRUE`);
- `SCREEN` структура, що зберігає дані терміналу (терміналом називають екран, на який здійснюється висновок; для Windows він

збігається з консольним вікном командного рядка, можливо розкритим на весь екран комбінацій клавіш Alt + Enter);

– **WINDOW** структура, що зберігає дані вікна (всередині терміналу можна визначати текстові вікна).

Програми на основі PDCurses використовують для організації призначеного для користувача інтерфейсу наступні поняття: екран (screen), вікно (window) і подокно (sub-window). Екран являє собою матрицю осередків, в які можна виводити символи. Вікно прямокутна частина екрана. При використанні PDCurses програма зазвичай має наступну структуру:

```
...
#include <curses.h>
...
initscr ();
робота з pdcurses
endwin ();
...
```

Функція `initscr ()` виконує ініціалізацію даних, необхідних для роботи бібліотеки. При цьому створюється вікно з ім'ям `stdscr`, розмір якого збігається з роз-ром всього екрану. Після цього можна використовувати функції PDCurses. При завершенні роботи з бібліотекою потрібно викликати функцію `endwin ()`. Розглянемо приклад

```
1.  #include <curses.h>
2.  int main(void)
3.  {
4.    // инициализация (должна быть выполнена
5.    ///перед использованием PDCurses)
    initscr();
6.    //перемещение курсора в стандартном экране
    y=10 x=30 move(10,30);
7.    printf("Hello world !!!"); // вывод строки
    refresh(); // обновить экран
8.    getch(); // ждём нажатия символа endwin();
9.    // завершение работы с PDCurses } //.
```

Тут використані наступні функції бібліотеки:

- `move (y, x)` переміщує курсор в зазначені координати `x` і біля екрану, початок координат знаходиться в лівому верхньому кутку екрану, вісь `X` спрямована вправо, вісь `Y` вниз;
- `printw ("Рядок")` друкує рядок в поточній позиції курсору;
- `refresh ()` оновлює змінилася частина екрану.

Бібліотека `PDCurses` дозволяє прискорити виведення на екран за рахунок того, що вона використовує запис не відразу на екран, а

буфер пам'яті, і коли потрібно відобразити всі зміни, використовується функція `refresh ()`.

Компіляція програми, що використовує бібліотеку `ncurses` в середовищі Unix-систем не викликає труднощів. У Windows необхідно вказати компоновальнику місце розташування файлу `pdcurses.a` скомпільованій версії бібліотеки. Наприклад, при роботі в інтегрованому середовищі розробки `CodeBlocks`, необхідно скористатися пунктом меню "Settings, Compiler and debugger" і на закладці "Linker settings" діалогового вікна в список "Link libraries" додати шлях до бібліотеки (як правило `C: \ Program Files \ CodeBlocks \ MinGW \ PDCurses \ win32 \ pdcurses.a`).

4.2 Функції введення і виведення `PDCurses`

Стандартні функції введення-виведення, описані в бібліотеках мови C (C++) не можна використовувати при програмуванні інтерфейсу `PDCurses`. Це пов'язано з тим, що після ініціалізації `PDCurses` функцією `initscr ()` висновок на екран повинен відбуватися з використанням стандартного вікна `stdscr`. Тому для введення і виведення повинні використовуватися функції, оголошені в `PDCurses`.

4.2.1 Висновок на екран

Для виведення на екран в стандартне вікно `stdscr` передбачені наступні функції:

- `printw ()` аналог функції `printf ()`;
- `addch (chtype X)` аналог функції `putchar` виводить символ `X` в поточну позицію курсору і переміщує курсор на один символ вправо. Якщо курсор знаходився біля правої межі екрану, то він переміщається в початок наступного рядка;

- `insch (ctype X)` вставляє символ `X` зліва від курсора, а всі символи праворуч від курсора переміщуються на одну позицію вправо;
 - `addchnstr (ctype * S, int n)` виводить перші `n` символів рядка `S` починаючи з поточної позиції курсора. Рядок `ctype * S` являє собою масив символів типу `ctype`. Якщо `n = -1`, то виводиться вся рядок;
 - `addstr (char * S)` виводить рядок `S` починаючи з поточної позиції курсору. Рядок `S` являє собою масив символів типу `char`;
 - `insnstr (char * S, int n)` вставляє перші `n` символів рядка `S` у позицію, де розташований курсор, положення курсору не змінюється. Якщо `n = -1`, виводиться вся рядок;
 - `insertln ()` вставляє порожній рядок в поточну позицію курсору.
- Символи типу `ctype` відрізняються від звичайних символів типу `char` наявністю у символу атрибутів миготіння (`A_BLINK`), підвищеної яскравості (`A_BOLD`), нормального відображення (`A_NORMAL`), зниженої яскравості (`A_DIM`), підкреслення (`A_UNDERLINE`), інверсного зображення (`A_REVERSE`), а також інформації про колір. Наприклад, операція
- `ctype C = 'a' | A_BOLD;`
- присвоює змінній `C` значення символу `'a'` підвищеної яскравості.

4.2.2 Робота з кольором

Перед використанням кольорів потрібно проініціалізувати палітру. Функція `has_colors ()` дозволяє перевірити, чи можна використовувати кольори. Вона повертає істинне значення, якщо можна працювати з кольором. Функція `start_color ()` включає підтримку кольору. Далі необхідно призначити номери використовуваних квітів. Номер 0 зарезервований для стандартного відображення. Номери починаючи з 1 можна використовувати в програмі. Кожному номеру необхідно задати колір символу і колір фону з наступного списку:

```
COLOR_BLACK чорний,
COLOR_RED червоний,
COLOR_GREEN зелений,
COLOR_YELLOW жовтий,
COLOR_BLUE синій,
COLOR_MAGENTA малиновий,
```

COLOR_CYAN блакитний,
COLOR_WHITE білий.

Для цього передбачена функція `init_pair ()`. Першим її аргументом є номер кольору з палітри, другим і третім найменування квітів символу і фону відповідно, наприклад:

```
// Цвет с номером 1
init_pair(1, COLOR_RED, COLOR_WHITE);
// Цвет с номером 2
init_pair(2, COLOR_YELLOW, COLOR_BLUE);
```

Тим самим буде сформована палітра кольорів. Тепер можна використовувати атрибути символів, що позначають колір

```
chtype C = 'a' | COLOR_PAIR (1); //.
```

Тут змінної `C` присвоюється значення символу 'а' з номером кольору 1 з палітри.

При виведенні на екран рядків або символів, для яких не вказані атрибути використовуються стандартні атрибути. Замінити атрибути можна за допомогою наступних функцій.

Функція `int attron (int A)` включає атрибут `A`. По суті перелічені вище назви кольорів, а також атрибути символів, є константами цілого типу, певними в бібліотеці `PDCurses`. Тому атрибут `A` в прототипі функції має цілий тип. Наприклад, щоб вивести на екран повідомлення підвищеної яскравості кольором з номером 2 з палітри, встановлюємо два атрибути `COLOR_PAIR (2)`, `A_BOLD` і виводимо повідомлення

```
attron (COLOR_PAIR (2));
attron (A_BOLD);
printw ( "Повідомлення !!!");
```

Функція `int attroff (int A)` вимикає атрибут `A`.

Функція `void bkgdset (int A)` встановлює атрибути з якими очищається екран. Очистити екран можна за допомогою функції `clear ()`. При необхідності включення або виключення відразу декількох атрибутів їх необхідно перераховувати, розділяючи вертикальною лінією.

Як приклад наведемо програму виведення на екран яскравого вітання.

```

1.  #include <curses.h>.
2.  int main(void)
3.  {
4.  // ініціалізуємо екран initscr();
5.  // разрешаем работу с цветом start_color();
6.  // ініціалізуємо палітру
7.  // додаємо колір з номером 1
8.  init_pair(1, COLOR_RED, COLOR_GREEN);
9.  // встановлюємо яскравість символів
   attron(A_BOLD);
10. // устанавливаем цвет
   attron(COLOR_PAIR(1));
11. // переміщуємо курсор
12. move(10, 15);
13. // Виводимо привітання
14. printw("Добро пожаловать!!!\n");
15. // выключаем яркость символов
   attroff(A_BOLD);
16. // вмикаємо мерцання
17. attron(A_BLINK);
18. // переміщуємо курсор move(15, 15);
19. // Виводимо рядок printw("PDCurses");
20. // Обновляем экран refresh();
21. // Затримка до натискання клавіші getch();
22. // вмикаємо PDCurses endwin();
23. }
```

4.2.3 Введення з клавіатури

Для введення з клавіатури передбачені наступні функції:

– scanw () аналог функції scanf ();

- `getch (void)` аналог функції `getchar ()` повертає введений символ;
- `getstr (char * S)` вводить рядок в змінну `S`;
- `getnstr (char * S, int n)` вводить рядок в змінну `S`, але не довше `n` символів.

При побудові користувацького інтерфейсу досить важну роль відіграє функція `getch ()`. Найчастіше інтерфейс користувача реалізується у вигляді меню, керованого так званими клавішами управління. Ці клавіші істотно відрізняються від алфавітно-цифрових. До них відносяться функціональні клавіші, клавіші управління курсором і безліч інших керуючих комбінацій, що виникають, наприклад, при використанні модифікуючих клавіш `Shift`, `Ctrl`, `Alt`.

При натисканні керуючої клавіші в потік вводу поступає не один, а ціла послідовність кодів клавіш. Використовуючи бібліотеку `PDCurses`, можна запрограмувати реакцію програми на натискання будь-яких клавіш, включаючи керуючі. Для цього по-перше необхідно перевести термінал в такий режим роботи, коли Зауважте, що ви відразу передаються програмі без очікування натискання `Enter`. Це досягається викликом функції `cbreak ()`. Зворотній їй функція `ncbreak ()` повертає колишні налаштування. По-друге необхідно включити за допомогою функції

```
int keypad (WINDOW * win, bool bf);
```

режим обробки програмою керуючих клавіш. Перший аргумент цієї функції вказує, для якого вікна включається обробка керуючих клавіш. Поки нам відомо тільки головне стандартне вікно `stdscr`. Другий аргумент логічного типу `bool` може приймати значення `TRUE` (обробка включена) або `FALSE` (обробка вимкнена).

Послідовність кодів, введену з клавіатури можна інтерпретувати як ціле число, що складається з декількох байт. Тому результатом роботи функції `getch ()` є значення типу `int`, а не `char`. Коди символів, що лежать в діапазоні від 32 до 127, відносяться до символів стандартної таблиці `ASCII` (див. Додаток). Коди від 128 до 255 представляють собою другу половину таблиці символів і залежать від використовуваної кодової сторінки (для російської мови: `CP866`, `CP1251` або `KOI8-R`). Решта коди належать керуючим клавішам і їх комбінацій.

Для зручності в бібліотеці PDCurses оголошені константи, що дозволяють використовувати осмислені імена замість кодів керуючих клавіш. Далі перераховані деякі з них:

```
KEY_DOWN = 0x102 стрілка вниз,
KEY_UP   = 0x103 стрілка вгору,
KEY_LEFT = 0x104 стрілка вліво,
KEY_RIGHT = 0x105 стрілка вправо,
KEY_HOME = 0x106 Home,
KEY_BACKSPACE = 0x107 Backspace,
KEY_DC    = 0x14a Delete,
KEY_IC    = 0x14b Insert,
KEY_ENTER = 0x157 Enter,
KEY_END   = 0x166 End,
KEY_NPAGE = 0x152 Page Down,
KEY_PPAGE = 0x153 Page Up,
KEY_F (n) функціональні клавіші, де n може приймати значення від 0 до 63. Повний перелік таких констант можна знайти в файлі curses.h.
```

Функція `noecho()` відключає автоматичне дублювання символів, що вводять на екрані, а функція `echo()` включає дублювання.

Функція `halfdelay (int time)` обмежує час очікування введення символу часом `time` (вимірюється в десятих частках секунди). Якщо протягом цього часу жодна клавіша не було натиснуто, функція `getch()` повертає значення `ERR = -1`. Скасувати такий режим введення можна за допомогою функції `posbreak()`.

Розглянемо кілька прикладів. Нижче наведена програма, яка відображає на екрані коди натискаючих клавіш до тих пір, поки не натиснута клавіша Esc (код Esc дорівнює 27).

```
1.  #include <curses.h>
2.  int main(void)
3.  {
4.  // инициализация экрана
5.      initscr();
6.  // разрешение работы с цветом
7.      start_color();
8.  // добавление цвета с номером 1
```

```

9.          init_pair(1, COLOR_RED, COL-
            OR_GREEN);
10. // заливка екрана зелёним цветом
11.          bkgdset(COLOR_PAIR(1));
12.          clear();
13. // включение обработки управляющих клавиш
            keypad(stdscr, TRUE);
14. // Обновление экрана
15. refresh();
16. Переменная для ввода int C;
17. do
18. {C=getch();    // Чтение с клавиатуры
19. clear();      // Очистка экрана
20. move(10,15); // Перемещение курсора
21. printf("%d",C); //Вывод кода
22. refresh();} //Обновление экрана
23. while (C!=27); // Пока не Esc
24. // выключаем PDCurses
25. endwin();
26. return 0;
27. }

```

4.2.4 Управління курсором

У попередніх прикладах була використана функція `move (y, x)` позиціонування курсору в осередок екрану з вказаними координатами. Така послідовність завдання координат (спочатку `y`, потім `x`) обумовлена наступними обставинами. З точки зору `PDCurses` екран являє собою прямокутну матрицю осередків, в яких можуть відображатися символи. Положення елемента в матриці зазвичай визначається номером рядка і номером стовпця. Номер рядка за змістом збігається з координатою `y`, а номер стовпчика з координатою `x`.

Бібліотека `PDCurses` містить ще кілька функцій для роботи з курсором.

Функція `getyx (WINDOW * win, int y, int x)` дозволяє отримати поточні координати курсору щодо зазначеного вікна `win`, наприклад, щодо головного вікна `stdscr`.

Функції

```
int getmaxx(WINDOW *win),
int getmaxy(WINDOW *win)
```

видають максимальні значення координат курсору по горизонталі і вертикалі щодо зазначеного вікна.

Наступна програма надає користувачам можливість пересувати курсор за допомогою клавіш управління курсором (стрілки).

```
1.  #include <curses.h>
2.  int main(void)
3.  {
4.  // инициализация экрана
5.  initscr();
6.  // включение обработки управляющих клавиш
   keypad(stdscr, TRUE);
7.  // Обновление экрана
8.  refresh();
9.  // Переменная для ввода
10. int C;
11. // Текущие координаты курсора
12. int x=10;
13. int y=10;
14. move(y,x);
15. // Максимальные координаты курсора
16. int maxx=getmaxx(stdscr);
17. int maxy=getmaxy(stdscr);
18. do
19. {C=getch(); // Чтение с клавиатуры // Ана-
   лиз нажатой клавиши
20. if ((C==KEY_LEFT)&&(x>0)) x--;
21. if ((C==KEY_RIGHT)&&(x<maxx)) x++;
```

```

22. if ((C==KEY_UP)&&(y>0)) y--;
23. if ((C==KEY_DOWN)&&(y<maxy)) y++;
24. move(y,x); // Перемещение курсора
25. refresh(); //Обновление экрана
26. while (C!=27); // Пока не Esc // выключение
    PDCurses
27. endwin();
28. return 0;
29. }

```

4.3 Вікна та панелі PDCurses

У бібліотеці PDCurses передбачена можливість роботи з вікнами в текстовому режимі. Вікно являє собою прямокутну область екрана. Насправді при роботі з PDCurses завжди доводиться працювати з вікнами, так як при ініціалізації бібліотеки створюється стандартне вікно `stdscr` розміром в екран. Описані вище функції введення-виведення, управління виводу і курсором призначені для роботи зі стандартним вікном і тільки з ним. Для роботи з іншими вікнами необхідно використовувати модифіковані аналоги цих функцій. Модифікація як правило полягає в тому, що ім'я функції використовується з приставкою "w" (наприклад, `wprintw` замість `printw`) і першим аргументом модифікованих функцій є ім'я використовуваного вікна.

4.3.1 Вікна

Перерахуємо далі функції для роботи з вікнами.

```
WINDOW * newwin (int lines, int cols, int y, int x)
```

— створює вікно з координатами лівого верхнього кута (x, y), в якому lines рядків і cols стовпців.

```
WINDOW * subwin (WINDOW * parent, int lines, int cols, int y, int x)
```

– створює під вікно з координатами лівого верхнього кута (x, y) щодо всього екрану, в якому lines рядків і cols стовпців, parent батьківське вікно.

```
WINDOW * derwin (WINDOW * parent, int lines, int
cols, int y, int x)
```

– створює підвікно в якому lines рядків і cols стовпців з координатами лівого верхнього кута (x, y) щодо батьківського вікна parent.

```
int delwin (WINDOW * win)
```

– видаляє вікно / підвікно win.

```
int mvwin (WINDOW * win, int y, int x)
```

– переміщує вікно win в нову позицію (x, y) щодо екрану.

```
int mvderwin (WINDOW * win, int y, int x)
```

– переміщує під вікно win в нову позицію (x, y) щодо батьківського вікна.

Проілюструємо використання вікон прикладом. Модифікуємо програму виведення кодів клавіш так, щоб використовувалися вікна. Початок програми традиційне для PDCurses

```
1.  #include <curses.h>
2.  int main(void)
3.  {
4.  // инициализация экрана
5.  initscr();
6.  // разрешение работы с цветом
7.  start_color();
8.  // добавление цвета с номером 1
9.  init_pair(1, COLOR_RED, COLOR_GREEN);
10. init_pair(2, COLOR_RED, COLOR_BLUE);
11. // заливка экрана зелёным цветом
12. bkgdset(COLOR_PAIR(1));
13. clear();// Далі обчислюються розміри екрану
    максимальні значення коор-динат курсору
```

```

14. int maxx=getmaxx(stdscr);
15. int maxy=getmaxy(stdscr); //і створюється
    вікно у верхній частині екрану для виведення
    заголовка програми
16. WINDOW * topwin = newwin (3, maxx, 0,0);
17. // Його висота становить три рядки, а шири-
    на збігається з шириною екрану. Вікна topwin
    призначається колір очищення з номером 2 і
    виконується очищення вікна,
18. wbgdset (topwin, COLOR_PAIR (2));
19. wclear (topwin);
20. // тим самим вікно заливається синім кольо-
    ром. // Далі організуємо висновок у вікно
    topwin.
21. // Назначение атрибутов цвета вывода
    watttrn(topwin,COLOR_PAIR(2));
22. // Рисование рамки окна
23. box(topwin,'|','-' );
24. // Перемещение курсора в координаты 25,1
25. wmove(topwin,1,25);
26. // Вывод строки
27. wprintw(topwin,"Программа вывода кодов кла-
    виш"); //. Аналогічно створюється друге
    вікно, в яке будуть виводитися коди натискаю-
    чих клавіш
28. // Создание окна
29. WINDOW *mildwin=newwin(5,10,5,30);
30. // Назначение атрибутов цвета очистки
    wbgdset(mildwin,COLOR_PAIR(2));
31. // Очистка окна
32. wclear(mildwin);
33. // Назначение атрибутов цвета вывода
    watttrn(mildwin,COLOR_PAIR(2));

```

```

34. // Рисование рамки окна
35. box(mildwin,'*', '*'); //. При виведенні
    будь-яких даних у вікно mildwin намальована
    зі-рочками рамка може бути стерта. Щоб уник-
    нути цього створимо підвікно трохи меншого
    розміру всередині вікна mildwin.
36. // Создание подокна
37. WINDOW *submildwin=derwin(mildwin,3,8,1,1);
38. // Назначение атрибутов цвета очистки
    wbkgdset(submildwin,COLOR_PAIR(1));
39. // Очистка окна
40. wclear(submildwin);
41. // Назначение атрибутов цвета вывода
    watttrn(submildwin,COLOR_PAIR(1)); //.
42. Далі оновлюється зображення у всіх вікнах і
    запускається цикл читання з клавіатури, поки
    Ви не натиснете Esc (код 27). Код ко-жної
    натиснутої клавіші виводиться в підвікно
    submildwin з попе-реднім його очищенням.
43. // включение обработки управляющих клавиш
    keypad(submildwin,TRUE);
44. // Обновление экрана
45. refresh();
46. // Обновление окна topwin
47. wrefresh(topwin);
48. // Обновление окна mildwin
49. wrefresh(mildwin);
50. // Обновление подокна submildwin
51. wrefresh(submildwin);
52. // Переменная для ввода
53. int C;
54. do

```

```

55. { // Чтение с клавиатуры из окна submildwin
    C=wgetch(submildwin);
56. // Очистка окна submildwin
57. wclear(submildwin);
58. // Перемещение курсора
59. wmove(submildwin,1,2);
60. //Вывод кода клавиши
61. wprintw(submildwin,"%d",C);
62.
63. // Обновление окна submildwin
64. wrefresh(submildwin);
65. }
66. while (C!=27); // Пока не Esc // выключение
    PDCurses
67. endwin();
68. return 0; //.
69.

```

Вікна в PDCurses знаходяться в одній площині екрану і при накладенні можуть прати вміст один одного. Для усунення цього недоліку в PDCurses існує поняття панелі.

4.3.2 Панелі

Панель являє собою вікно, що володіє властивістю глибини. Глибина вказує в якому порядку розташовуються панелі на екрані. Кожну панель можна перемістити вгору або вниз щодо інших панелей, а також приховати і показати. Для використання панелей потрібно включити в програму заголовки `panel.h`. Панель створюється на основі вже існуючого вікна за допомогою функції

```
PANEL * new_panel (WINDOW * win)
```

і поміщається в вершину стека панелей, тобто на верхній шар, наприклад,

```

PANEL * panel1;
panel1 = new_panel (topwin);

```

Перелічимо основні функції для роботи з панелями, реалізовані в PDCurses:

```
int del_panel (PANEL * panel1)
```

– видаляє панель `panel1`. Вікно, пов'язане з панеллю, не видаляється. При необхідності його потрібно видалити окремо.

```
WINDOW * panel_window (const PANEL * panel1)
```

– повертає покажчик на вікно пов'язане з панеллю `panel1`.

```
int hide_panel (PANEL * panel1)
```

– приховує панель. При цьому панель видаляється з стека панелей.

```
int show_panel (PANEL * panel1)
```

– показує приховану панель. При цьому панель поміщається в стек панелей на самий верхній рівень.

```
int top_panel (PANEL * panel1)
```

– переміщує панель на самий верх, поверх інших панелей.

```
int bottom_panel (PANEL * panel1)
```

– переміщує панель нижче всіх панелей.

```
int move_panel (PANEL * pannel1, int y, int x)
```

– переміщує панель так, щоб її верхній лівий кут опинився в точці (x, y).

```
int replace_panel (PANEL * panel1, WINDOW * win)
```

– змінює в панелі `panel1` поточне вікно на інше вікно `win`.

```
PANEL * panel_above (const PANEL * pan)
```

– повертає вказівник на панель, яка знаходиться вище на один рівень. Якщо такої панелі немає, то повертає 0.

```
PANEL * panel_below (const PANEL * pan)
```

– повертає вказівник на панель, яка знаходиться нижче на один рівень. Якщо такої панелі немає, то повертає 0.

```
void update_panels ()
```

– оновлює стек панелей. Для відображення змін на екрані потрібно викликати функцію `doupdate ()`.

4.4 Робота з мишею в PDCurses

У бібліотеці PDCurses робота з мишею реалізована в такий спосіб. Функція `mousemask ()`, дозволяє включити або вимкнути відстеження подій від миші. Її першим аргументом є так звана бітова маска, в якій вказується, які події необхідно від-злежуються. Для основних комбінацій бітових масок перед-Дивися константи, наприклад, `BUTTON1_CLICKED` – клацання лівою кнопкою миші, `BUTTON1_DOUBLE_CLICKED` – подвійне клацання лівою кнопкою миші, `BUTTON1_TRIPLE_CLICKED` – потрійний клацання лівою кнопкою миші, `ALL_MOUSE_EVENTS` – відстеження всіх подій від миші.

Другим аргументом функції `mousemask ()` може бути або вказівник на допоміжну змінну, призначену для тимчасового зберігання попередньої бітової маски, або константа `NULL`.

Виявлення події, з генерованого мишею, рівнозначно виявлення в стандартні потоці введення коду `KEY_MOUSE = 539`. Такий підхід дозволяє роботу з мишею об'єднати в один цикл з обробкою натискань клавіш клавіатури. Наступний цикл буде виконуватися поки не відбудеться будь-яка подія від миші

```
int C;
keypad (stdscr, TRUE);
do
C = getch ();
while (C! = KEY_MOUSE);
```

Замість стандартного вікна `stdscr` тут можна використовувати будь-яке інше вікно або панель.

Після виявлення події миші його необхідно вважати з черги. Сама подія є структурою даних, в якій зазначено, яке саме подія відбулася, координати миші при його настанні і т.д. У PDCurses передбачений спеціальний тип даних `MEVENT`, а також функція `nc_getmouse (MEVENT *)` зчитує події з черги. Структура `MEVENT` є об'єднанням

різнотипних змінних, серед яких присутні змінні x і y , що містять екранні координати покажчика миші під час настання події. Скористатися ними можна за допомогою уточненого імені: `ІмяСтруктури.x`, `ІмяСтруктури.y`. Наступний фрагмент програми відображає у верхньому лівому кутку екрана координати клацань миші

```
int C;
 keypad(stdscr, TRUE);
do
{
 move(0,0);
 C=getch();
 if (C==KEY_MOUSE)
 {
  clear();
  MEVENT M;
  nc_getmouse(&M);
  printf("%d      %d", M.x, M.y);
 }
}
while(C!=27);
```

Приклад.

Нехай необхідно написати програму, що реалізовує перемикання активного вікна за допомогою миші.

У верхній частині екрану розташуємо заголовне вікно, в якому буде відображатися номер активного вікна і координати клацання мишею. У середній частині екрана будуть перебувати три вікна, одне з яких може бути активним. Ознакою його активності виберемо червоний колір фону. Клацання миші на будь-якому з трьох вікон повинен робити його активним. Клацання правіше вікон знімає активність з усіх вікон.

Оскільки дії з вікнами середньої частини екрану є однотипними, доцільно описати масив з трьох вікон, а саме

```
// Описуємо масив вікон
WINDOW *win[3];      //.
```

Спочатку головною функції main ініціалізується екран, робота з кольором, призначаються номери кольорів палітри

```
// инициализация экрана
initscr();
// разешение работы с цветом
start_color();
// добавление цвета с номером 1, 2, 3
init_pair(1, COLOR_WHITE, COLOR_GREEN);
init_pair(2, COLOR_WHITE, COLOR_BLUE);
init_pair(3, COLOR_WHITE, COLOR_RED); //.
```

Колір з номером 1 будемо використовувати для заливки всього екрану, колір з номером 2 для заливки неактивного вікна, а колір з номером 3 для заливки активного вікна. Далі очищаємо весь екран, обчислюємо максимальні значення координат курсору

```
// заливка экрана зелёным цветом
bkgdset(COLOR_PAIR(1)); clear();
// вычисление максимальных координат
int maxx=getmaxx(stdscr);
int maxy=getmaxy(stdscr);
і створюємо заголовне вікно
// создание окна
WINDOW *topwin=newwin(3,maxx,0,0);
// назначение цвета очистки
wbkgdset(topwin,COLOR_PAIR(2));
// назначение атрибутов вывода
wattron(topwin,COLOR_PAIR(2));
//очистка окна
wclear(topwin);
//рисование рамки окна
box(topwin,'|','-' );
// перемещение курсора в координаты 25,1
wmove(topwin,1,25);
// вывод сообщения
wprintw(topwin,"No click!!!"); //.
```

Тут створено вікно з ім'ям topwin розміром 3 рядки і maxx стовпців з координатами лівого верхнього кута (0,0). Колір з номером 2

палітри (білий на синьому) призначений для очищення і для виведення в це вікно. Після очищення вікна намальована рамка і виведений текст "No click !!!".

Наступний фрагмент створює вікна, що належать масиву win.

```
int k; // переменная цикла
for (k=0; k<=2; k++)
{
    // создание окна с номером k
    win[k]=newwin(10, 25, 5, k*25+k);
    wbkgdset(win[k], COLOR_PAIR(2));
    wclear(win[k]);
    watttrn(win[k], COLOR_PAIR(2));
    box(win[k], '|', '-');
} //.
```

Кожне з них має розмір 10 рядків і 25 стовпців. Вікна розташовані горизонтально, тому координата у лівого верхнього кута дорівнює 5, а координата x обчислюється за формулою $x = k \cdot 25 + k$, де k номер вікна. Вікна заливуються синім кольором і вимальовуються рамки. Далі виконується оновлення екрану, очищення всіх вікон, включення реакції програми на клацання мишею і натискання функціональних клавіш,

```
// Обновление экрана
refresh();
// Обновление окна topwin
wrefresh(topwin);
// Обновление окон массива win
for (k=0; k<=2; k++) {wrefresh(win[k]);}
// Включение обработки событий мыши mouse-
mask(ALL_MOUSE_EVENTS, NULL);
// Включение обработки функциональных клавиш key-
pad(stdscr, TRUE);
```

а також оголошуються цілі змінні C і active для зберігання коду натиснутої клавіші і номера активного вікна відповідно:

```
int C;
int active=0; //.
```

Наступна частина програми є цикл обробки повідомлень від миші:

```
// начало цикла пока не нажата клавиша Esc
do
{
// Считывание кода нажатой клавиши
C=getch();
// если есть события от мыши
if (C==KEY_MOUSE)
{
// Считывание события из очереди
MEVENT M;
nc_getmouse(&M);
// Анализ координат нажатия
if ((M.y>=5)&&(M.y<15)) {active=M.x/25;}
// Вывод номера активного окна
wmove(topwin,1,25); wprintw(topwin,"Active%d %d
%d",active,M.x,M.y); wrefresh(topwin);
// Перерисовка окон
repaint(active);
}
}
while(C!=27); //.
```

Якщо клацання миші стався в діапазоні вертикальних координат $5 \leq y < 15$, то змінюється активне вікно. Номер активного вікна є результатом цілочисельного ділення координати натискання на розмір вікон по осі OX. Наприклад, клацання в позиції з координатами $y = 10$, $x = 38$ робить активним середнє вікно, так як $38 = 25 = 1$.

При зміні активного вікна потрібно виконати перемальовування вікон з масиву win. Для цього передбачимо функцію repaint (int a). Як параметр функція приймає номер активного вікна a і перемальовує всі вікна. Перемальовування активного і неактивних вікон відрізняється тільки номером використовуваного кольору.

```
void repaint(int a)
{
int k; // переменная цикла
// цикл по окнам
```

```

for(k=0;k<=2;k++)
{
    if (k==a) // если окно активное
    {
        // то назначается цвет 3
        wbkgdset(win[k],COLOR_PAIR(3));
        //очистка окна
        wclear(win[k]);
        // рисование рамки
        box(win[k], '|', '-');
        //обновление окна
        wrefresh(win[k]);
    }
    else
    {
        //иначе назначается цвет 2
        wbkgdset(win[k],COLOR_PAIR(2));
        //очистка окна
        wclear(win[k]);
        //рисование рамки
        box(win[k], '|', '-');
        // обновление окна
        wrefresh(win[k]);
    }
}
} //.
```

Змінна циклу *k* по черзі приймає значення номера перемальовувати вікна. Якщо цей номер збігається з номером активного вікна, то відбувається перемальовування з номером кольору 3. Всі інші вікна перемальовувати з номером кольору 2. Нижче наведено повний текст розглянутої програми.

1. `#include <curses.h>`
2. `// массив окон`
3. `WINDOW *win[3];`
4. `// функция перерисовки окон`
5. `void repaint(int a)`

```

6.  {
7.  int k; // переменная цикла
8.  // цикл по окнам
9.  for(k=0;k<=2;k++)
10. {
11. if (k==a) // если активное окно
12. {
13. // то назначается цвет 3
14. wbgdset(win[k],COLOR_PAIR(3));
15. //очистка окна
16. wclear(win[k]);
17. // рисование рамки
18. box(win[k], '|', '-');
19. // обновление окна
20. wrefresh(win[k]);
21. }
22. else
23. {
24. // иначе назначается цвет 2
25. wbgdset(win[k],COLOR_PAIR(2));
26. //очистка окна
27. wclear(win[k]);
28. // рисование рамки
29. box(win[k], '|', '-');
30. обновление окна
31. wrefresh(win[k]);
32.     }
33. }
34. }
35. // Главная функция
36. int main(void)
37. {
38. initscr(); // инициализация экрана

```

```

39. start_color(); // разешение работы с цветом
40. // добавление цветов в палитру
41. init_pair(1, COLOR_WHITE, COLOR_GREEN);
42. init_pair(2, COLOR_WHITE, COLOR_BLUE);
43. init_pair(3, COLOR_WHITE, COLOR_RED);
44. // заливка экрана зелёным цветом
45. bkgdset(COLOR_PAIR(1));
46. clear();
47. int maxx=getmaxx(stdscr);
48. int maxy=getmaxy(stdscr);
49. // Создание заголовочного окна
50. WINDOW *topwin=newwin(3,maxx,0,0); wbkgd-
    set(topwin,COLOR_PAIR(2));
51. wclear(topwin);
52. wattron(topwin,COLOR_PAIR(2));
53. box(topwin,'|','-' );
54. wmove(topwin,1,25);
55. wprintw(topwin,"No click!!!");
56. // Создание окон массива win
57. int k;
58. for(k=0;k<=2;k++)
59. {
60. win[k]=newwin(10,25,5,k*25+k);
61. wbkgdset(win[k],COLOR_PAIR(2));
62. wclear(win[k]);
63. wattron(win[k],COLOR_PAIR(2));
64. box(win[k],'|','-' );
65. }
66. // Обновление экрана и окон
67. refresh();
68. wrefresh(topwin);
69. for(k=0;k<=2;k++) {wrefresh(win[k]);}
70.

```

```

71. // Включение обработки событий мыши
    mousemask(ALL_MOUSE_EVENTS, NULL);
72. // Включение обработки функциональных клавиш keypad(stdscr, TRUE);
73. int C; // Переменная для ввода
74. int active;
75. // Цикл обработки событий
76. do
77. {
78. C=getch();
79. if (C==KEY_MOUSE)
80. {
81. MEVENT M;
82. nc_getmouse(&M);
83. if ((M.y>=5)&&(M.y<15)) {active=M.x/25;}
84. wmove(topwin,1,25);
85. wprintw(topwin,"Active%d %d
    %d",active,M.x,M.y);
86. wrefresh(topwin);
87. repaint(active);
88. }
89. }
90. while(C!=27);
91.          endwin();
92. return 0;
93. }

```

4.5 Практичне завдання

При виконанні практичного завдання дотримуйтесь наступної послідовності дій.

- 1) Обов'язково вивчіть опис, запропоноване вище.
- 2) Розберіть приклади, що наводяться в описі.
- 3) Осмисліть завдання, запропоновану в вашому варіанті.
- 4) Складіть словесний алгоритм вирішення задачі.

5) Уявіть алгоритм рішення у вигляді блок-схеми.

6) Переведіть алгоритм на мову програмування.

4.5.1 Варіанти завдань

Варіант 1.

Реалізуйте на екрані горизонтальне меню, кероване клавішами управління курсором. При виборі пунктів меню повинні відбуватися якісь дії.

Варіант 2.

Реалізуйте на екрані вертикальне меню, кероване клавішами управління курсором. При виборі пунктів меню повинні відбуватися якісь дії.

Варіант 3.

Реалізуйте на екрані меню, кероване мишею. При виборі пунктів меню повинні відбуватися якісь дії.

Варіант 4.

Реалізуйте на екрані дворівневе меню, кероване клавішами управління курсором. При виборі пунктів меню повинні відбуватися якісь дії.

Варіант 5.

Реалізуйте переміщення по екрану прямокутника, кероване клавішами управління курсором, з можливістю призупинення руху і його продовження.

Варіант 6.

Напишіть програму розв'язання системи трьох лінійних рівнянь. Передбачте інтерфейс користувача.

Варіант 7.

Здійсніть перегляд в вікні меншого, ніж екран розміру текстового файлу з можливістю прокрутки тексту.

Варіант 8.

У текстовому режимі екрана реалізуйте можливість малювання (будь-якими символами) прямокутників за допомогою клавіш управління курсором.

Варіант 9.

У текстовому режимі екрана реалізуйте можливість малювання (будь-якими символами) ліній за допомогою клавіш управління курсором.

Варіант 10.

У текстовому режимі екрана реалізуйте можливість малювання (будь-якими символами) прямокутників за допомогою миші.

Варіант 11.

У текстовому режимі намальований символами об'єкт (прямокутник, лінія) повинні переміщатися під керуванням клавіш управління курсором.

Варіант 12.

На екрані присутні два текстових вікна. В одне з них завантажуються текстовий файл. В іншому при натисканні цієї клавіші відображається той же текст, зашифрований азбукою Морзе.

Варіант 13.

На екрані присутні два текстових вікна. В одне з них виводиться вміст текстового файлу. В іншому - при натисканні цієї клавіші відображається той же текст, але зашифрований. Наприклад, до коду кожного символу додається константа.

Варіант 14.

Напишіть програму розрахунку визначника матриці 3x3. Передбачте інтерфейс користувача.

Варіант 15.

Користувач набирає в відведеному для цього вікні текстовий файл. При натисканні F12 текст записується в файл.

5 ЛАБОРАТОРНА РОБОТА № 5 «ІНТЕРПОЛЯЦІЯ АЛГЕБРАЇЧНИМИ МНОГОЧЛЕНАМИ»

Мета роботи: ознайомлення з постановкою завдання інтерполяції; ознайомлення з графічною бібліотекою; придбання навичок побудови графіків.

5.1 Введення в завдання інтерполяції

Нехай між якимись величинами x і y існує функціональна залежність $y = f(x)$, аналітичний вигляд якої невідомий. В результаті експерименту отримані пари значень (x_k, f_k) , $f_k = f(x_k)$, $k = \overline{0, n, n+1}$ - кількість виконаних вимірювань. Тим самим функція $f(x)$ задана таблицею значень в деяких точках (вузлах інтерполяції) відрізка $[a; b]$.

Завдання інтерполяції полягає в підборі на основі експериментальних даних функції $g(x)$ в деякому сенсі близькою до $f(x)$. Будемо шукати функцію $g(x)$ в класі алгебраїчних многочленів виду

$$g(x) = P_n(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n \quad (5.1)$$

Очевидно функція $P_n(x)$ повністю визначається коефіцієнтами $a_k, k = \overline{0, n}$. Вимагатимемо, щоб значення многочлена $P_n(x)$ збігалися з виміряними значеннями функції y у вузлах інтерполяції, в результаті отримаємо систему лінійних рівнянь щодо a_k :

$$\begin{cases} a_0 + a_1x_0 + a_2x_0^2 + \dots + a_nx_0^n = f_0, \\ a_0 + a_1x_1 + a_2x_1^2 + \dots + a_nx_1^n = f_1, \\ \dots \\ a_0 + a_1x_k + a_2x_k^2 + \dots + a_nx_k^n = f_k, \\ \dots \\ a_0 + a_1x_n + a_2x_n^2 + \dots + a_nx_n^n = f_n, \end{cases} \quad (5.2)$$

Рішення системи рівнянь (5.2) дозволяє висловити коефіцієнти a_k через експериментальні дані $(x_k; f_k)$, підставивши які в формулу (5.1), можна отримати вираз для інтерполяційного многочлена. Система рівнянь (5.2) має єдине рішення, яке можна записати порізно. Широко поширені інтерполяційні многочлени в формі Лагранжа і Ньютона.

Інтерполяційна формула Лагранжа дозволяє уявити многочлен (5.1) у вигляді

$$P_n(x) = \sum_{k=0}^n f_k \frac{(x-x_0)\dots(x-x_{k-1})(x-x_{k+1})\dots(x-x_n)}{(x_k-x_0)\dots(x_k-x_{k-1})(x_k-x_{k+1})\dots(x_k-x_n)} =$$

$$= \sum_{k=0}^n f_k \frac{\prod_{j=0, j \neq k}^n (x-x_j)}{\prod_{j=0, j \neq k}^n (x_k-x_j)}.$$

Інтерполяція за формулою Лагранжа досить просто реалізується у вигляді підпрограми, але має істотний недолік: при додаванні нових вузлів інтерполяційний многочлен потрібно будувати заново.

Подолати зазначений недолік дозволяє використання інтерполяційної формули Ньютона. Вона висловлює многочлен $P_n(x)$ через значення $f(x)$ в одному з вузлів і так звані розділені різниці.

Для побудови розділених різниць будемо вважати, що серед точок x_k немає співпадаючих. Розділеними різницями першого порядку називаються величини

$$f_{i,j} = \frac{f_j - f_i}{x_j - x_i}, \quad i, j = \overline{0, n}, \quad i \neq j.$$

На основі розділених різниць першого порядку, складеним по сусіднім вузлам $f_{0,1}, f_{1,2}, \dots, f_{n-1,n}$, можна побудувати розділені різниці другого порядку

$$f_{0,1,2} = \frac{f_{1,2} - f_{0,1}}{x_2 - x_0},$$

$$f_{1,2,3} = \frac{f_{2,3} - f_{1,2}}{x_3 - x_1},$$

$$f_{n-2,n-1,n} = \frac{f_{n-1,n} - f_{n-2,n-1}}{x_n - x_{n-2}}.$$

Аналогічно визначаються розділені різниці більш високого ($k + 1$) порядку по вже відомим різницям порядку k :

$$f_{j,j+1,\dots,j+k,j+k+1} = \frac{f_{j+1,j+2,\dots,j+k+1} - f_{j,j+1,j+k}}{x_{j+k+1} - x_j}.$$

Використання розділених різниць дозволяє записати інтерполяційний многочлен Ньютона для інтерполяції вперед (початковою точкою обрана x_0)

$$P_n(x) = f_0 + f_{0,1}(x - x_0) + f_{0,1,2}(x - x_0) + \dots + f_{0,1,\dots,n-1,n}(x - x_0)(x - x_1)\dots(x - x_{n-1}) \quad (5.3)$$

і для інтерполяції назад (початковою точкою обрана x_n)

$$P_n(x) = f_n + f_{n-1,n}(x - x_n) + f_{n-2,n-1,n}(x - x_n)(x - x_{n-1}) + \dots + f_{0,1,\dots,n-1,n}(x - x_n)(x - x_{n-1})\dots(x - x_1) \quad (5.4)$$

Нумерація вузлів при проведенні розрахунків за формулою Ньютона не має значення. Це дозволяє легко додавати нові вузли. Наприклад, додавання одного вузла x_{n+1} призводить до появи в інтерполяційному многочлені (5.3) доданка виду

$$f_{0,1,\dots,n,n+1}(x - x_0)(x - x_1)\dots(x - x_{n-1})(x - x_n).$$

Розглянемо тепер один із способів розрахунку коефіцієнтів многочлена (5.3) в програмі. Зауважимо, що коефіцієнтами многочлена є одне із значень функції в початковому вузлі $f(x_0)$, одна з розділених різниць першого порядку $f_{0,1}$, одна з розділених різниць другого порядку $f_{0,1,2}$. Отже, інші значення функції і розділених різниць є проміжними даними. Нехай значення x_k і f_k розташовані в одновимірних

масивах з $n + 1$ елементів, зображених для ілюстрації в першій і другій рядку табл. 9.

Почнемо обчислювати розділені різниці з кінця масиву. Після обчислення величини $f_{n-1,n} = (f_n - f_{n-1}) / (x_n - x_{n-1})$ значення f_n , що займає останнє місце в масиві більше не буде потрібно. Тому обчислену кінцеву різницю можна зберегти на останньому місці в масиві f , як показано в рядку 3 таблиці 9. Аналогічно після обчислення величини $f_{n-2,n-1} = (f_{n-1} - f_{n-2}) / (x_{n-1} - x_{n-2})$ не потрібно передостанній елемент масиву f_{n-1} . Тому на його місці можна зберегти обчислену різницю (рядок 4) і т.д. В результаті обчислення всіх розділених різниць першого порядку

```
for(k=n;k>=1;k--){f[k]=(f[k]-f[k-1])/(x[k]-x[k-1]);}
```

вміст масиву f буде відповідати рядку 5 табл. 5.1.

Таблиця 5.1 – Обчислення кінцевих різниць

1	x_0	x_1	x_2	...	x_{n-1}	x_n
2	f_0	f_1	f_2	...	f_{n-1}	f_n
3	f_0	f_1	f_2	...	f_{n-1}	$f_{n-1,n}$
4	f_0	f_1	f_2	...	$f_{n-2,n-1}$	$f_{n-1,n}$
...						
5	f_0	$f_{0,1}$	$f_{1,2}$	...	$f_{n-2,n-1}$	$f_{n-1,n}$
6	f_0	$f_{0,1}$	$f_{0,1,2}$	...	$f_{n-2,n-1}$	$f_{n-2,n-1,n}$
...						
7	f_0	$f_{0,1}$	$f_{0,1,2}$	...	$f_{0,1,...,n-1}$	$f_{0,1,...,n-1,n}$

Розділені різниці другого порядку обчислюються на основі розділених різниць першого порядку і зберігаються в елементах масиву f [2] ... f [n]:

```
for (k = n; k>= 2; k--)  
{f [k] = (f [k] -f [k-1]) / (x [k] -x [k-2]);} //.
```

Розділені різниці порядку j обчислюються на основі розділених різниць порядку $j-1$ і зберігаються в елементах масиву f [j] ... f [n]:

```
for (k = n; k>= j; k--)  
{f [k] = (f [k] -f [k-1]) / (x [k] -x [k-j]);} //.
```

В результаті масив f буде містити коефіцієнти інтерполяційного многочлена Ньютона, а всі проміжні дані будуть загублені (рядок 7).

Обчислення значень інтерполяційного многочлена найбільш економічно проводити за схемою Горнера, одержуваної шляхом послідовного винесення за дужки множників $(x - x_i)$

$$P_n(x) = f_0 + (x - x_0)[f_{0,1} + (x - x_1)[f_{0,1,2} + \dots + (x - x_{n-2})[f_{0,1,\dots,n-1} + (x - x_{n-1})f_{0,1,\dots,n}]]\dots].$$

5.2 Графічна бібліотека WinBGI

У графічному режимі область виведення на екран (це може бути весь екран монітора або графічне вікно) являє собою прямокутну матрицю точок, колір кожної з яких можна змінювати окремо.

Для ефективного програмування графічного режиму доцільно використовувати різні графічні бібліотеки. Однією з них є бібліотека WinBGI аналог WinBGI фірми Borland. Ця бібліотека містить велику кількість графічних процедур і функцій. Передбачити запуск графічного режиму можна в будь-якому місці програми. Для цього потрібно включити в програму заголовний файл `graphics.h` і описати дві допоміжні змінні цілого типу

```
#include <graphics.h>
...
int gd, gm; //.
```

Змінна `gd` (graphic driver) використовується для зберігання номера (коду) графічного драйвера, а `gm` (graphic mode) графічного режиму. Імена змінних `gd` і `gm` можуть бути будь-якими. Перед ініціалізацією графіки необхідно присвоїти значення змінним `gd` і `gm`. Однак можна надати вибір найкращого драйвера і режиму функції автовизначення параметрів графіки

```
void detectgraph (int * graphdriver, int * graphmode);
```

Можливі значення номера графічного драйвера наведені в табл. 5.2. Найчастіше на персональних комп'ютерах використовується графічний драйвер VGA з номером 9 з одним з трьох можливих режимів, перерахованих в табл. 5.3.

Таблиця 5.2 – Графічні драйвери

Назва	Номер
DETECT	0 (автовизначення)
CGA	1
MCGA	2
EGA	3
ECGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

Таблиця 5.3 – Режими VGA

Назва	Номер	Розширення	Палітра
VGA0	0	640x200	16 кольорів
VGA1	1	640x350	16 кольорів
VGA2	2	640x480	16 кольорів

Ініціалізацію графічного режиму виконує функція

```
void initgraph (int * graphdriver,
int * graphmode, char * path);
```

Тут параметр path шлях до файлу графічного драйвера. При автовизначення параметрів як параметр path доцільно використовувати порожній рядок. При успішному виконанні функції initgraph буде створено додаткове вікно графічного виведення програми. Після закінчення роботи з графічним режимом його необхідно деініціалізувати (закрити) за допомогою функції

```
void closegraph (int w = ALL_WINDOWS); //.
```

Оскільки є можливість відкриття кількох графічних вікон, параметр функції w дозволяє вказати, яке саме графічне вікно необхідно

закрити. Цей параметр може приймати тільки одне з двох значень: `CURRENT_WINDOW` (закриття поточного вікна) і `ALL_WINDOWS` (закриття всіх графічних вікон, використовується за умовчанням).

У наступному прикладі виконується ініціалізація графічного режиму і його закриття після натискання будь-якої клавіші.

```

1.  #include <iostream.h>
2.  #include <graphics.h>
3.  int main()
4.  {
5.  int gd, gm;
6.  detectgraph(&gd, &gm);
7.  initgraph(&gd, &gm, "");
8.  getch();
9.  closegraph();
10. return 0;
11. }  //.
```

Для кодування кольору в бібліотеці `WinBGIm` використовується нумерація кольорів цілими числами. Передбачені також символічні назви кольорів, наведені в табл. 2.4.

5.2.1 Графічний курсор, вказівник

Процес побудови зображень в графічному вікні спирається на систему координат з початком в лівому верхньому кутку вікна. Ось `X` спрямована вправо, вісь `Y` - вниз. Аналогом текстового курсору в графічному режимі є невидимий поточний вказівник. Найчастіше малювання об'єктів виконується щодо поточного вказівника. Максимальне значення координат вказівника залежить від дозволу (кількості точок, що розміщуються у вікні). Отримати максимально можливі координати можна за допомогою функцій

```

int getmaxx (void);
int getmaxy (void);  //.
```

Таблиця 5.4 – Символічні назви кольорів

Назва	Номер
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7
DARKGRAY	8
LIGHTBLUE	9
LIGHTGREEN	10
LIGHTCYAN	11
LIGHTRED	12
LIGHTMAGENTA	13
YELLOW	14
WHITE	15

Переміщення покажчика без промальовування на екрані здійснюється за допомогою однієї з функцій

```
void moveto (int x, int y);
void moverel (int dx, int dy); //.
```

Перша з них переміщує покажчик в зазначені координати, друга зміщує покажчик на величини dx і dy щодо поточної позиції.

Дізнатися поточні координати покажчика можна за допомогою функцій

```
int getx (void);
int gety (void); //.
```

5.2.2 Малювання точок і ліній

У бібліотеці WinBGIм вивід точки здійснює функція

```
void putpixel (int x, int y, int color);
```

де x , y координати розташування точки, $color$ колір.

Наприклад, команда

```
putpixel (getmaxx () / 2, getmaxy () / 2, GREEN);
```

виводить в центрі екрану зелену точку.

Для побудова відрізків прямих служать функції `line`, `lineto`, `linereel`. Функція

```
void line (int x1, int y1, int x2, int y2);
```

малює лінію з початком в точці з координатами (x_1, y_1) і кінцем в точці (x_2, y_2) . Слід зауважити, що багато функцій малювання графічних примітивів, зокрема функція `line`, не мають параметра установки кольору. У цьому випадку колір задається перед малюванням об'єкта функцією

```
void setcolor (int color);
```

наприклад,

```
setcolor (RED);
```

```
line (0, 0, 100, 200); //.
```

Аналогічно функція `setbkcolor (int color)` призначає поточний колір фону.

Функції

```
int getcolor (void);
```

```
int getbkcolor (void);
```

дозволяють отримати номер поточного кольору вказівника і фону відповідно, а функція

```
unsigned getpixel (int x, int y);
```

– номер кольору точки вікна з вказаними координатами x, y . Функція

```
void lineto (int x, int y);
```

будує лінію з точки поточного положення покажчика в точку з координатами x, y .

Функція

```
void linerel (int dx, int dy);
```

будує лінію між точкою поточного положення покажчика (x, y) і точкою $(x + dx, y + dy)$.

Допускається установка стилю лінії функцією

```
void setlinestyle (int A, unsigned B, int C);
```

де параметр A визначає тип лінії, можливі значення якого описані у вигляді констант і наведені в табл. 5.5.

Таблиця 5.5 – Стилі ліній

Назва	Номер	Опис
SOLID_LINE	0	суцільна
DOTTED_LINE	1	пунктирна
CENTER_LINE	2	штрих-пунктирна
DASHED_LINE	3	штрихова
USERBIT_LINE	4	стиль, що задається зразком

Параметр `C` визначає товщину лінії. Можливі значення товщини: `NORM_WIDTH = 1` (лінія в 1 піксель) і `THICK_WIDTH = 3` (жирна лінія товщиною в 3 пікселі).

Параметр `B` є зразком стилю і використовується, якщо стиль лінії дорівнює `USERBIT_LINE` (в іншому випадку `B = 0`). Зразок задається в форматі двобайтового числа. У його довічним поданні одиниці відповідають світлим ділянкам лінії, нулі темним. Зразок штриховий лінії може виглядати як

`1111000011110000 = 0xF0F0`,

а пунктирною лінії як

`1010101010101010 = 0xAAAA`,

наприклад,

```
setlinestyle (SOLID_LINE, 0, NORM_WIDTH);
```

```
line (30, 30, 555, 400);
```

або

```
setlinestyle (USERBIT_LINE, 0x9898, THICK_WIDTH);
```

```
line (30, 30, 555, 400); //.
```

5.2.3 Малювання прямокутників

Прямокутник можна побудувати, вказавши координати його двох характерних точок, наприклад лівого верхнього (`x1, y1`) і правого нижнього (`x2, y2`) кутів.

Функція

```
void rectangle (int x1, int y1, int x2, int y2);
```

малює зафарбований прямокутник, а функція

```
void bar (int x1, int y1, int x2, int y2);
```

– зафарбований прямокутник.

Функція

```
void bar3d (int x1, int y1,
int x2, int y2, int d, int b);
```

викреслює тривимірний зафарбований прямокутник (паралелепіпед). Параметр *d* визначає глибину тривимірного контуру, а параметр *b* закрита верхня "кришка" паралелепіпеда.

5.2.4 Малювання кіл і еліпсів

Побудова еліптичних дуг здійснює функція

```
void ellipse (int x, int y, int a, int b, int Rx,
int Ry);
```

де *x*, *y* координати центру еліпса, *a*, *b* початковий і кінцевий кути, відраховані від горизонталі, *Rx*, *Ry* радіуси в горизонтальному і вертикальному напрямках. Для побудови замкнутого еліпса початковий і кінцевий кути встановлюють рівними 0 і 360 градусів. Окружність є окремим випадком еліпса при однакових горизонтальному і вертикальному радіусах. Так, наприклад, оператор `ellipse (100, 100, 0, 360, 50, 50);` малює в графічному вікні окружність.

Функція

```
void fillellipse (int x, int y, int Rx, int Ry);
```

малює зафарбований еліпс.

Стиль і колір заливки призначаються функцією

```
void setfillstyle (int style, int color);
```

де *style* стиль заливки, *color* колір заливки. Можливі значення стилю заливки наведені в табл. 5.6.

Таблиця 5.6 – Стили заливання

Назва	Номер	Опис
EMPTY_FILL	0	суцільна кольором фону
SOLID_FILL	1	суцільна
линіями		
LINE_FILL	2	горизонтальними
LTSLASH_FILL	3	похилими
SLASH_FILL	4	товстими похилими
BKSLASH_FILL	5	товстими похилими
LTBKSLASH_FILL	6	похилими
клеткой		
HATCH_FILL	7	тонкими лініями
XHATCH_FILL	8	товстими лініями
INTERLEAVE_FILL	9	діагональної штрихуванням
WIDE_DOT_FILL	10	рідкісними точками
CLOSE_DOT_FILL	11	частими точками
USER_FILL	12	по заданій масці

Функція

```
void pieslice(int x, int y, int a, int b, int R);
```

використовується для побудови секторів кола з заливкою. Координати x, y центр окружності, сектор малюється від кута a до кута b.

Функція

```
void sector (int x, int y, int a, int b, int Rx, int Ry);
```

малює сектор еліпса з заливкою. Координати x, y - центр еліпса, a b - початковий і кінцевий кути сектора, Rx, Ry радіуси еліпса в горизонтальному і вертикальному напрямках.

5.2.5 Виведення тексту в графічному режимі

Для виведення на екран текстових написів в графічному режимі використовуються функції outtext () і outtextxy ().

Функція

```
void outtext (char * string);
```

виводить рядок тексту, починаючи з поточного положення покажчика. наприклад,

```
outtext ( "Для продовження натисніть Enter"); //.
```

Якщо необхідно вказати точку початку виведення, доцільно використовувати функцію

```
void outtextxy (int x, int y, char * string);
```

де x, y - координати точки початку виведення тексту. наприклад,

```
outtextxy (50,100, "Для продовження натисніть  
Enter"); //.
```

Слід зауважити, що для виведення на екран чисельних даних їх необхідно попередньо перевести в строковий вид. Це можна зробити за допомогою функції

```
sprintf (char * S, "Форматний рядок", ...);
```

яка аналогічна функції printf (), тільки висновок виробляється в рядок S, зазначену першим аргументом. наприклад,

```
x = 3.14159;
```

```
char * S;
```

```
sprintf (S, "% f", x)
```

```
outtextxy (20, 40, S); //.
```

Функція

```
void settextstyle (int font, int B, int size);
```

призначена для управління параметрами шрифту при виводі тексту в графічному режимі. Тут font номер шрифту (за замовчуванням DEFAULT_FONT = 0), b напрямок (горизонтальне b = 0 або вертикальне b = 1), size розмір виведених символів: при size = 1 розмір букви 8x8 точок, при size = 2 розмір букви 16x16 точок. Віз-можна шрифти, які підтримуються бібліотекою, і їх номери перераховані в табл. 5.7.

Таблиця 5.7 – номери шрифтів

Назв	Номер
DEFAULT_FONT	0
TRIPLEX_FONT	1
SMALL_FONT	2
SANS_SERIF_FONT	3
GOTHIC_FONT	4
SCRIPT_FONT	5
SIMPLEX_FONT	6
TRIPLEX_SCR_FONT	7
COMPLEX_FONT	8
EUROPEAN_FONT	9
BOLD_FONT	10

Очищення графічного вікна і установка покажчика в точку з координатами (0,0) здійснює процедура

```
void cleardevice (void); //.
```

5.3 Побудова графіків функцій

Як приклад напишемо програму, яка малює в графічному вікні графік функції $f(x) = x \sin(x)$.

Включимо в програму заголовки бібліотек

```
#include <graphics.h>
```

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

Головна функція main () починається ініціалізацією графічного режиму

```
int gd, gm;
```

```
detectgraph (& gd, & gm);
```

```
initgraph (& gd, & gm, "");
```

призначається білий колір фону і виконується очистка екрану

```
setbkcolor (WHITE);
```

```
cleardevice (); //.
```

Будемо розрізняти екранну систему координат XOY з початком в лівому верхньому кутку вікна і систему координат xoy, щодо якої буде побудований графік. Центр системи xoy знаходиться в точці (X0,

Y0) щодо системи координат XOY. Вісь абсцис ох направимо вправо, вісь ординат оу вгору, а також виберемо розміри одиничних відрізків edx , edy . Тоді для будь-якої точки з екранними координатами (X, Y) можна обчислити її координати в системі хоу:

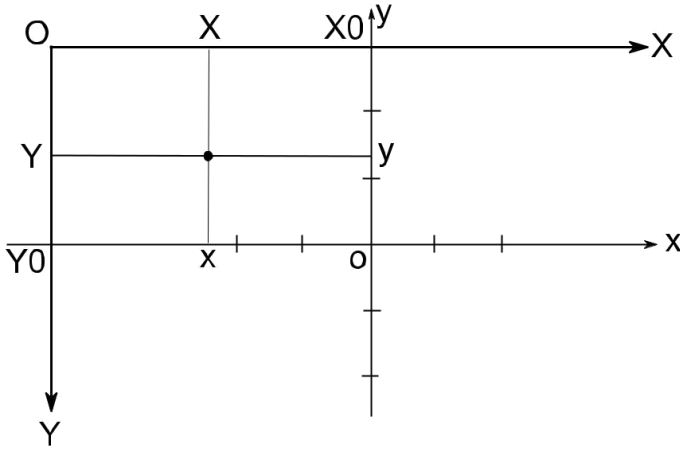


Рисунок 5.1 Перетворення екранних координат

$x = (X - X0) / edx$, $y = (Y - Y0) / edy$ (див. рис. 22). І навпаки, координати точок графіка (x, y) можна перерахувати в екранні координати:

$X = X0 + edx * x$, $Y = Y0 - edy * y$.

Далі необхідно намалювати осі координат, щодо яких буде побудований графік. Для цього вибираємо екранні координати (X0, Y0) центру нової системи координат посередині екрану

```
int X0 = getmaxx () / 2;
int Y0 = getmaxy () / 2;
```

і малюємо осі

```
line (0, Y0, getmaxx (), Y0); line (X0, 0, X0,
getmaxy ()); //.
```

Перпендикулярно осям маленькими лініями зобразимо зарубки з кроком, рівним одиничним відрізкам

```
int edx=20;
int edy=15;
// текущий цвет - зелёный setcolor(LIGHTBLUE);
// переменная цикла
```

```

int k;
//рисуем засечки
    // по оси x for(k=0;edx*k<X0;k++)
{
line(X0+k*edx,Y0-3,X0+k*edx,Y0+3); line(X0-
k*edx,Y0-3,X0-k*edx,Y0+3);
}

    //по оси y
for(k=0;edy*k<Y0;k++)
{
line(X0-3,Y0+k*edy,X0+3,Y0+k*edy);
line(X0-3,Y0-k*edy,X0+3,Y0-k*edy);
} //.
```

Далі приступаємо до малювання графіка. Ставив поточними рожевий колір і суцільний тип лінії товщиною 3 точки

```

setcolor (LIGHTRED);
setlinestyle (SOLID_LINE, 0, THICK_WIDTH); //.
```

Оголошуємо змінні дійсного типу: a , b кордону області побудови по осі ox , x , y поточні координати покажчика, δ крок побудови по осі ox

```

float a, b, x, y, delta;
// інтервал дискретизації
delta = 0.1;
// границы построения по оси x
a=-15; b=15; //.
Початкова точка побудови графіка має координати
(a, f (a)). Перемістимо в неї вказівник
(графічний курсор).
// Вычисление начальных координат
x=a;
```

```

y=x*sin(x);
// перемещение указателя в начало графика
moveto(X0+edx*x,Y0-edy*y); //.
```

Для побудови будемо обчислювати координати наступної точки графіка (x , $f(x)$) і малювати лінію між поточним становищем покажчика і знову обчисленої

точкою. Таким чином, графік є лама-ною лінією, наближену до реального графіку.

```
// цикл построения графика
for (x=a; x<=b; x=x+delta)
{
    y=x*sin(x);
    lineto(X0+edx*x, Y0-edy*y);
}; //.
```

На завершення програми в графічне вікно виводиться заголовний рядок.

```
// установка стиля текста
settextstyle(SANS_SERIF_FONT, 0, 2);
// вывод заголовка
outtextxy(180, 10, "График функции
f(x)=x*sin(x)");
```

Нижче наведено повний текст програми побудови графіка, а результат її роботи графічне вікно показано на рис. 23.

```
1.  #include <graphics.h>
2.  #include <math.h>
3.  int main()
4.  {
5.      // Инициализация графики int gd, gm;
      detectgraph(&gd, &gm); initgraph(&gd, &gm, "");
6.      // цвет фона - белый setbkcolor(WHITE);
7.      // очистка окна cleardevice();
8.      // текущий цвет - черный setcolor(BLACK);
9.      // координаты центра
10.
11.     int X0=getmaxx()/2;
12.     int Y0=getmaxy()/2;
13.     // оси координат
14.         line(0, Y0, getmaxx(), Y0);
15.         line(X0, 0, X0, getmaxy());
```

```

16. // единичные отрезки в пикселах int edx=20;
17. int edy=15;
18. // текущий цвет - зелёный
19.     setcolor(LIGHTBLUE);
20. // переменная цикла
21. int k;
22. // рисуем засечки
23. // по оси x for(k=0;edx*k<X0;k++)
24. {line(X0+k*edx,Y0-3,X0+k*edx,Y0+3);
    line(X0-k*edx,Y0-3,X0-k*edx,Y0+3);}
25. // по оси y for(k=0;edy*k<Y0;k++)
26. {line(X0-3,Y0+k*edy,X0+3,Y0+k*edy);
    line(X0-3,Y0-k*edy,X0+3,Y0-k*edy);}
27. // текущий цвет - розовый setcol-
    or(LIGHTRED);
28. // стиль линии - сплошная толстая setlin-
    estyle(SOLID_LINE,0,THICK_WIDTH);
29. // описание переменных
30. float a,b,x,y,delta;
31. // интервал дискретизации
32. delta=0.1;
33. // границы построения по оси x
34. a=-15; b=15;
35. // Вычисление начальных координат x=a;
36. y=x*sin(x);
37. // перемещение указателя в начало графика
    moveto(X0+edx*x,Y0-edy*y);
38. // цикл построения графика
    for(x=a;x<=b;x=x+delta)
39. {y=x*sin(x); lineto(X0+edx*x,Y0-edy*y);};
40. // установка стиля текста
    settextstyle(SANS_SERIF_FONT,0,2);
41. // вывод заголовка

```

```

42. outtextxy(180,10,"График функции
    f(x)=x*sin(x)"); getch(); // ожидание
43. // закрытие графического окна closegraph();
44. return 0;
45. } //.
```

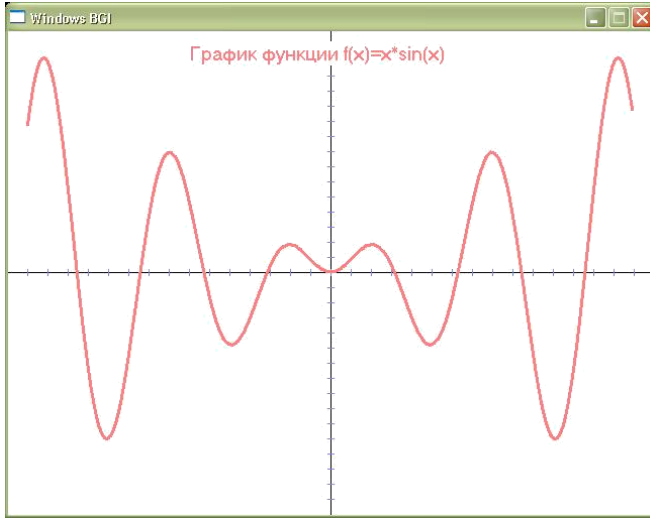


Рисунок 5.2 Графічне вікно

5.4 Практичне завдання

Дана $n + 1$ пара значень $(x; y)$, пов'язаних функціональною залежністю $f(x)$ вашого варіанту. Значення x обрані в $n + 1$ рівновіддєлєних точках відрізка $[a; b]$. Необхідно побудувати інтерполяційний многочлен $P_n(x)$. Для цього:

- 1) Розрахуйте значення функції $f(x)$ в рівновіддєлєних точках відрізка $[a; b]$. Тим самим будуть заповнені два масиви x і y , які можна інтерпретувати як експериментально виміряні значення яких-небудь фізичних величин.
- 2) Розрахуйте розділені різниці за значеннями функції в $n + 1$ вузлі.
- 3) Запрограмуйте обчислення інтерполяційного многочлена $P_n(x)$ за схемою Горнера.

- 4) Побудуйте в графічному вікні графіки вихідної функції $f(x)$ і інтерполяційного многочлена $P_n(x)$ на відрізку $[a; b]$. Значення > 0 Будь ласка, виберіть таким, щоб область побудови графіка по осі x була в 1: 5 2 рази більше відрізка, на якому обрали вузли дискретизації.
- 5) Дослідить поведінку інтерполяційного многочлена при різних кількості вузлів дискретизації (4,5,7,10 точок).
- 6) Змодельуйте похибка завдання значень функції у вузлах дискретизації шляхом додавання до значень функції випадкових чисел.

№	$f(x)$	$[a,b]$	№	$f(x)$	$[a,b]$
1	e^x	$[-1,1]$	9	$1/(1-x^2)$	$[2,5]$
2	e^{-x}	$[-2,2]$	10	$x^2 \ln(x)$	$[1,5]$
3	$\sin(x)$	$[-\pi/2, \pi/2]$	11	$\ln(x)$	$[-2,2]$
4	$\cos(x)$	$[0, 4\pi]$	12	$ \sin(x) $	$[-\pi/2, \pi/2]$
5	$\operatorname{tg}(x)$	$[-\pi/4, \pi/4]$	13	$\ln(x^2+1)$	$[-2,2]$
6	$1/x$	$[1,3]$	14	$e^{- x }$	$[-1,1]$
7	$1/x^2$	$[1,3]$	15	$\sqrt{x^3}$	$[0,3]$
8	$\sqrt{x}$	$[0,5]$	16	$\cos(x) - \sin(x)$	$[-4,4]$

6 ЛАБОРАТОРНА РОБОТА № 6 «ЧИСЕЛЬНЕ ІНТЕГРУВАННЯ»

Мета роботи: ознайомлення з методами чисельного інтегрування; набуття практичних навичок побудови призначеного для користувача інтерфейсу засобами бібліотеки Qt.

6.1 Основні методи чисельного інтегрування

У практичній роботі радіоінженера часто виникає необхідність обчислення інтегралів, які або важко, або неможливо обчислити аналітично.

Визначений інтеграл

$$I = \int_a^b f(x)dx \quad (6.1)$$

можна трактувати як площу фігури під графіком підінтегральної функції, тобто як площа криволінійної трапеції. Багато методів чисельного інтегрування засновані на розбитті інтервалу інтегрування на більш дрібні відрізки, на яких можна застосувати прості формули для площі маленької трапеції. Потім площі маленьких трапецій складаються.

Розіб'ємо, наприклад, відрізок інтегрування на N рівних відрізків дискретизації довжиною $h = (b - a) / N$ точками

$a = x_0, x_1, \dots, x_N = b$. Якщо відрізки дискретизації досить малі, то на кожному відрізку площа трапеції можна приблизно замінити площею прямокутника. Прямокутники можна будувати, спираючись на ліву чи праву сторону відрізка (див. рис. 6.1). Звідси отримуємо метод лівих або правих прямокутників. Ясно, що зі збільшенням числа відрізків, сума площ прямокутників все з більшою точністю буде збігатися з площею криволінійної трапеції.

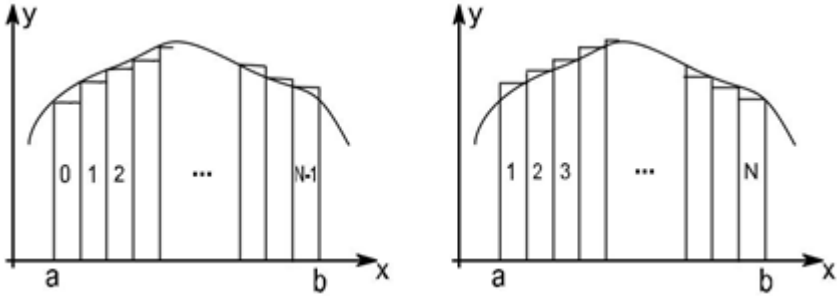


Рисунок 6.1 Ілюстрація методів прямокутників

Площа маленького прямокутника залежить від його номера k і дорівнює $S_k = hf(x_k) = hf(a + hk)$. Тоді для наближеного обчислення інтеграла справедливі формули лівих прямокутників

$$I \approx \sum_{k=0}^{N-1} S_k = \frac{b-a}{N} \sum_{k=0}^{N-1} f(a + hk) \quad (6.2)$$

і правих прямокутників

$$I \approx \sum_{k=0}^N S_k = \frac{b-a}{N} \sum_{k=0}^N f(a + hk) \quad (6.3)$$

Якщо на кожному відрізку площа криволінійної трапеції наближено замінити площею прямолінійної трапеції (див рис. 6.2), то можна обчислити інтеграл точніше

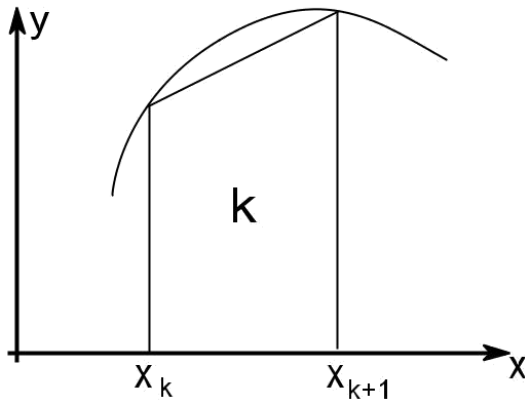


Рисунок 6.2 Ілюстрація методу трапецій

Площа прямокутної трапеції з номером k дорівнює
 $S_k = h [f(x_k) + f(x_{k+1})] / 2 = h [f(a + hk) + f(a + h(k + 1))] / 2$;
 а формула трапецій для обчислення інтеграла має вигляд

$$I \approx \sum_{k=0}^{N-1} S_k = \frac{b-a}{2N} \sum_{k=0}^{N-1} [f(x_k) + f(x_{k+1})] = \\ = \frac{b-a}{N} \left[\sum_{k=1}^{N-1} f(a + kh) + \frac{f(a) + f(b)}{2} \right]. \quad (6.4)$$

Вирази наближеного обчислення певних інтегралів (6.2) - (6.4) називають квадратурними формулами. Математично методика їх отримання як правило зводиться до розбиття інтервалу інтегрування на багато дрібних під інтегралів та подання інтеграла (6.1) у вигляді

$$I = \int_a^b f(x) dx = \sum_{k=0}^{N-1} \int_{x_k}^{x_{k+1}} f(x) dx. \quad (6.5)$$

Далі під інтегральна функція на кожному під інтервалі апроксимується деякої іншої функцією $g_k(x)$, інтеграл від якої легко обчислюється аналітично. Якщо апроксимувати функцію $f(x)$ константою $g_k(x) = f(x_k)$ або $g_k(x) = f(x_{k+1})$, то отримуємо формули прямокутників (6.2) і (6.3). Якщо апроксимувати функцію прямого, проведеної через точки $(x_k, f(x_k))$ і $(x_{k+1}, f(x_{k+1}))$, то отримуємо формулу трапецій (6.4). У загальному вигляді квадратурної формули можна записати у вигляді

$$I \approx \sum_{k=0}^N A_k f(x_k), \quad (6.6)$$

де коефіцієнти A_k визначаються способом апроксимації під інтегральної функції. Перерахуємо деякі квадратурні формули, засновані на (6.6).

Формула Сімпсона (формула парабол)

$$I \approx \frac{b-a}{3N} \left[4 \sum_{k=1}^{N/2} f(x_{2k-1}) + 2 \sum_{k=1}^{N/2-1} f(x_{2k}) + f(a) + f(b) \right]. \quad (6.7)$$

Значення N передбачається парним.

Формула Гаусса

$$I \approx \frac{b-a}{2} \sum_{k=1}^N H_k f(x_k), \quad x_k = \frac{a+b}{2} + \frac{b-a}{2} t_k \quad (6.8)$$

заснована на виборі нерівновіддаленими вузлами дискретизації t_k , що збігаються з нулями поліномів Лежандра порядку N на стандартному інтервалі $[1; 1]$. Значення вузлів t_k і ваг H_k для різних N наведені в табл. 6.1.

Таблиця 6.1 – Вузли та ваги квадратурної формули Гаусса

$N = 1$	$t_1 = 0,5$	$H_1 = 2$
$N = 2$	$-t_1 = t_2 = 0.577350269$	$H_1 = H_2 = 2$
$N = 3$	$-t_1 = t_3 = 0.774596669$ $t_2 = 0$	$H_1 = H_3 = 0.555555555$ $H_2 = 0.888888888$
$N = 4$	$-t_1 = t_4 = 0.861136311$ $-t_3 = t_2 = 0.339981043$	$H_1 = H_4 = 0.347854845$ $H_2 = H_3 = 0.652145155$
$N = 5$	$-t_1 = t_5 = 0.906179846$ $-t_2 = t_4 = 0.538469310$ $t_3 = 0$	$H_1 = H_5 = 0.236926885$ $H_2 = H_4 = 0.478628670$ $H_3 = 0.568888888$

6.2 Практичне завдання

Обчисліть чисельно усіма розглянутими методами інтеграл, заданий у Вашому варіанті. Порівняйте отриманий результат з знайденим аналітично. Передбачте графічний інтерфейс користувача.

6.2.1 Варіанти завдань

1. $\int_{-2}^2 (1 - 2 x) dx.$	2. $\int_{-2}^2 \cos(2x) dx.$
3. $\int_0^5 \sqrt{x} dx.$	4. $\int_1^4 \ln(x) dx.$
5. $\int_{-3}^3 \exp(-x^2) dx.$	6. $\int_0^{\pi} \cos^2(x) dx.$
7. $\int_0^4 \ln(x^2 + 1) dx.$	8. $\int_1^8 (x^2 + x) dx$

9. $\int_1^5 \ln(x) dx.$	10. $\int_{-2}^2 (x^3 + x^2 + 1) dx.$
11. $\int_{-4}^1 e^x dx$	12. $\int_0^{10} x \exp(-x^2) dx.$
13. $\int_0^{\pi} \sin(x) dx.$	14. $\int_{-10}^0 x^3 \exp(-x^2) dx.$
15. $\int_{-\pi/2}^{\pi/2} \cos(x) dx.$	16. $\int_{-5}^5 (x^2 + x) dx.$

7 ЛАБОРАТОРНА РОБОТА № 7 «НАБЛИЖЕНЕ РІШЕННЯ НЕЛІНІЙНИХ РІВНЯНЬ»

Мета роботи: ознайомлення з методами чисельного рішення нелінійних рівнянь; закріплення практичних навичок побудови призначеного для користувача інтерфейсу засобами бібліотеки Qt.

7.1 Основні методи чисельного рішення рівнянь

У практичній роботі радіоінженера часто виникає необхідність вирішення нелінійних рівнянь, які або важко, або неможливо вирішити аналітично. Рішення таких рівнянь можна покласти на ЕОМ. Тому необхідно мати уявлення про методи вирішення рівнянь на ЕОМ. Будь-яке рівняння можна представити у вигляді

$$f(x) = 0;$$

де $f(x)$ деяка нелінійна функція. Так, наприклад, для рівняння $x^3 - 5x^2 + 2x = -5$ маємо

$$f(x) = x^3 - 5x^2 + 2x + 5;$$

Значення x при якому справедливо рівність $f(x) = 0$ називається коренем рівняння. Графічно корені рівняння є абсциси точок перетину графіка функції $f(x)$ з віссю ОХ, тобто корені рівняння це нулі функції $f(x)$.

Процес відшукування коренів ділитися на два етапи:

1. Відділення коренів, тобто визначення відрізка містить один корінь.

2. Уточнення кореня з заданою точністю.

Для першого етапу немає загальних методів рішення, відрізки визначаються або табуляцією, або виходячи з фізичного змісту, або аналітичними методами. Іноді виявити відрізки, на яких знаходяться корені, можна попередніми дослідженням функції $f(x)$. Ознакою того, що на відріжку $[a; b]$ знаходиться корінь рівняння є нерівність $f(a) f(b) < 0$. У прикладі кубічного рівняння функція $f(x) = x^3 - 5x^2 + 2x + 5$ являє собою кубічну параболу, яка може мати один максимум і один мінімум. Для того, щоб знайти положення максимуму і мінімуму прирівнюємо нулю похідну функції $f(x)$:

$$3x^2 - 10x + 2 = 0;$$

Вирішимо отримане рівняння: $x_1 \approx 3.12, x_2 \approx 0.21$. Оскільки $f(0;21) > 0$, $f(3;12) < 0$, $f(a) f(b) < 0$, то на відрізку $[0;21; 3;12]$ знаходиться хоча б один корінь. Із загальної виду функції $f(x)$ ясно, що знайдений відрізок містить тільки один корінь. Для двох інших коренів відрізки доведеться підібрати. Скористаємося тим, що при $x \in (-\infty; 0.21)$ і $x \in (3.12; +\infty)$ функція $f(x)$ зростає. Отже, якщо взяти досить маленький x , наприклад, $x = -2$, то $f(-2) < 0$ і відрізок $[-2; 0;21]$ містить корінь. Аналогічно підберемо досить великий $x = 5$, так що $f(5) > 0$. Тоді відрізок $[3;12; 5]$ також містить корінь.

При інших функціях $f(x)$ подібне міркування, можливо, провести не вдасться. Тоді можна вибрати довільні досить маленькі a і b , і поступово зміщувати їх вправо, поки не буде знайдено потрібний відрізок. Критерієм придатності відрізка $[a; b]$ є нерівність $f(a) f(b) < 0$.

Другий етап уточнення кореня виконується різними ітераційним методами, суть яких в тому, що будується числова послідовність x_k сходяється до кореня. Виходом з ітераційного процесу є одне з двох умов: $|f(x_k)| < \varepsilon$ або $|x_k - x_{k-1}| < \varepsilon$. Таким чином після знаходження відрізків локалізації коренів необхідно звужувати ці відрізки до тих пір, поки або їх довжина не стане менше заздалегідь заданої величини, яку будемо називати точністю, наприклад $\varepsilon = 10^{-6}$, яке значення функції $f(x)$ не наблизиться до нуля з точністю ε . Розглянемо найбільш вживані на практиці методи: дихотомії (ділення відрізка навпіл), ітерацій і дотичних.

7.1.1 Метод половинного ділення

Дана монотонна, безперервна функція $f(x)$, яка містить корінь на відрізку $[a; b]$, де $b > a$. Визначити корінь з точністю ε , якщо відомо, що $f(a) f(b) < 0$.

Процес зменшення довжини відрізка можна зробити наступним чином:

- а) знаходимо середину відрізка $d = (a + b) / 2$,
- б) знаходимо значення функції $f(a)$, $f(b)$, $f(d)$,
- в) якщо $f(a) f(d) < 0$, то корінь знаходиться в лівій половині відрізка $[a; b]$. Переносимо праву межу відрізка в його середину $b = d$;

інакше переносимо ліву межу відрізка в його середину $a = d$ (див. рис. 7.1).

г) Якщо довжина $|b - a| > \varepsilon$, то переходимо до пункту 1.

Як тільки довжина відрізка стає менше необхідної точності $|b - a| < \varepsilon$, можемо коренем рівняння вважати будь-яку точку відрізка, наприклад, його середину.

7.1.2 Метод хорд

Процес зменшення довжини відрізка відбувається швидше, якщо ділити його не навпіл, а використовуючи хорду. Для цього:

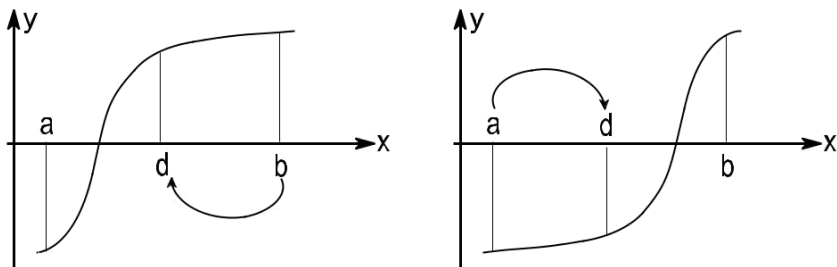


Рисунок 7.1 Ілюстрація методу половинного ділення

а) В думках проводимо пряму між точками $(a; f(a))$ і $(b; f(b))$.

Ця пряма перетинає вісь X в точці $d = a - (b - a) f(a) / [f(b) - f(a)]$.

Цією точкою відрізок $[a; b]$ розділяється на 2 частини,

б) знаходимо значення функції $f(a)$, $f(b)$, $f(d)$,

в) якщо $f(a) f(d) < 0$, то корінь знаходиться в лівій частині від-
реставрування $[a; b]$. Переносимо праву межу відрізка в точку d ($b = d$); інакше переносимо ліву межу відрізка в точку d ($a = d$).

г) Якщо довжина $|b - a| > \varepsilon$, то переходимо до пункту 1.

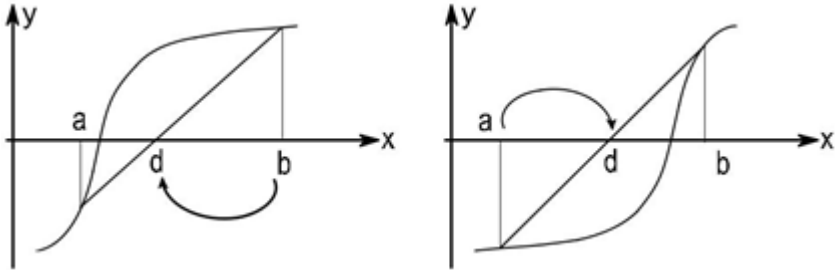


Рисунок 7.2 Ілюстрація методу хорд

При досягненні довжини відрізка менше за необхідну точності $|b - a| < \varepsilon$, вибираємо в якості кореня рівняння будь-яку точку відрізка, наприклад, його середину.

7.1.3 Метод ітерацій

Замінімо рівняння $f(x) = 0$ рівносильним рівнянням

$$x = \varphi(x). \quad (13)$$

Графічно корені рівняння (13) є абсиси точок перетину графіків функцій $y = \varphi(x)$ і $y = x$. Виберемо наближене значення кореня x_0 , що належить відрітку $[a; b]$, підставимо його в праву частину рівняння (13), отримаємо нове значення

$$x_1 = \varphi(x_0). \quad (14)$$

Далі підставимо x_1 в праву частину рівняння (13) отримаємо:

$$x_2 = \varphi(x_1). \quad (15)$$

І так далі на кроці з номером n маємо

$$x_n = \varphi(x_{n-1}). \quad (16)$$

Якщо послідовність x_n є збіжною при $n \rightarrow \infty$, то даний алгоритм дозволяє визначити шуканий корінь.

7.1.4 Метод дотичних (Ньютона)

1) Вибираємо грубе наближення кореня x_0 (або точку a , або b).

2) Знаходимо значення функції в точці x_0 і проводимо дотичну до графіка $f(x)$. Точку перетину дотичної з віссю абсцис $x_1 = x_0 - f(x_0) / f'(x_0)$ вважаємо таким наближенням кореня.

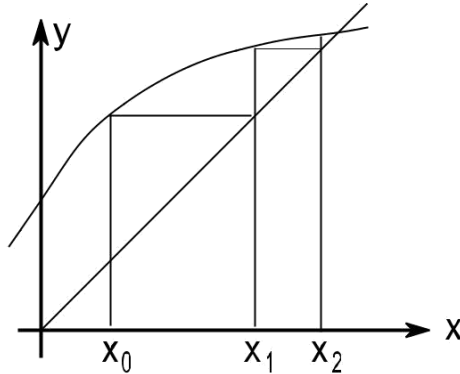


Рисунок 7.3 Ілюстрація методу ітерацій

3) Визначаємо значення функції в точці x_1 , через цю точку знову проводимо дотичну отримаємо точку $x_2 = x_1 - f(x_1) / f'(x_1)$.

4) На кроці з номером n : $x_n = x_{n-1} - f(x_{n-1}) / f'(x_{n-1})$. Якщо послідовність x_n є збіжною при $n \rightarrow \infty$, то даний алгоритм дозволяє приблизно визначити шуканий корінь.

Як приклад наведемо програму, яка знаходить коріння рівняння $x^3 - 5x^2 + 2x = -5$ методом половинного ділення.

```

1.  #include <iostream.h>
2.  #include <math.h>
3.  // Функція, що описує рівняння
4.  float f (float x)
5.  {return x * x * x-5 * x * x + 2 * x + 5;}
6.  float eps = 1e-6; // Точність
7.  float a, b, d, res;
8.  int main()
9.  {
10. // Вводим с клавиатуры границы отрезка
11. // локализации корня

```

```

12. cout << "Введите границы отрезка: ";
13. cin >> a >> b;
14. if (f(a)*f(b)>0)
15. {cout << "Отрезок задан неверно!";}
16. else
17. {
18.     do // пока длина отрезка больше точности
19.     {
20.         d=(a+b)/2; // находим середину
21.         // выбираем половину отрезка с корнем
22.         if (f(a)*f(d)<0) {b=d;} else {a=d;}
23.     }
24.     while (fabs(b-a)>eps);
25.     // находим результат
26.     res=(b+a)/2;
27.     cout << "Найдено решение " << res << endl;
28.     cout << "Проверка" << f(res) << endl;
29. }
30. return 0;
31. } // .

```

Якщо рівняння настільки складне, то знайти відрізок локалізації не вдається, то можна передбачити в програмі пошук відрізка, наприклад, так:

```

a=-6;    b=a+0.1;
while (f(a)*f(b)>0)
{
a=b;
b=b+0.1;
} // .

```

7.2 Практичне завдання

Складіть програму, яка будь-якими двома з розглянутих методів знаходить всі корінні рівняння. Передбачте графічний інтерфейс користувача.

7.2.1 Варіанти завдань

1. $x^3 - x = 2x^2 - 2$	2. $x^3 + 12 = x(3x - 4)$
3. $x^3 = 4x(x - 1)$	4. $x^3 - x^2 - 8x = -12$
5. $-x^2 = \ln(x)$	6. $\ln(x) + 2 = x^3$
7. $x^2 = 2\sin(x) + 1$	8. $x^2 = 2\cos(2x) + 1$
9. $x^3 - 3x^2 - 5x = -12$	10. $x^3 = 2\sin(3x) + 1$
11. $\ln(x^2 + 1) = 1$	12. $\ln(x^4 + 1) = 2$
13. $\arctg(2x^2 - 2) = 1$	14. $2x^3 = 4x(x + 1)$
15. $x^4 + x = 2$	16. $x^4 + 5x = 3$
17. $x^4 + 5x = 3 - \ln(x)$	18. $x^2 + \sin(x) = 2 + x$
19. $x^2 + \sin(x) = 2 + x$	20. $x^2 + \sin(x) = \lg(x) + x$
21. $x^2 + \cos(x) = 7 + 4x$	22. $x^2 + \cos(x) = 7 - 4x$
23. $x^4 + \cos(x) = 5$	24. $3x^3 + 2x^2 = 5x + 2$
25. $3x^2 + 2x^2 = 3x + 1$	26. $x^3 - 10\sqrt{x} = -5$

Додаток А
Приклад оформлення титульної сторінки

Міністерство освіти і науки України
Національний університет «Запорізька політехніка»

Кафедра МіНЕ

ЛАБОРАТОРНА РОБОТА

з дисципліни
„ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ“

(тема роботи)

Виконав:

студент ____ курсу, група РТ-____

(Ініціали, Прізвище)

Прийняв:

(посада)

(Ініціали, Прізвище)

20 __ р.

Додаток Б

«Створення графічного інтерфейсу користувача засобами бібліотеки Qt»

Переважна більшість сучасні програмних продуктів мають так званим графічним інтерфейсом користувача (Graphics User Interface, GUI). У свою чергу сучасні операційні системи надають можливості створення GUI. Так, наприклад, Windows має у своєму складі бібліотеку API (Application Programming Interface) функцій. Однак, їх використання вимагає великого практичного досвіду і глибоких знань особливостей роботи операційної системи. Для полегшення написання програм створені інші бібліотеки більш доступних інструментів: MCF (Microsoft Foundation Classes), VCL (Visual Component Library) фірма Borland та ін.

Бібліотека Qt являє собою набір інструментальних засобів для побудови графічного інтерфейсу користувача і має як мінімум дві переваги в порівнянні з іншими бібліотеками. По-перше вона є кросплатформенною. Це означає, що програми на базі Qt виглядають однаково на різних операційних системах. По-друге існує версія Qt, яка поширюється вільно. У даній лабораторній роботі будуть описані лише деякі можливості бібліотеки Qt, оскільки для повного її опису знадобиться не одна така книга. Бажаючим більш глибоко вивчити Qt можна порадити книги [17, 18, 19].

Бібліотека Qt побудована у вигляді бібліотеки класів. Тому перед її вивченням доцільно також детально ознайомитися з принципами об'єктно орієнтованого програмування [6, 7, 8, 9].

Створення головного вікна програми

Програми з графічним інтерфейсом користувача складаються як мінімум з одного головного вікна, яке з'являється при запуску програми. В ході роботи можливе відкриття і закриття інших вікон. При закритті головного вікна завершується робота всієї програми. Основною метою використання графічного інтерфейсу є взаємодія з користувачем. Тому програма повинна виконувати так званий "цикл обробки подій". Події відбуваються від дій користувача: переміщень і клацань мишею, натискання клавіш на клавіатурі і т.д. Інформація про подію, що відбулася передається програмі, яка може містити підпрограму обробки цієї події.

Для реалізації перерахованих властивостей в бібліотеці Qt реалізований клас `QApplication`. Переважна більшість програм на Qt на самому початку створюють екземпляр цього класу і використовують його метод `exec ()` для запуску циклу обробки подій. Тому зазвичай використовується наступна структура Qt-програми:

```
#include <QApplication>

...
int main (int argc, char * argv [])
{
    QApplication app (argc, argv);
    ...
    return app.exec ();
} //.
```

Директива `#include <QApplication>` включає в текст програми заголовки `QApplication.h`. При використанні ще яких-небудь класів необхідно включати в програму заголовки кожного з них або скористатися більш загальним файлом `QtGui.h`. Аргументи функції `main` служать для взаємодії програми з операційною системою. Операційна система дозволяє запускати програму різними способами, наприклад, з командного рядка або за допомогою ярлика. При цьому після імені програми можуть бути вказані аргумент, кількість яких заздалегідь невідомо. Всі аргументи передаються програмі у вигляді масиву рядків `argv`, а їх кількість у вигляді цілої змінної `argc`. Подальша доля цих змінних залежить від самої програми. Рядок `QApplication app (argc, argv);` створює екземпляр `app` класу `QApplication`. В якості аргументів конструктору класу передається масив рядків `argv` і розмір цього масиву `argc`. Це обставина дає можливість управляти зовнішнім виглядом додатки. При запуску Qt-програми може бути вказаний параметр `-style = Ім'яСтиля`. Як ім'я стилю допускається використовувати `Windows`, `CDE`, `Motif`, `Plastique` або `Cleanlooks` в будь-якій операційній системі. Функція `exec ()` являє собою метод класу `QApplication` і реалізує цикл обробки подій. Вона повертає цілий результат код завершення програми, що передається операційній системі. При нормальному завершенні програми код дорівнює нулю.

Компіляція Qt програми

Компіляція Qt програми має деякі особливості. Як правило програма складається з декількох файлів, розташованих в одному катало-

зі, який будемо називати каталогом проекту. Досить просто виконувати компіляцію з допомогою командного рядка. Для цього необхідно:

- 1) У командному рядку зробити поточним каталог проекту. Бібліотека Qt при установці створює ярлик "Qt Command Prompt" для запуску командного рядка з уже налаштованими параметрами компіляції. Можна також користуватися командним рядком оболонки Far manager або консоллю Linux.
- 2) Виконати команду `qmake -project`. При цьому буде створено файл проекту з розширенням `pro`.
- 3) Виконати команду `qmake`. В результаті будуть створено `Makefile`.
- 4) Виконати команду `make` для компіляції.

Описаний спосіб компіляції є самим універсальним. Можна також компілювати програми за допомогою інтегрованих середовищ. Найзручнішою є IDE Kdevelop операційної системи Linux. Інтегровану середу Code :: Blocks можна використовувати тільки разом з плагіном QtWorkBench. Але зусилля, витрачені на установку і настройку плагіна можна порівняти з компіляцією за допомогою командного рядка. Тому питання використання Qt спільно з інтегрованими середовищами програмування не будемо розглядати в даній книзі.

Віджети

Візуальні елементи керування програми, (кнопки, рядки введення, текстові написи і т.д.) називаються віджетами. Кожному віджету відповідає в бібліотеці Qt свій клас. Наприклад, для відображення текстових написів використовується клас `QLabel` (мітка). Кожен віджет по суті являє собою вікно, тому головне вікно програми можна будувати на основі якого-небудь елемента. Створимо в програмі мітку екземпляр класу `QLabel`:

```
QLabel Lab ("Hello !!!");
```

змінимо її розмір за допомогою методу `resize` і покажемо мітку методом `show`

```
Lab.resize (50,30);
```

```
Lab.show (); //
```

В результаті отримуємо наступний текст програми

```
1.  #include <QtGui>
2.  int main (int argc, char * argv [])
3.  {
```

```

4.  QApplication app (argc, argv);
5.  QLabel Lab ("Hello !!!");
6.  Lab.resize (50,30);
7.  Lab.show ();
8.  return app.exec ();
9.  } //.

```

Аналогічно можна створити додаток на основі інших віджетів. У наступному прикладі програма створюється на основі віджета класу QPushButton, який являє собою кнопку.

```

1.  #include <QtGui>
2.  int main (int argc, char * argv [])
3.  {
4.  QApplication app (argc, argv);
5.  QPushButton * Btn = new QPushButton ("But-
   ton");
6.  Btn-> resize (50,30);
7.  Btn-> show ();
8.  return app.exec ();
9.  } //.

```

Тут екземпляр класу QPushButton створюється динамічно за допомогою операції new. В результаті робота з новоствореним об'єктом "кнопка" виконується через покажчик Btn. Класи всіх віджетів є нащадками класу QWidget. Примірники цього класу можна використовувати в якості "підкладки" головного вікна програми. Створимо вікно на основі об'єкта класу QWidget

```

1.  #include <QtGui>
2.  int main (int argc, char * argv [])
3.  {
4.  QApplication app (argc, argv);
5.  QWidget * wdt = new QWidget ();
6.  wdt-> resize (300,300);
7.  wdt-> setWindowTitle ( "!!! Window !!!");
8.  wdt-> show ();

```

```

9.   return app.exec ();
10. } //.

```

В результаті інтерфейс програми представляє собою порожній вікно розміром 300 на 300 точок і заголовком "!!! Winwow !!!". На цьому порожньому вікні можна розміщувати інші елементи управління. Розмістимо, наприклад, дві кнопки. Для цього динамічно створимо об'єкти класу QPushButton

```

QPushButton * btn1 = new QPushButton ( "Button1");
QPushButton * btn2 = new QPushButton ( "Button2");

```

а віджет wdt призначимо батьком створених кнопок

```

btn1-> setParent (wdt);
btn2-> setParent (wdt); //.

```

Далі перемістимо кнопки в зазначені координати щодо батьківського віджета і покажемо головне вікно

```

btn1-> move (10,10);
btn2-> move (100,10);
wdt-> show (); //.

```

Таким чином, отримаємо наступний текст програми:

```

1.   #include <QtGui>
2.   int main (int argc, char * argv [])
3.   {
4.     QApplication app (argc, argv);
5.     QWidget * wdt = new QWidget ();
6.     wdt-> resize (300,300);
7.     wdt-> setWindowTitle ( "!!! Window !!!");
8.     QPushButton * btn1 = new QPushButton (
        "Button1");
9.     QPushButton * btn2 = new QPushButton (
        "Button2");
10.  btn1-> setParent (wdt);
11.  btn2-> setParent (wdt);
12.  btn1-> move (10,10);

```

```

13. btn2-> move (100,10);
14. wdt-> show ();
15. return app.exec ();
16. } // .

```

Зовнішній вигляд головного вікна показаний на рис. 24. Однак, таке застосування не володіє ніякою функціональністю, оскільки не передбачені дії, що виконуються при натисканні кнопок.

Автоматичне розміщення віджетів

Ручне розміщення віджетов всередині головного вікна, що застосовується в попередніх прикладах, володіє недоліком: при зміні розмірів вікна розміщення віджетів не підлаштовується під нові розміри.



Рисунок Б.1 – Зовнішній вигляд вікна з двома кнопками

Позбутися від цього недоліка дозволяє застосування так званих менеджерів компоновання. Вони являють собою об'єкти-контейнери, автоматично мають в своєму розпорядженні одні віджети на поверхні іншого віджета.

Бібліотека Qt надає три основні класи менеджерів компоновання: `QHBoxLayout` (для горизонтального розміщення), `QVBoxLayout`

(для вертикального розміщення) `QGridLayout` (для табличного розміщення). Продemonструємо їх застосування на прикладі.

Нехай необхідно розмістити в головному вікні дві кнопки, два текстових рядка введення і одну мітку. Спочатку окремо створюємо такі незалежні один від одного об'єкти: додаток,

```
QApplication app (argc, argv);
```

віджет головного вікна

```
QWidget * wdt = new QWidget ();
```

дві кнопки

```
QPushButton * btn1 = new QPushButton ( "Button1");
```

```
QPushButton * btn2 = new QPushButton (
    "Button2");
```

два рядки введення

```
QLineEdit * edt1 = new QLineEdit ( "Text1");
```

```
QLineEdit * edt2 = new QLineEdit ( "Text2");
```

мітка

```
QLabel * lb = new QLabel ( "Text Label");
```

два горизонтальних менеджера компоновки

```
QHBoxLayout * hlayout1 = new QHBoxLayout; QHBoxLayout * hlayout2 = new QHBoxLayout;
```

один вертикальний менеджер компоновування

```
QVBoxLayout * vlayout = new QVBoxLayout; //.
```

Потім призначаємо розміри головного вікна, його заголовок і спосіб вирівнювання тексту мітки:

```
wdt->resize (300,150);
```

```
wdt->setWindowTitle ( "!!! Window !!!");
```

```
lb->setAlignment (Qt :: AlignCenter);
```

І, нарешті, вводимо в дію менеджери компоновки. Для цього рядка введення `edt1` і `edt2` додаємо за допомогою методу `addWidget ()` до першого горизонтального менеджера компоновування `hlayout1`, а кнопки `btn1` і `btn2` до другого `hlayout2`

```
hlayout1->addWidget (edt1);
```

```
hlayout1->addWidget (edt2);
```

```
hlayout2->addWidget (btn1);
```

```
hlayout2->addWidget (btn2); //.
```

Тепер в менеджер вертикальної компоновки занурюємо по черзі: `hlayout1` з двома текстовими рядками, мітку `lb` і `hlayout2` з двома кнопками

```

vlayout-> addLayout (hlayout1);
vlayout-> addWidget (lb);
vlayout-> addLayout (hlayout2); //.

```

Тут для додавання до одного менеджера компоновки іншого, використаний метод `addLayout ()`. Тепер об'єкт `vlayout` необхідно розмістити на головному вікні `wdt` за допомогою методу `setLayout ()`

```
wdt-> setLayout (vlayout); //.
```

На рис. Б.2 зображено вийшло вікно програми і штриховими лініями виділені горизонтальні менеджери компоновки і мітка, суцільною лінією показаний вертикальний менеджер компоновки. Нижче наведено повний текст вийшла програми. Більш докладно автоматичне розміщення віджетів описано в книгах [17, 18, 19].

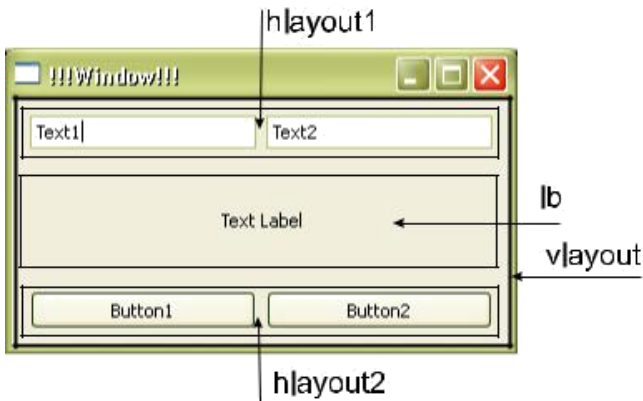


Рисунок Б.2 – Приклад використання менеджерів компоновки

```

1. #include <QtGui>
2. int main(int argc, char* argv[])
3. {
4.   QApplication app(argc, argv); QWidget *wdt
   = new QWidget();
5.   QPushButton *btn1 = new QPushButton(
   "Button1");
6.   QPushButton *btn2 = new QPushButton(
   "Button2");
7.   QLineEdit *edt1 = new QLineEdit("Text1");

```

```

8.  QLineEdit *edt2 = new QLineEdit("Text2");
9.  QLabel *lb = new QLabel("Text Label");
10. QHBoxLayout *hlayout1 = new QHBoxLayout;
11. QHBoxLayout *hlayout2 = new QHBoxLayout;
12. QVBoxLayout *vlayout = new QVBoxLayout;
13. wdt->resize(300,150);
14. wdt->setWindowTitle("!!!Window!!!");
15. lb->setAlignment(Qt::AlignCenter);
16. hlayout1->addWidget(edt1);
17. hlayout1->addWidget(edt2);
18. hlayout2->addWidget(btn1);
19. hlayout2->addWidget(btn2);
20. vlayout->addLayout(hlayout1);
21. vlayout->addWidget(lb);
22. vlayout->addLayout(hlayout2);
23. wdt->setLayout(vlayout);
24. wdt->show();
25. return app.exec();
26. }  //.
```

Створення власних віджетів

Вікно програми, побудоване в попередньому пункті можна реалізувати за допомогою створення власного віджета, який є спадкоємцем класу QWidget. Назвемо клас нового віджету TForm. Об'єкти, розташовані на головному вікні (в нашому випадку дві кнопки, два рядки введення і мітка) будуть вважатися членами нового класу. Опис віджету TForm розмістимо в заголовки form.h:

```

1.  #include <QtGui>
2.  #ifndef F_H_INCLUDED
3.  #define F_H_INCLUDED
4.  class TForm : public QWidget
5.  {
6.  public :
```

```

7.  TForm(); // конструктор
8.  private :
9.  // виджеты главного окна
10. QPushButton *btn1;
11. QPushButton *btn2;
12. QLineEdit *edt1;
13. QLineEdit *edt2;
14. QLabel *lb;
15. QHBoxLayout *hlayout1;
16. QHBoxLayout *hlayout2;
17. QVBoxLayout *vlayout;
18. };
19. #endif // F_H_INCLUDED.

```

Обговоримо нові конструкції, які зустрілися в описі класу TForm. Директиви препроцесора `#ifndef`, `#define`, `#endif` служать тут для запобігання багаторазового включення заголовки в текст програми.

Взагалі директива `define` дозволяє зв'язати ідентифікатор з деякою замісною рядком (можливо порожній), наприклад, рядок `#define PI 3.14159`

ідентифікатором PI ставить у відповідність значення 3,14159. Перед компіляцій програми препроцесор виконає заміну в тексті програми заміщаються ідентифікаторів заміщають рядками.

Директива `ifndef` управляє включенням в програму фрагментів тексту. Весь текст, розташований між `#ifndef NAME` і `#endif` включається в програму за умови, що константа NAME не визначена.

Клас TForm оголошується спадкоємцем класу QWidget. Загальнодоступним є конструктор класу TForm (). Закритими вважаються об'єкти, розташовані на головному вікні кнопки, мітки і т.д. Оскільки об'єкти передбачається створювати динамічно, членами класу є вказівники на них.

Реалізацію методів класу розмістимо у файлі з ім'ям form.cpp:

```

1.  #include "form.h"
2.  TForm::TForm()
3.  {

```

```

4.  btn1 = new QPushButton("Button1");
5.  btn2 = new QPushButton("Button2");
6.  edt1 = new QLineEdit("Text1");
7.  edt2 = new QLineEdit("Text2");
8.  lb = new QLabel("Text Label");
9.  hlayout1 = new QHBoxLayout;
10. hlayout2 = new QHBoxLayout;
11. vlayout = new QVBoxLayout;
12. lb->setAlignment(Qt::AlignCenter);
13. hlayout1->addWidget(edt1);
14. hlayout1->addWidget(edt2);
15. hlayout2->addWidget(btn1);
16. hlayout2->addWidget(btn2);
17. vlayout->addLayout(hlayout1);
18. vlayout->addWidget(lb);
19. vlayout->addLayout(hlayout2);
20. this->setLayout(vlayout);
21. } //.
```

В описі класу присутній тільки один метод конструктор. У ньому відбувається створення об'єктів і розміщення їх на формі. Текст методу відповідає програмі, розібраної в попередньому пункті, і не потребує коментарів за винятком слова `this`. Під `this` розуміється ім'я об'єкта, якому належить конструктор (або інший метод). Це пов'язано з тим, що клас є лише проектом об'єкта, його описом. Самого об'єкта поки не існує, і ім'я його невідоме.

Нарешті, саму програму, що використовує новий клас розмістимо у файлі `main.cpp`:

```

1.  #include <QtGui>
2.  #include "form.h"
3.  int main(int argc, char* argv[])
4.  {
5.  QApplication app(argc, argv);
6.  TForm *wdt = new TForm();
7.  wdt->resize(300,150);
```

```

8.   wdt->setWindowTitle("!!!Window!!!");
9.   wdt->show();
10.  return app.exec();
11.  }   //.
```

Тут динамічно створюється об'єкт `wdt` екземпляр класу `TForm` головне вікно програми. Оскільки створення об'єкта супроводжується виконанням його конструктора, всі інші віджети будуть створені автоматично. На перший погляд застосування власних віджетів призводить до необґрунтованого ускладнення програми. Однак при наповненні програми функціональністю саме створений нами віджет дозволить додавати до програми нові функції.

Наповнення програми функціональністю.

Сигнали і слоти

Відмінною особливістю бібліотеки Qt є наявність механізму сигналів і слотів, що дозволяє зв'язати об'єкти між собою. Вважається, що об'єкти можуть випускати сигнали. Одним з найяскравіших прикладів сигналу є телефонний дзвінок. Піднявши слухавку, людина виконує "обробку" сигналу. З точки зору програми сигнал являє собою метод класу без реалізації і не повертає ніякого значення. Аргументи цього методу можуть передати оброблювачу сигналу додаткову інформацію, як, наприклад, телефон з визначником номера разом з сигналом повідомляє людині додаткову інформацію номер абонента. Стандартні класи бібліотеки Qt мають всі необхідні їм сигнали, але при описі власних класів можна вводити нові сигнали. Необхідною для цього умовою є наявність ключового слова `Q_OBJECT` в секції `private` опису класу (як правило на самому початку), наприклад:

```

class TForm: public QWidget
{
Q_OBJECT
...
signals:
void NameSignal (void);
...
}; //.
```

Тут описаний сигнал з ім'ям `NameSignal` без аргументів. Причиною появи сигналу може бути будь-яка подія, наприклад натискання

кнопки, переміщення повзунка, вибір пункту меню і т.д. Можна також примусово вислати сигнал за допомогою ключового слова

```
emit NameSignal (); //.
```

Оскільки сигнали можуть надсилатися лише об'єктами того класу, якому вони належать, бажано передбачити метод, вислання сигналу

```
class TForm: public QWidget
{
Q_OBJECT
...
signals:
void NameSignal (void);
...
public:
void SendNameSignal (void)
{
emit NameSignal ();
}
...
}; //.
```

Слоти – це обробники сигналів. По суті слоти є звичайними методами класу, але ще мають властивість приєднуватися до сигналів. Об'єкти бібліотеки вже мають необхідний набір слотів для реалізації інтерфейсу програми. Але для наповнення програми потрібної функціональністю доводиться створювати власні слоти.

З'єднання сигналів зі слотами відбувається за допомогою методу connect () класу QObject самого верхнього в ієрархії класів Qt

```
connect (Sender, SIGNAL (NameSignal ()),
Receiver, SLOT (NameSlot ())); //.
```

Тут Sender покажчик на об'єкт, який висилає сигнал, NameSignal () висланий сигнал, Receiver вказівник який бере об'єкт, NameSlot () слот для обробки сигналу. Один сигнал може бути з'єднаний з декількома слотами.

Як приклад адаптуємо програму з попереднього пункту до вирішення завдання варіанту 14 лабораторної роботи №1, в якій необхідно обчислити суму квадратів N цілих чисел $1^2 + 2^2 + 3^2 + 4^2 + \dots + N^2$. Вихідними даними є кількість доданків N . Нехай величину N вводить користувач у рядок введення edt1. При натисканні на кнопку btn1 ви-

конується розрахунок і результат виводиться в рядок введення edt2. До розрахунку мітка lb відображає запрошення до введення вихідних даних, а після повідомлення про виконання.

Для реалізації перерахованих дій в класі TForm опишемо слот Btn1Click (). В результаті заголовки form.h приймає наступний вигляд

```

1.  #include <QtGui>
2.  #ifndef F_H_INCLUDED
3.  #define F_H_INCLUDED
4.  class TForm: public QWidget
5.  {
6.  Q_OBJECT
7.  public:
8.  TForm ();
9.  public slots:
10. void Button1Click (void);
11. private:
12. QPushButton * btn1;
13. QPushButton * btn2;
14. QLineEdit * edt1;
15. QLineEdit * edt2;
16. QLabel * lb;
17. QHBoxLayout * hlayout1;
18. QHBoxLayout * hlayout2;
19. QVBoxLayout * vlayout;
20. };
21. #endif // F_H_INCLUDED.
```

Тут доданий макрос Q_OBJECT і прототип слота. У конструкторі класу виконаємо наступні зміни: сигнал clicked () кнопки btn1 зв'яжемо зі слотом Btn1Click (), сигнал clicked () кнопки btn2 зв'яжемо зі слотом close () головного вікна, змінимо текстові написи на об'єктах. Отримаємо такий зміст конструктора

```

TForm :: TForm ()
{
btn1 = new QPushButton (
QString :: fromLocal8Bit ( "Порахувати" ));
```

```

btn2 = new QPushButton (
QString :: fromLocal8Bit ( "Вихід"));
edt1 = new QLineEdit ( "0");
edt2 = new QLineEdit ( "0");
lb = new QLabel (
QString :: fromLocal8Bit ( "Введіть N"));
hlayout1 = new QHBoxLayout;
hlayout2 = new QHBoxLayout;
vlayout = new QVBoxLayout;
lb-> setAlignment (Qt :: AlignCenter);
hlayout1-> addWidget (edt1);
hlayout1-> addWidget (edt2);
hlayout2-> addWidget (btn1);
hlayout2-> addWidget (btn2);
vlayout-> addLayout (hlayout1);
vlayout-> addWidget (lb);
vlayout-> addLayout (hlayout2);
this-> setLayout (vlayout);
// З'єднання сигналів і слотів
connect (btn1, SIGNAL (clicked ()),
this, SLOT (Button1Click ()));
connect (btn2, SIGNAL (clicked ()),
this, SLOT (close ()));
} //.

```

Тут для перетворення кодування використана функція `QString :: fromLocal8Bit ()`. Це пов'язано з тим, що для в Qt додатках для подання рядків використовується двобайтового кодування Unicode, а при написанні тексту програми використовується одnobайтового кодування (в Windows CP1251, в Linux зазвичай koі8-r). Функція `fromLocal8Bit ()` є методом класу `QString` і призначена для перетворення рядків з одnobайтної системою кодуванням в формат рядків класу `QString`.

Реалізацію слота розберемо трохи докладніше. Спочатку необхідно отримати вихідні дані значення змінної `N`. Введені користувачем дані являють собою текстовий рядок. Для її зчитування об'єкт `edt1` має метод `text ()`, який повертає об'єкт класу `QString`. Клас `QString` володіє методами перетворення рядка в число: `toInt ()`, `toDouble ()`. Тому для зчитування вихідних даних доцільно використовувати присвоювання `N = edt1-> text (). ToInt ();`

де вираз `edt1-> text ()` отримує об'єкт класу `QString`, а `.toInt ()` викликає його метод перетворення тексту в ціле число. Розрахунок суми квадратів цілих чисел не викликає ускладнень і може бути виконаний оператором

```
for (k = 1; k <= N; k ++) {s = s + k * k;}; //.
```

Результат обчислень містить змінна `s`. Для виведення в рядок введення `edt2` її необхідно перетворити в строковий вид. Створимо допоміжний об'єкт `tmp` класу `QString` і за допомогою методу `setNum ()` збережемо текстове представлення цілої змінної `s` в об'єкті `tmp`. І, нарешті, за допомогою методу `setText ()` об'єкта `edt2` виводимо текст з об'єкта `tmp` і міняємо напис мітки `lb`

```
QString tmp;
tmp.setNum (s);
edt2-> setText (tmp);
lb-> setText (
QString :: fromLocal8Bit ( "Виконано"));
В результаті отримаємо наступну реалізацію слота
void TForm :: Button1Click (void)
{
int N;
N = edt1-> text (). ToInt ();
int k;
int s = 0;
for (k = 1; k <= N; k ++) {s = s + k * k;};
QString tmp;
tmp.setNum (s);
edt2-> setText (tmp);
lb-> setText (
QString :: fromLocal8Bit ( "Виконано"));
} //.
```

Аналогічно можна створювати слоти, які виконуються при натисканні інших кнопок.

Створення інтерфейсу за допомогою QtDesigner

Бібліотека Qt комплектується програмою Qt Designer, призначеної для конструювання користувальницького інтерфейсу. Розглянемо коротко методику побудови програми із застосуванням Qt Designer. При запуску програми пропонується створити нове вікно на основі

одного з шаблонів: діалогове вікно з кнопками внизу, вгорі, без кнопок, головне вікно або порожній віджет. Щоб оцінити ефективність використання Qt Designer створимо з його допомогою інтерфейс програми, розглянутої в попередньому пункті. Нехай вікно програми буде засновано на порожньому віджеті. Будемо називати далі проєктоване вікно формою. У лівій частині вікна Qt Designer розташовані панелі з елементами управління, які можна розташувати на формі. У правій частині вікна Qt Designer розташовані панелі інспектора об'єктів, редактора властивостей і т.д. Слід звернути увагу на ім'я форми `Form`, вказане в редакторі властивостей. Воно буде потрібно надалі. Виконаємо далі наступну послідовність дій:

- 1) розмістимо на формі дві кнопки і призначимо їм за допомогою редактора властивостей імена `btn1` і `btn2`;
- 2) виділимо їх разом і виберемо в контекстному меню пункт `Lay Out Horizontally`;
- 3) розмістимо на формі два рядки введення з іменами `edt1` і `edt2`;
- 4) виділимо їх разом і виберемо в контекстному меню пункт `Lay Out Horizontally`;
- 5) розмістимо на формі мітку з ім'ям `lb`;
- 6) виділимо всі об'єкти і виберемо в контекстному меню пункт `Lay Out Vertically`;
- 7) зробимо праве клацання на вільному місці форми і в контекстному меню виберемо пункт `Lay Out Vertically`.

Збережемо проєкт під ім'ям `forma.ui`. В результаті в цьому файлі буде описаний клас з ім'ям `UiForm`. При компіляції кошти бібліотеки Qt перетворюють `forma.ui` в заголовки `ui_forma.h` з описом класу `UiForm` на мові C++. Щоб скористатися вікном, спроектованим в Qt Designer необхідно виконати дві дії. По-перше, клас головного вікна програми должен стати спадкоємцем не тільки класу `QWidget`, але і класу `UiForm`. Тоді заголовки `form.h` стане значно коротше

```

1.  #include <QtGui>
2.  #include "ui_forma.h"
3.  #ifndef F_H_INCLUDED
4.  #define F_H_INCLUDED
5.  class TForm: public QWidget, private Ui ::
    Form {
6.  Q_OBJECT

```

```

7.  public:
8.  TForm ();
9.  public slots:
10. void Button1Click (void);
11. };
12. #endif // F_H_INCLUDED.

```

По-друге, в конструкторі класу головного вікна необхідно виконати команду `setupUi (this)`. Тоді файл `form.cpp` має наступне вміст

```

1.  #include "form.h"
2.  void TForm :: Button1Click (void)
3.  {
4.  int N;
5.  N = edt1-> text (). ToInt ();
6.  int k;
7.  int s = 0;
8.  for (k = 1; k <= N; k ++) {s = s + k * k;};
9.  QString tmp;
10. tmp.setNum (s);
11. edt2-> setText (tmp);
12. lb-> setText (QString :: fromLocal8Bit
    ("Виконано"));
13. }
14. TForm :: TForm ()
15. {
16. setupUi (this);
17. connect (btn1, SIGNAL (clicked ()),
18. this, SLOT (Button1Click ()));
19. connect (btn2, SIGNAL (clicked ()),
20. this, SLOT (close ()));
21. }

```

Як видно з лістингів, використання коштів Qt Designer дозволяє дещо скоротити текст програми.

В даному розділі розглянуто лише мала частина можливостей, які надає бібліотека Qt. Освоєння цієї малої частини повинно бути достатнім для виконання наступних двох лабораторних робіт. Бажаючи більш повно освоїти програмування з використанням Qt можна порадити вивчення книг [17, 18, 19].