**МІНІСТЕРСТВО ОСВІТИ І НАУКИ,
МОЛОДІ ТА СПОРТУ УКРАЇНИ**

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

**Методичні рекомендації
до практичних завдань
з модуля "Принципи розробки Windows-додатків"
навчальної дисципліни**

"ОСНОВИ ПРОГРАМУВАННЯ ТА АЛГОРИТМІЧНІ МОВИ"

**для студентів напряму підготовки
"Комп'ютерні науки"
всіх форм навчання**

Харків. Вид. ХНЕУ, 2011

Затверджено на засіданні кафедри інформаційних систем.
Протокол № 5 від 25.11.2010 р.

Укладачі: Федорченко В. М.
Тарасов О. В.
Гриньов Д. В.

М54 Методичні рекомендації до практичних завдань з модуля "Принципи розробки Windows-додатків" навчальної дисципліни "Основи програмування та алгоритмічні мови" для студентів напряму підготовки "Комп'ютерні науки" всіх форм навчання / укл. В. М. Федорченко, О. В. Тарасов, Д. В. Гриньов. – Х. : Вид. ХНЕУ, 2011. – 156 с. (Укр. мов.)

Викладено матеріал, що включає методику проведення практичних занять з навчальної дисципліни. Велику увагу приділено засвоєнню методу використання теорії, придбання професійних компетенцій, а також практичних вмінь, необхідних для виконання лабораторних робіт. Наведено багато практичних прикладів, що ілюструють реалізацію конкретних завдань.

Рекомендовано для студентів, викладачів і користувачів, які вивчають основи програмування на алгоритмічній мові C++ та застосовують її у різних галузях економіки.

Методичні рекомендації

Практичне заняття – це форма організації навчального процесу, що припускає виконання студентами за завданням і під керівництвом викладача однієї або декількох практичних робіт. І якщо на лекції головна увага студентів зосереджується на роз'ясненні теорії навчальної дисципліни, то практичні заняття служать для навчання методам її застосування. Як правило, практичні заняття проводяться паралельно із читанням усіх основних навчальних дисциплін. Головною метою практичних занять є засвоєння методу використання теорії, придбання професійних компетенцій, а також практичних вмінь, необхідних для виконання лабораторних робіт (ІНДЗ) і вивчення наступних дисциплін.

Практичні заняття відіграють важливу роль у виробленні у студентів навичок застосування отриманих знань для вирішення практичних завдань разом з викладачем. Практичні заняття з навчальної дисципліни проводяться в комп’ютерних класах через 2 лекції і логічно продовжують роботу, почату на лекції, допомагають студенту якісно виконати завдання поточної лабораторної роботи.

Загальна структура практичних занять: вступне слово викладача; відповіді на запитання студентів; практична частина як планова; заключне слово викладача.

Найважливішою частиною будь-якої форми практичних занять є вправи. Основа вправи – приклад, який розбирається з позицій теорії, розвинутої в лекції. Як правило, основна увага приділяється формуванню конкретних вмінь та навичок, що визначає зміст діяльності студентів. Проводячи вправи, слід спеціально звертати увагу на формування зда-тності до алгоритмічного мислення, осмислення і розуміння завдання. Студент повинен розкрити і виявити свої здатності, свій особистий потенціал. Критерії ефективності практичного заняття: цілеспрямованість, планування, організація, стиль проведення, відносини "викладач – студенти", керування групою.

Усі практичні заняття з навчальної дисципліни проводяться за єдиною схемою:

Вступ (5 – 8 хвилин): перевірка відвідування занять, оголошення теми, мети, основних питань і порядку проведення заняття; проведення контрольного опитування, де студенти письмово відповідають на запитання з матеріалу поточних лекцій і здають відповіді.

Постановка й аналіз завдання (7 – 8 хвилин): студенти розбирають фізичну сутність, вхідні та вихідні дані, виконують математичний опис завдання, що пропонується викладачем. Один зі студентів працює біля дошки.

Створення специфікації програми (7 – 8 хвилин): студенти створюють специфікацію програми та вибирають форму вхідних / вихідних даних для завдання, що запропоноване викладачем. Один зі студентів виконує завдання біля дошки.

Побудова алгоритму (7 – 8 хвилин): на підставі математичного опису завдання студенти розробляють порядок дій та схему алгоритму розглянутого завдання. Один зі студентів виконує завдання біля дошки.

Розробка програми, трансляція і редагування програми, тестування програми (15 – 25 хвилин): кожен студент самостійно створює програмний код (додаток), для рішення завдання використовує приклад, наведений у методичній розробці, у ході компіляції програми виявляються та виправляються синтаксичні помилки, у ході тестування виявляються логічні помилки. Викладач відповідає на запитання, індивідуально допомагає студенту.

Самостійна робота (25 – 30 хвилин): студенти самостійно або в міні-групах розв’язують завдання, що запропоноване викладачем, викладач відповідає на запитання, індивідуально допомагає, оцінює роботу студентів.

Закінчення (5 хвилин): підсумки заняття, аналіз роботи групи на занятті, узагальнення питань, які частіше всього виникали при виконанні завдань заняття. Отримання завдання для самопідготовки.

Практичні заняття покликані для поглиблення, розширення, деталізації знань, отриманих на лекції в узагальненій формі, і сприяють виробленню навичок професійної діяльності. Вони розбудовують наукове мислення й мову, дозволяють перевірити знання студентів і виступають як засіб оперативного зворотного зв'язку.

Практичне заняття не повинне бути топтанням на одному місці. Слід організувати свою роботу на практичному занятті так, щоб постійно відчувати зростання складності виконуваних завдань, отримувати позитивні емоції від переживання власного успіху у програмуванні, бути зайнятим напруженою творчою роботою, пошуками правильних і точних розв'язків.

Практичне заняття 1. Розробка програм обробки масивів з використанням покажчиків

Використання покажчиків у мові С++ настільке поширене, що навіть важко уявити собі більш-менш професійну програму, в якій би не використовувалися ці типи даних. Для успішного написання програм з використанням покажчиків необхідно глибоко опрацювати теоретичний матеріал і чітко знати відповіді на такі запитання:

1. Що таке покажчик та які основні операції можуть з ним виконуватися?
2. Який взаємозв'язок існує між покажчиками і масивами?
3. Основи динамічного розподілу пам'яті.
4. Що таке покажчик на функцію і яким чином він використовується?

Частково деякі з цих питань розглядалися в роботі [21]. Розглянемо ще декілька практичних прикладів.

Завдання 1.1. Написати програму, яка визначає добуток двох матриць, який слід обчислити в окремій функції. Виділити пам’ять під масиви динамічно. Обробку елементів масиву та виклики функцій здійснити з використанням покажчиків.

```
#include "stdafx.h"
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <locale.h>
// Виділення пам'яті, ініціалізація і виведення значень
// одновимірного масиву ( через покажчики)
int fmass(int N, bool init)
{
    int *p1, *p2;
    // Виділення динамічної пам'яті під одновимірні масиви
    p1=new int[N];
    p2=(int*) malloc(N*sizeof(int));
    // ініціалізація елементів масивів
    if (init){
```

```
            // ініціалізація датчика випадкових чисел
            srand( (unsigned)time( NULL ) );
      // Увесь масив
         for (int i=0; i<N; i++) {
               *(p1+i)=i; *(p2+i)=rand();
         } // for i
       } else {
     // окремі елементи
        *(p1+1)=3;        *(p2+3)=2;
       }
       // Виведення масивів на екран
       printf("Виведення %s ініціалізованого масиву\n", (init?"":"не"));
       for (int i=0; i<N; i++) {
            printf("i=%2d p1=%10d p2=%10d \n", i, *(p1+i), *(p2+i));
       }
       // Звільнення пам'яті
       delete []p1;          free(p2);
       return 0;
}
// Заповнення масиву випадковими значеннями
void SetArray(int** pA, int Row, int Col)
{
       for (int i=0; i<Row; i++)
       for (int j=0; j<Col; j++ ){
            *(*(pA+i)+j)=rand()% 4;
       }
}
// Виведення двовимірного масиву  на екран
void ShowArray(int** pA, int Row, int Col, char* Text)
{      printf("%s \n",Text);
       for (int i=0; i<Row; i++){
         for (int j=0; j<Col; j++ ){
               printf("%3d", *(*(pA+i)+j));
        }
         printf("\n");
       }
}
```

```
// Виділення пам'яті і ініціалізація двох
// двовимірних масивів A і B( через покажчики)
// Визначення добутку масивів : C = A x B
void fmass2(int N, int M, int K)
{
        int **A, **B, **C;
        // Динамічне виділення пам'яті під двовимірні масиви
   A=new int*[N]; for (int i=0; i<N; i++) *(A+i)=new int[M];
   B=new int*[M]; for (int i=0; i<M; i++) *(B+i)=new int[K];
   C=new int*[N]; for (int i=0; i<N; i++) *(C+i)=new int[K];
        int sum;

        SetArray(A,N,M);
        SetArray(B,M,K);

        for (int i=0; i<N; i++) {          // прохід по рядках результуючої матриці
              for (int k=0; k<K; k++) { // прохід по стовпцях результуючої
матриці
                        sum=0;
                        for (int j=0; j<M; j++){ // сума (рядок х стовпець)
                              sum += *(*(A+i)+j) * *(*(B+j)+k);
                        } // for (k)
                        *(*(C+i)+k) = sum;          // елемент результуючої матриці
               } // for (j)
        } // for (i)
        printf("Добуток масивів A x B = C\n");
        ShowArray(A,N,M,"Масив A:");
        ShowArray(B,M,K,"Масив B:");
        ShowArray(C,N,K,"Результуючий масив C:");

   delete []A;
   delete []B;
   delete []C;
}

void main()
```

```
{
   setlocale(0,"RUS");
        // Визначення покажчиків на функцію
        int  (*pf) (int,bool);
        void (*pf2)(int, int, int);
   pf=fmass;    // адреса функції fmass
        // Виклик функції fmass
        pf(4,false);
        pf(5,true);
        pf2=fmass2;  // адреса функції fmass2
        // Виклик функції fmass2
        pf2(3,2,3);
        return;
}
```

Результати роботи програми представлені на рис. 1.1.

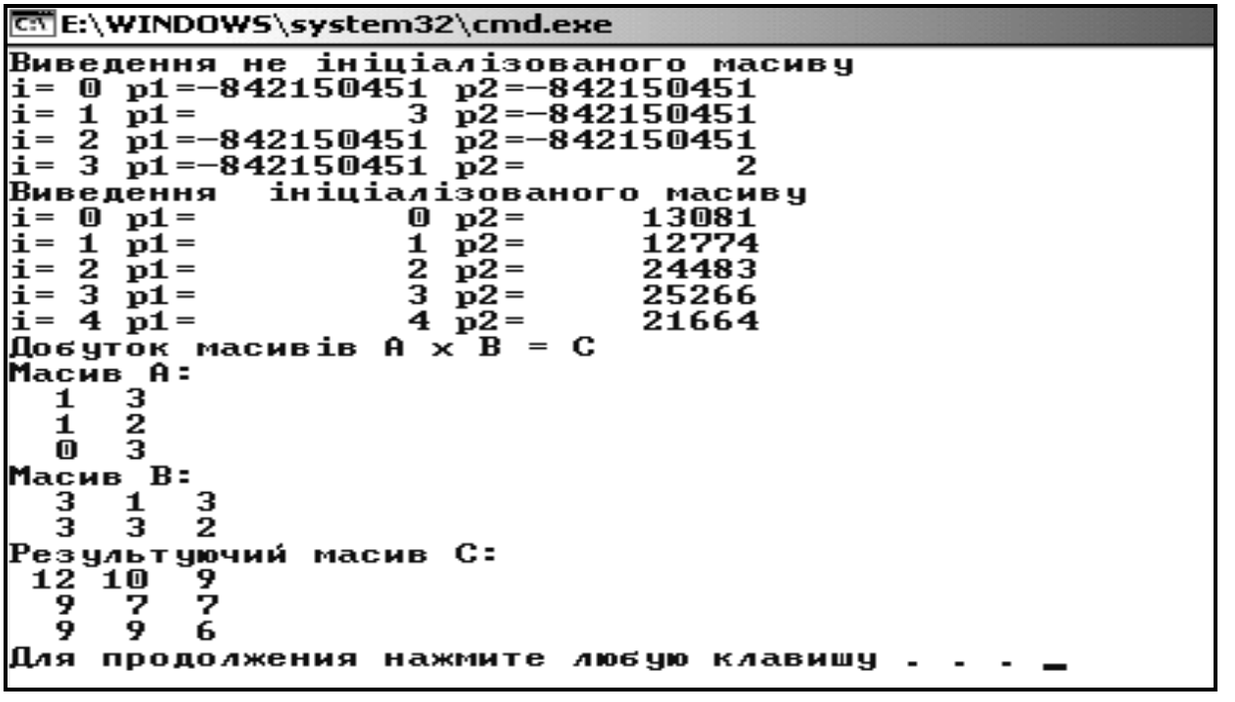

Рис. 1.1. **Вікно з результатами роботи програми до завдання 1.1**

Завдання 1.2. Вивести різними кольорами на консоль таблицю символів ASCII з кодами від 32 (пробіл) до 255. Виведення здійснити у два стовпчики. Встановлення та отримання поточного положення курсора здійснити за допомогою функцій, які викликати через покажчик.

```
#include "stdafx.h"
#include <windows.h>
#include <fstream>
#include <iostream>
#include <string>

#define LEFT_COL      5     //  Положення першого стовпчика
#define RIGHT_COL   45   //  Положення другого стовпчика
using namespace std;

HANDLE consoleOutput;                          // Хендл консолі
CONSOLE_SCREEN_BUFFER_INFO csbi; // Характеристика буфера консолі
COORD CurPos;                                    // Координати

// Функція переміщення курсора у позицію <X,Y>  на екрані
void Set_XY(int X, int Y)
{
        CurPos.X=X;  CurPos.Y=Y;
        SetConsoleCursorPosition(consoleOutput, CurPos);
}

// Функція визначення поточних координат курсора
void Get_XY(int &X, int &Y)
{
        // Отримання інформації про стан консолі ( зокрема координат курсора)
        GetConsoleScreenBufferInfo( consoleOutput, &csbi );
        // Визначаємо окремі координати курсора
        X=csbi.dwCursorPosition.X;
        Y=csbi.dwCursorPosition.Y;
}

// Головна  програма
int main()
{ int x,y,xc;
  char Simb[224];
```

```
 void (*pSetCur) (int, int);        // покажчик на функцію
 void (*pGetCur) (int&,int&);  // покажчик на функцію

 pSetCur=Set_XY;  pGetCur=Get_XY;
 setlocale(0,"RUS");

 // Отримуємо хендл консолі
 consoleOutput = GetStdHandle(STD_OUTPUT_HANDLE);
 // Заповнюємо масив символами з кодами ASCII від 32 до 255
 for (int i=0; i<224; i++) *(Simb+i)=i+32;

 // Очищаємо екран і встановлюємо початкове положення курсора
 system("CLS");   pSetCur(x=LEFT_COL,1);
 // Виводимо у циклі символи з масиву  різними кольорами
 for (int i=0; i<224; i++){
   SetConsoleTextAttribute ( consoleOutput, i );
   printf("%c ",*(Simb+i));
        // Після кожних 16 символів переходимо на 2 рядки
        if ((i+1)%16==0){ //printf("\n");
     pGetCur(xc,y);
          // Після 13 рядка повертаємость на перший
          if (y==13) {
                y=-1; x=RIGHT_COL;
          };
          pSetCur(x,y+2);
        }
 }
   // Установка  кольору тексту і фону
       SetConsoleTextAttribute ( consoleOutput,        FOREGROUND_BLUE  |
(BACKGROUND_GREEN | BACKGROUND_RED | BACKGROUND_ IN-
TENSITY));

       // Виводимо текст з нової позиції
       Set_XY(15,20); printf("Програма закінчила роботу\n");
   getchar();
       return 0;
}
```

Результати роботи програми представлені на рис. 1.2.

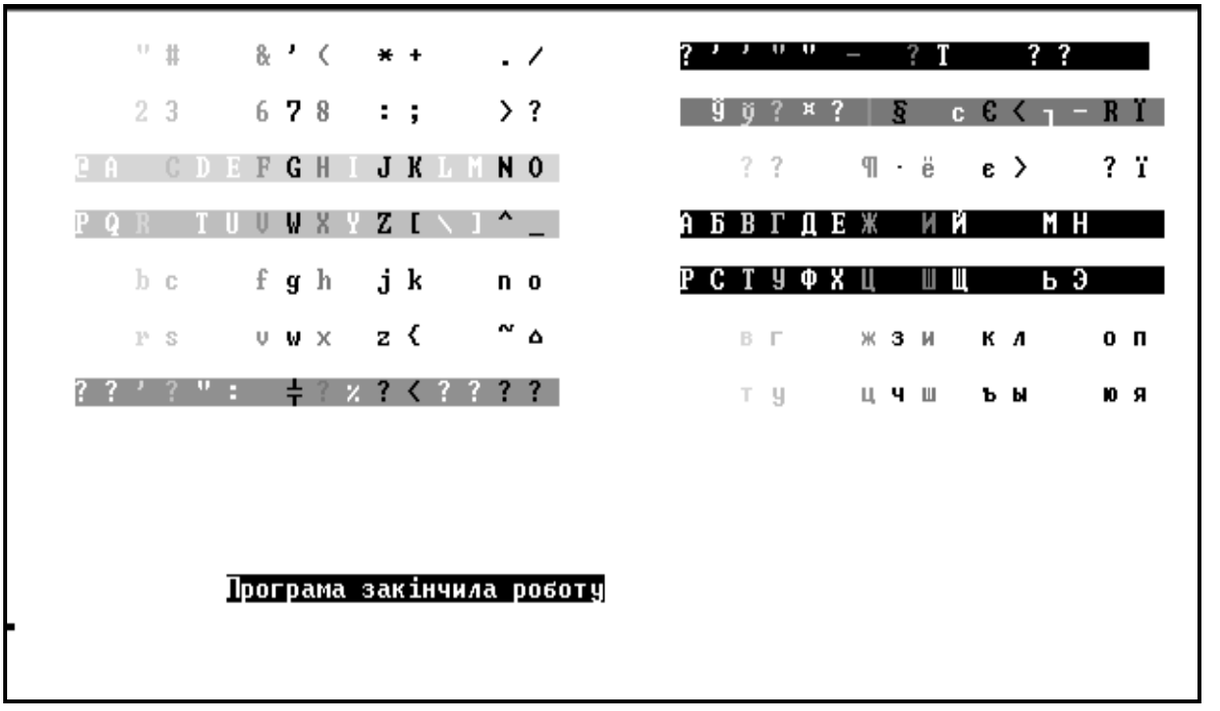

Рис. 1.2. **Вікно з результатами роботи програми до завдання 1.2**

Завдання 1.3. Написати програму, яка виводить на екран консолі графік функції. У параметрах функції, що відображає графік на екрані, передати адрес відображуваної функції через покажчик. У наведеному прикладі відображується функція $y = \cos(x) \cdot \cos(2x)$, однак передбачено, що можуть відображатися інші графіки.

```
#include "stdafx.h"
#include <math.h>
#include <iostream>
#include <Windows.h>
using namespace std;

typedef double (*FUN_GRAPH)(double x);  // Тип даних  як покажчик на функцію
#define WIDTH   79                      // Ширина консолі для виведення графіка
#define HEIGHT  24                      // Висота консолі для виведення  графіка

HANDLE consoleOutput;           // Хендл консолі
CONSOLE_SCREEN_BUFFER_INFO csbi; // Характеристика буфера консолі
```

```
COORD CurPos;                  // Координати
////////////////////////////////////////////////////////////////////////////
//                    ПРОТОТИПИ  ФУНКЦІЙ
////////////////////////////////////////////////////////////////////////////
// Функція переміщення курсора у позицію <X,Y>  на екрані
void Set_XY(int X, int Y);
// Функція визначення координати X на екрані  залежно від значення ар-
гументу x
int XPos(double xmin, double xmax, double x);
// Функція визначення координати Y на екрані  залежно від значення
функції y
int YPos(double ymin, double ymax, double y);

double fun1(double x);    // перша функція
double fun2(double x);    // друга функція
double fun3(double x);    // третя функція
// Функція відтворення графіка y=fun(x)
void ShowGraph(FUN_GRAPH fun, double xmin, double xmax);
// Головна  програма
int main()
{   FUN_GRAPH fun;   // покажчик на функцію

        // Отримуємо хендл консолі
   consoleOutput = GetStdHandle(STD_OUTPUT_HANDLE);
   fun=fun3;                       // присвоєння адреса функції покажчику
   ShowGraph(fun, 0, 3.14);    // відтворюємо графік функції
   return 0;
}
// Функція відтворення графіка y=fun(x)
void ShowGraph(FUN_GRAPH fun, double xmin, double xmax)
{   double y,dy,ymin, ymax;
   double x,dx;
        int xC, yC;
        system ("cls");
        dx=(xmax-xmin)/WIDTH;           // визначаємо шаг dx
        // визначаємо мінімальне та максимальне значення функції
```

```
        for (x=xmin, ymin=ymax=fun(xmin); x<xmax; x+=dx) {
      y=fun(x);
      if (ymin>y) ymin=y;
      if (ymax<y) ymax=y;
        }
        // відтворюємо графік функції на екрані консолі
        for (x=xmin; x<xmax; x+=dx) {
      y=fun(x);
          xC=XPos(xmin,xmax,x);  yC=YPos(ymin,ymax,y);
          Set_XY(xC, yC);       cout<<"*";
        }
   cout<<"\n";
}
double fun1(double x) { return sin(x);  }           // перша функція
double fun2(double x) { return cos(3*x);}                  // друга функція
double fun3(double x) { return cos(x)*cos(2*x);};        // третя функція

// Функція визначення координати X на екрані в залежності від значення
аргументу x
int XPos(double xmin, double xmax, double x)
{   return (int)(WIDTH*(x-xmin)/(xmax-xmin));  }

// Функція визначення координати Y на екрані залежно від значення
функції y
int YPos(double ymin, double ymax, double y)
{
        return (int)(HEIGHT*(ymax-y)/(ymax-ymin));
}
// Функція переміщення курсора у позицію <X,Y>  на екрані
void Set_XY(int X, int Y)
{
        CurPos.X=X;  CurPos.Y=Y;
        SetConsoleCursorPosition(consoleOutput, CurPos);
}
// Функція визначення поточних координат курсора
void Get_XY(int &X, int &Y)
{
```

```
    // Отримання інформації про стан консолі ( зокрема координат курсора)
       GetConsoleScreenBufferInfo( consoleOutput, &csbi );
       // Визначаємо окремі координати курсора
       X=csbi.dwCursorPosition.X;
       Y=csbi.dwCursorPosition.Y;
}
```

Результати роботи програми представлені на рис. 1.3.

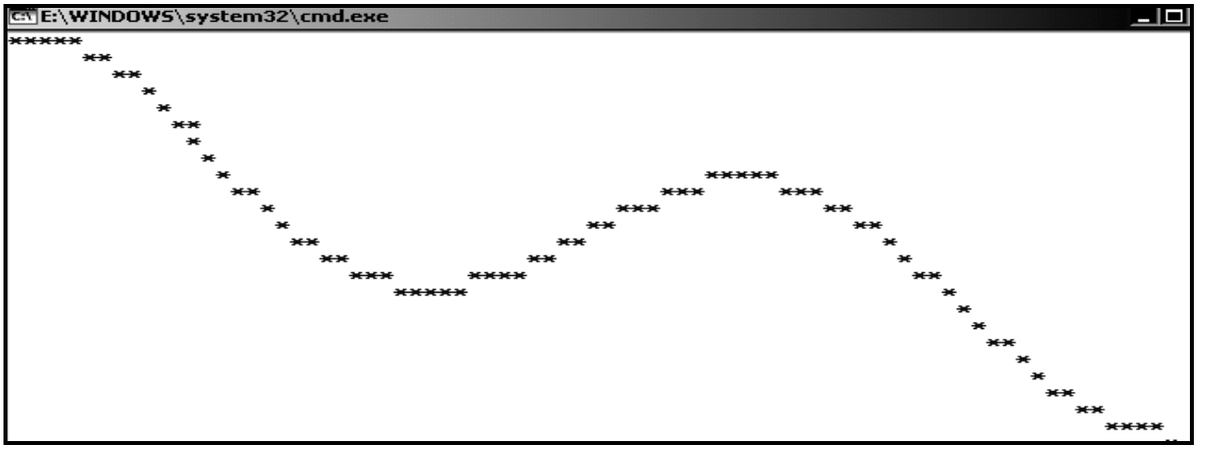

Рис. 1.3 **Вікно з результатами роботи програми до завдання 1.3**

Зробити модифікацію програми таким чином, щоб:

1. Графік відображуваної функції вибирався через меню.
2. На екрані з'являвся надпис з функцією, що відтворюється.
3. На екрані відображалися координатні вісі.

Практичне заняття 2. Розробка програм обробки рядків, що зберігаються на зовнішніх носіях

Незважаючи на те, що велика кількість програм пишеться для виконання розрахунків, пов'язаних з обробкою числових даних, існує не менш вагомий клас завдань, які пов'язані з обробкою текстових даних та файлів, в яких ці дані можуть зберігатися. Глибоке розуміння суті рядків та способів роботи з файлами нерозривно пов'язане з:

1. Вивченням можливостей введення-виведення даних та освоєння методики обробки інформації з використанням рядків та багатофайлових програм.

2. Визначенням і правилами опису рядків з використанням засобів стандартної мови С як масиву символів та функцій, які використовуються для їх обробки.

3. Визначенням і правилами опису рядків з використанням класу string стандартної бібліотеки шаблонів та методів цього класу.

4. Вивченням основних функцій для роботи з файлами стандартної мови С.

5. Вивченням основних методів для роботи з потоками стандартного класу ios та його похідних.

У додатку А та додатку Б наведений перелік основних функцій та методів, що використовуються при вирішенні подібних завдань.

Зазначимо, що наведеними в даному практичному занятті прикладами ця тема не обмежується і в наступних практичних заняттях будуть приклади, які ілюструють тему роботи з файлами та рядками.

Окремо слід зазначити, що при виконанні наведених прикладів необхідно в створюваному проекті відключити опцію обробки UNICODE-рядків. Це робиться таким чином: вибрати пункт меню Project -> Properties, у результаті чого з'явиться вікно (рис. 2.1).

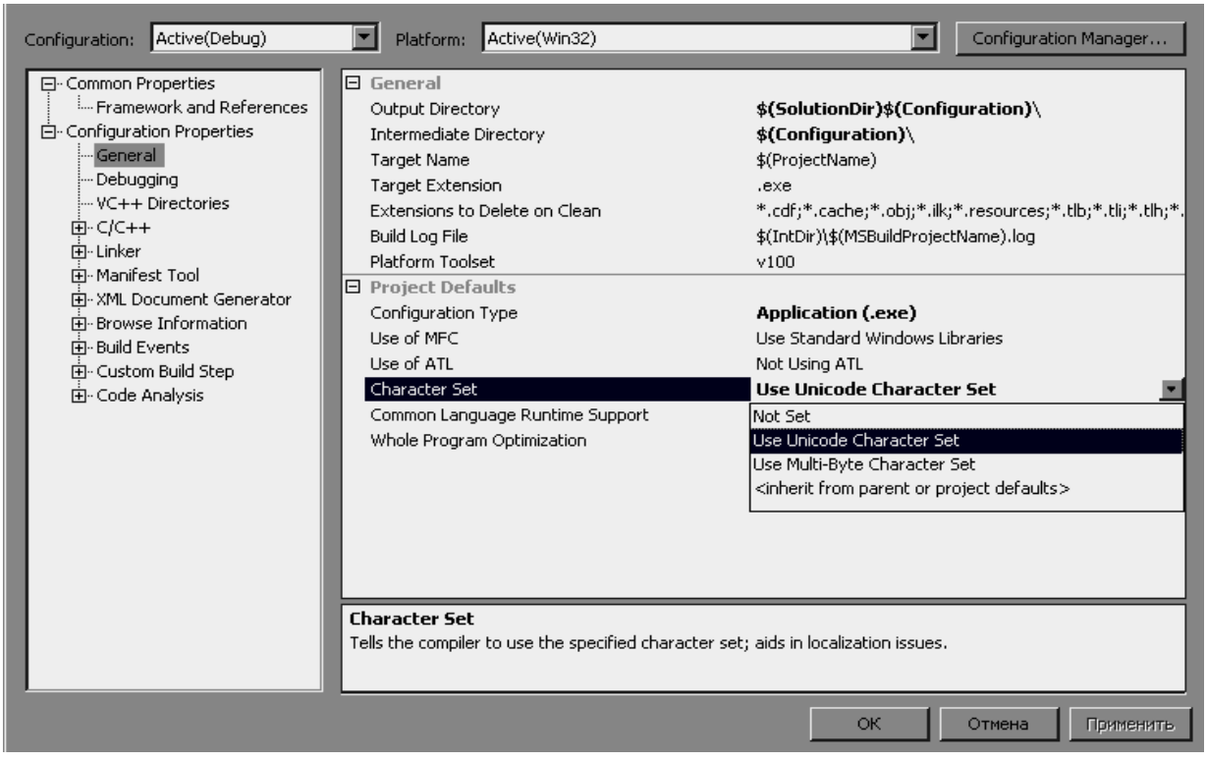

Рис. 2.1. **Відключення роботи з UNICODE-рядками**

Далі вибрати Configuration Properties -> General -> Character Set і замість Use Unicode Character Set встановити Not Set.

Завдання 2.1. Написати програму, яка здійснює пошук заданих символів "iso" у рядку і виводить повідомлення про результати пошуку. У подальшому програма шукає символи, які є закінченням речення, і виводить знайдені речення на екран.

```
#include "stdafx.h"
#include "locale.h"
#include <stdio.h>
#include <string.h>
#include <windows.h>

int main(void)
{
  // рядок, у якому шукаємо символи
  char string1[70] = " Example! This is the main application source file. That's all.";

  char string2[5] = "iso";  // символи, які шукаємо у рядку
  char *ptr;
  setlocale(0,"RUS");
  system("cls");

  printf("Пошук символів <iso> в рядку: \n <Example! This is the main application source file. That's all.>\n\n");
  ptr = strpbrk(string1, string2);

  if (ptr)
    printf("1) strpbrk знайшла перший символ:\n %c - %s \n\n", *ptr, ptr);
  else
    printf("1) strpbrk не знайшла  жодного символа  масиву\n\n");

printf("Шукаємо символи: кома, крапка, оклик, зірочка:\n");

/* strtok розміщує NULL-термінатор попереду знайденого зразка */
ptr = strtok(string1, ",.!");   // шукаємо символи: кома, крапка, оклик
```

```
if (ptr)  printf("%s\n", ptr);   // виводимо рядок до символа, якщо щось знайшли

/* Повторний виклик strtok,  якщо використовується NULL як перший параметр,  повертає показчик на наступний пошуковий символ */
while (ptr)
{
 ptr = strtok(NULL, ",.!*");          // знову пошук
 if (ptr)   printf("%s\n", ptr);   // знайшли? виводимо на екран
}

return 0;
}
```

Результати роботи програми представлені на рис. 2.2.

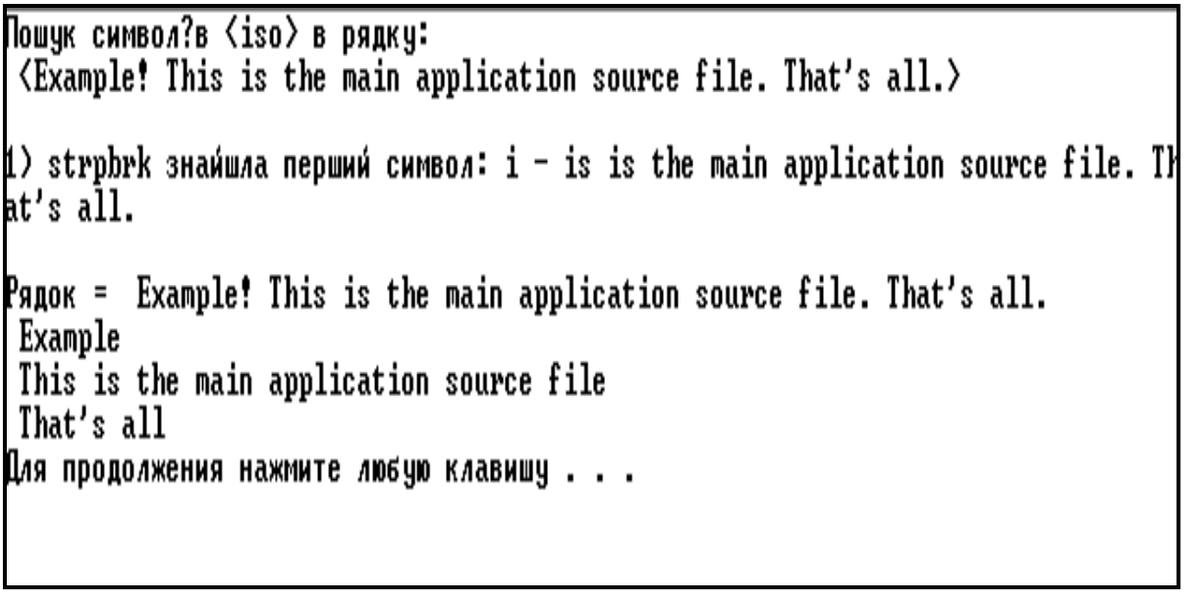

Рис. 2.2. **Вікно з результатами роботи програми до завдання 2.1**

Завдання 2.2. Написати програму, яка формує новий рядок з двох існуючих, причому пам'ять під новий рядок треба виділити динамічно. Далі виділяється підрядок довжиною у 93 символи, починаючи з 12 символа. У сформованому рядку здійснюється пошук речень і визначається максимальна довжина речення. Додатково треба знайти такі речення, в яких існує підрядок "les".

```
#include "stdafx.h"
#include <string.h>
#include <iostream>
using namespace::std;

int main()
{
  // перший рядок
  char S[]="Information Resources. Help files. Microsoft Press Books. ";
  // другий рядок
  char T[110]="C/C++ modules within a program typically share definitions of
types and data structures among themselves. ";
  char *G;                              // покажчик на об'єднаний рядок
  char *ptr;                            // покажчик на рядок
  int Lmax=strlen(S)+strlen(T);   // довжина об'єднаного рядка
  G=new char[Lmax+1];             // виділяємо пам'ять під рядок
  strcpy(G,S);                          // копіюємо перший рядок
  strcat(G,T);                 // дописуємо другий рядок
  strncpy(G,G+12,93);   // виділяємо підрядок з 12 символа довжиною 93
символи
  G[93]=0;                    // дописуємо ознаку кінця рядка
  printf("<%s>\n\n",G);               // виводимо новий рядок на екран
  // Виводимо  на екран усі речення
  ptr=strtok(G,".!?");           // пошук символів кінця речення
  Lmax=0;
  while (ptr!=NULL) {  // цикл,  доки знаходимо крапку, знак оклику або пи-
тання
          // знаходимо максимальну довжину речення
          if (strlen(ptr)>Lmax) Lmax=strlen(ptr);

          // перевіряємо, чи знаходиться у рядку підрядок "les"
          if (strstr(ptr,"les")){
        cout << "Find \"les\" in : "<< ptr << endl;    // виведення, якщо знайдено
          }
         ptr=strtok(NULL,".!?");      // пошук наступного речення
  }
```

```
 // Виводимо максимальну довжину рядка
 cout << "Max length of sentence= " << Lmax << endl;
 return 0;
}
```

Результати роботи програми представлені на рис. 2.3.

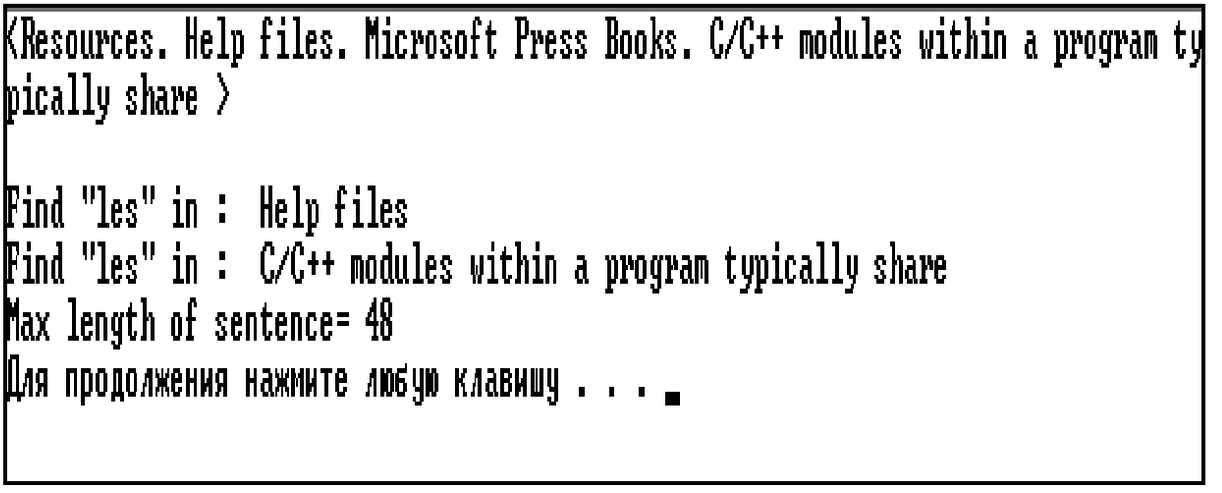

Рис. 2.3. **Вікно з результатами роботи програми до завдання 2.2**

Завдання 2.3. Створити програму, яка читає інформацію з одного текстового файла, рядок якої містить текстові, числові та знову текстові дані, і записує їх у новий файл у три стовпчики та виводить на екран рядок із зазначенням поточної позиції у файлі. У подальшому новий файл відкривається для зчитування і виводиться на екран з позначенням кінцевої позиції рядка у файлі.

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
#include <string.h>
#include <iomanip>  //  для setw - манипулятор в/в

using namespace std;

int main()
{
        char s[80],t[80]; // масиви для збереження рядків
        int itt;                // змінна в яку читається число з файла
        int poz=0,len=0;  // позиція у файлі та довжина рядка
```

```
        fstream infile;    // файл для зчитування
        fstream outfile;   // файл для запису, а згодом для зчитування

    // відкриваємо файл для зчитування
        infile.open("input.txt",ios::in | ios::binary);
    // відкриваємо файл для запису
        outfile.open("output.txt",ios::out | ios::binary);

        if (!infile)  // перевірка успішності відкриття файла для читання
        {
          cout<< "Error open file <input.txt>" << endl;  // повідомлення про по-
милку
          return 0;
        }
        if (!outfile)  // перевірка успішності відкриття файла для запису
        {
          // повідомлення про помилку
           cout<< "Error open file <output.txt>" << endl;
          return 0;
        }

        cout << "=============================================\n";
        cout << " Read from file <Input.txt>    write to file  <Output.txt>\n\n";

    // читання з файла тексту, числа і тексту, розділених пробілами
        while (!infile.eof())  // послівне (до пробілу) виведення на екран
        {
                infile >> s >> itt >> t;    // читання з файла
                // виведення інформації на екран  у три стовпчики
                // з зазначенням поточної позиції у файлі
                cout <<setw(15)<<s <<setw(10) <<itt<<setw(15) <<t << " Poz=" <<
(poz=infile.tellg())<<endl;
                // запис структурованого рядка у файл
                if (poz==-1) break;        // кінець файла - вихід з циклу
                outfile<<setw(15)<<s <<setw(10) <<itt<<setw(15) <<t <<"\n";
        }
        // закриття файлів
```

```
        infile.close();
        outfile.close();

    cout << "\n\n=========================================\n\n";
    cout << " Read from file <Output.txt>    write to Screen\n";

    // повторне відкриття сформованого файла  для читання
        outfile.open("output.txt",ios::in | ios::binary);

        if (!outfile)  // перевірка успішності
        {
          cout<< "Error open file <output.txt> step 2" << endl;
          return 0;
        }

    // цикл зчитування рядків
    while (!outfile.eof())  // перевірка кінця досягнення файла
        {
                outfile.getline(s,80); // зчитування рядка (без символа \n)
                len=strlen(s);          // довжина рядка

                // заміна символа "переведення каретки" (CR) на кінець рядка
                if (s[len-1]==0x0D)        s[len-1]=0;

                // виведення рядка на екран з позначенням кінцевої позиції
                cout << setw(20) << s <<" ==> " << outfile.tellg() <<  endl;
        }

        // закриття файла
        outfile.close();
        return 0;
}
```

Результати роботи програми представлені на рис. 2.4.

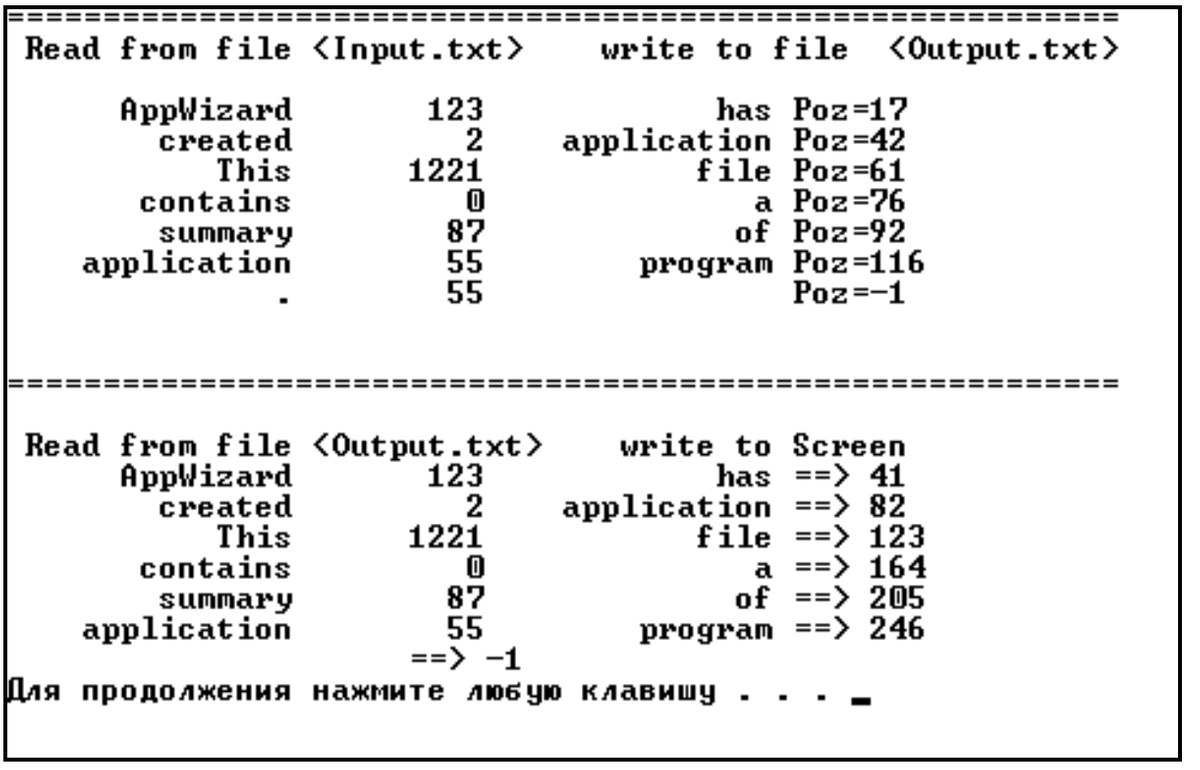

Рис. 2.4. **Вікно з результатами роботи програми до завдання 2.3**

Завдання 2.4. Навести приклад програми, яка ілюструє не тільке послідовну обробку файла, але й зміщення поточної позиції на задану кількість байтів. У програмі спочатку читаються рядки з одного файла і записуються в інший, але не прямо, а використовуючи просте перетворення символів (ASCII-код символа збільшується на одиницю). Тобто рядки кодуються. У подальшому покажчик поточної позиції вхідного файла переміщається у початок і знову починається його зчитування, але рядки читаються не від початку, а зі зміщенням (у прикладі на 10 байт). Результати обробки виводяться на екран. При другому проході перетворення рядків не відбувається.

```
#include "stdafx.h"
#include <stdio.h>
#include <string.h>
#include <locale.h>
#define MAXLEN        120     // максимальна довжина рядка

int main()
{ FILE *infile=NULL, *outfile=NULL; // показчики на файли
  char s[MAXLEN+1];             // буфер для зчитування рядків
```

```
int poz=0;                    // номер позиції
char *rez;                    // показчик на рядок
                             // (для контролю успішності зчитування)

 outfile=fopen("output.txt","wt");  // відкриваємо файл для запису
     fputs("**********\n",outfile);  // записуємо в нього рядок зірочок

     infile=fopen("input.txt","rt");    // відкриваємо файл для зчитування
     if (infile==NULL)                  // перевіряємо, чи відкрився файл
     {
            printf("Error input file\n");  // повідомлення про помилку
            fclose(outfile);              // закриваємо файл для запису
            return 1;                    // вихід з програми
     }
     printf("**********************************\n");
     printf("Output whole string from file    \n");
     printf("**********************************\n\n");
     // цикл зчитування рядків з файла до його кінця
     while(!feof(infile))
     {
            if (!fgets(s,80,infile)) break;   // читаємо рядок, вихід по кінцю

            printf("=> %s",s);    // виводимо на екран консолі
           // у циклі обробляємо символи рядка
            for (int i=1; i<strlen(s); i+=2)
            {
                  s[i]=s[i]+1;      // змінюючи код символа на одиницю
            }
           // записуємо перетворений рядок у вихідний файл
            fputs(s,outfile);
     }
     printf("**********************************\n");
     printf("Output string without ten simbols\n");
     printf("**********************************\n\n");
  // встановлюємо показчик поточної позиції файла на початок
 rewind(infile);
     while(!feof(infile))      // знову в циклі обробляємо файл до його кінця
```

```
    {
        poz=ftell(infile);       // визначаємо поточну позицію у файлі
      fseek(infile,10,SEEK_CUR); // зміщуємо її на 10 байтів
        if (!fgets(s,80,infile)) break ;      // читаємо рядок (не увесь!!!)
        printf("=> %s",s);       // виводимо прочитане на екран консолі
        fputs(s,outfile);        // дописуємо рядок у файл
    }
    fclose(infile);    fclose(outfile);   // закриваємо файли
    return 0;
}
```

Результати роботи програми представлені на рис. 2.5.

```
**********************************
Output whole string from file
**********************************

=> AppWizard has created this Lab02-2 application for you.
=> This file contains a summary of what you will find in
=> each of the files that make up your Lab02-2 application.
=> This is the main project file for VC++ projects generated
=> using an Application Wizard. It contains information
=> about the version of Visual C++ that generated the file,
=> and information about the platforms, configurations,
=> and project features selected with the Application Wizard.
**********************************
Output string without ten simbols
**********************************

=> has created this Lab02-2 application for you.
=> contains a summary of what you will find in
=> e files that make up your Lab02-2 application.
=> e main project file for VC++ projects generated
=> pplication Wizard. It contains information
=> version of Visual C++ that generated the file,
=> ation about the platforms, configurations,
=> t features selected with the Application Wizard.
Для продолжения нажмите любую клавишу . . .
```

Рис. 2.5. **Вікно з результатами роботи програми до завдання 2.4**

Завдання 2.5. В останньому завданні цього практичного заняття проілюструємо деякі можливості, які досі не використовувалися.

1. Використаємо у програмі UNICODE-рядки, для цього у властивостях проекту не будемо міняти опцію Use Unicode Character Set.

2. Передамо параметр у програму при її запуску Project-> Properties->Configuration Properties -> Debugging -> Command Argument і напишемо там input.txt

3. Ознайомимося з функціями, які працюють з UNICODE-рядками. Вони подібні до звичайних, але відрізняються префіксом wcs, наприклад, strcpy (wcscpy), strcat (wcscat) і т. д.

Програма виводить на екран своє ім'я і параметр як UNICODE-рядки і перетворює ім'я у звичайний ASCII-рядок.

```
#include "stdafx.h"
#include <iostream>
#include <tchar.h>

using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
   char *fn=NULL;         // покажчик на рядок з ім'ям файла
        int flen;              // довжина імені файла
        wchar_t wct[500];      // UNICODE-рядок
   setlocale(0,"RUS");
        cout<<"Илюстрація використання UNICODE-рядків\n";

        flen=wcslen(argv[0])+1;    // довжина ім'я програми (0-параметр)
   fn=(char*)malloc(flen);    // виділяємо пям'ять під ім'я програми

        wcout << "wchar par(0): " <<argv[0] << endl;    // виводимо ім'я програми  як UNICODE-рядок
        wcout << "wchar par(1): " <<argv[1] << endl;    // виводимо перший параметр  як UNICODE-рядок
        wcstombs(fn, argv[0],wcslen(argv[0])+1);       // перетворюємо ім'я програми у звичайний ASCII-рядок

        cout << "  char par(0): "<< fn << endl<<endl;    // виводимо  звичайний ASCII-рядок
        cout << "Кількість параметрів до програми = "<< argc << endl;
        return 0;
}
```

Результати роботи програми представлені на рис. 2.6.

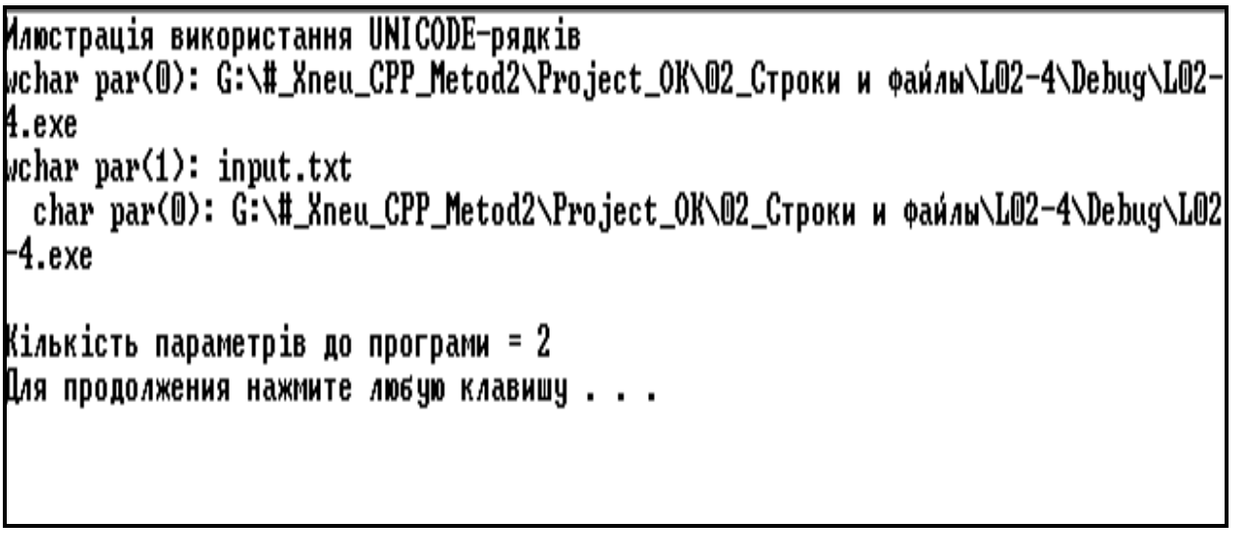

Рис. 2.6. **Вікно з результатами роботи програми до завдання 2.5**

Практичне заняття 3. Розробка програм обробки рядків з використанням макросів

Використання директив та макросів є настільки поширеною у мові C++, що навіть важко уявити собі програму, в якій би вони не використовувалися. Навіть розглядаючи найпростіші приклади [21], необхідно було використовувати, наприклад, директиву препроцесора #include, щоб підключити базові функції введення-виведення або стандартні математичні бібліотеки.

Успішне опанування цього матеріалу потребує вивчення таких питань:

1. Директиви препроцесора та їх призначення. Відмінність директиви #define від оператора typedef.

2. Умовна компіляція та засоби, які використовуються для цього.

3. Основні засоби роботи з макросами.

4. Засоби розробки багатофайлових проектів.

Спочатку розглянемо декілька прикладів створення макросів, а далі розглянемо програму, робота якої залежить від того, яким чином буде задана умовна компіляція.

Приклад 3.1. Задати в програмі розмір максимальної величини буфера масиву.

#define MAXLENGTH 100 // размер буфера

У подальшому можна використовувати визначення MAXLENGTH у програмі, вважаючи, що його значення дорівнює 100. Якщо виникне потреба змінити це значення, то це можна зробити тільки в директиві #define, не змінюючи код програми.

Приклад 3.2. Визначити макрос, який здійснює розрахунок квадрата числа.

#define square(x) (x)*(x)

Зазначимо, що визначення такого макросу у вигляді

#define square(x) x*x

є неправильним, бо, наприклад, якщо у програмі з'являється оператор **y=square(a+b)**, то він буде перетворений таким чином: **y=(a+b*a+b)**, а не **y=(a+b)*(a+b)**. Тому правильне використання дужок є дуже важливим моментом при визначенні макросів.

Приклад 3.3. При визначенні макросу слід особливу увагу звертати на недоречність використання додаткових пробілів, хоча у тексті програми вони в основному не відіграють особливої ролі.

#define BAD_PRINTX (x) printf("%d",x)

Додатковий пробіл між **PRINTX** та **(x)** призведе до того, що макрос **BAD_PRINTX** буде розкритий як **(x) printf("%d",x)**.

Правильне визначення повинне бути таким:

#define GOOD_PRINTX(x) printf("%d",x)

Існують дві операції, які дозволяється використовувати в директивах препроцесора: підстановка рядка (#) та конкатенація (##).

Якщо перед параметром макросу вказана операція підстановки символа (#), то компілятор при розкритті макросу замість параметра підставить його ім'я.

Приклад 3.4. Використання операції підстановки рядка у макросі.

#define PRINTVAL(val) printf(#val "=%d \n",(int)(val))

Замість #val підставляється ім’я параметра і оператор SHOWVAL(NN); видасть на екран рядок (допустимо, що змінна NN має значення 100):

NN = 100;

Операція конкатенації (або склеювання) викликає об’єднання двох рядків в один. Перед об’єднанням видаляються розділяючі їх пробіли. Наприклад [7]:

Приклад 3.5. Використання операції конкатенації рядків у макросі.

```
#define MACRO1  printf("Виклик MACRO_01\n");
#define MACRO2  printf("Виклик MACRO_02\n");
#define CREATE_MACRO(n) MACRO ## n
```

Якщо у програмі зустрічається оператор типу **CREATE_MACRO(1),** то він перетворюється у **MACRO1** і розкривається як

```
printf("Виклик MACRO1\n");
```

Визначити, які операції виконують такі макроси?

```
#define  PI  3.14159
#define  CIRCE_AREA(x)  (PI*square(x))
#define swap(x,y) {int tmp=(x);(x)=(y);(y)=tmp;}
#define WHAT_IS(x)     ((x)&(x-1))
```

Завдання 3.1. Написати програму, яка вводить рядок з клавіатури, пам’ять під який треба виділити динамічно. Здійснити контроль на допустиму довжину рядка та у випадку, якщо умова виконується, скопіювати рядок в інший і вивести на екран різними командами. Передбачити такі додаткові умови: визначення допустимої довжини рядка та функцій виведення й копіювання рядків здійснити за допомогую директив умовної компіляції; проект зробити багатофайловим, розмістивши функції копіювання й виведення в окремому файлі та створивши для нього свій заголовний файл.

```
// ТЕКСТ ОСНОВНОГО ФАЙЛА ////////////////////////////////
#include "stdafx.h"
#include <windows.h>
#include "mystr.h"
```

```
#define PROGRAM              // визначення макросу
#define MAXLENGTH   100 // розмір буфера

// задання максимальної довжини рядка  в залежно  від визначеної константи
#if   (L_VARIANT==1)
  #define MAXLEN 10   // допустима максим. довжина уводжуваного рядка
#elif (L_VARIANT==2)
  #define MAXLEN 20
#else
  #define MAXLEN 50
#endif

#if !defined(PROGRAM)
 #error Треба визначити константу PROGRAM
#endif

int main()
{
        // визначаємо символьні масиви для рядків
        char s1[MAXLEN+1]="", s2[MAXLEN+1]="";
        char *smax=NULL;    // покажчик на  рядок

        setlocale(0,"Rus");
        // виділення пам'яті під рядок
        smax=(char*) malloc(MAXLENGTH+1);
        // читання рядка з клавіатури
        printf("Введіть рядок без пропусків MAXLEN=%d : ",MAXLEN);
scanf("%s",smax);

        // перетворення рядка з урахуванням символів кирилиці
        OemToChar(smax,smax);
        if (strlen(smax)>MAXLEN){   // перевірка на допустиму довжину
                Vyvod("Недопустима довжина вхідного рядка!");
                return 0;
        }
```

```
   memcpy(s1,smax,strlen(smax));  // копіювання рядка smax в s1
        free(smax);                 // вивільнення пам'яті smax
        copystr(s2,s1);              // копіювання рядків за допомогою функції
        Vyvod(s1);                  // виведення  рядків на екран
        Vyvod(s2);
        return 0;
}

//  ТЕКСТ ФАЙЛА MYSTR.CPP З РОЗРОБЛЕНИМИ ФУНКЦІЯМИ

#include "stdafx.h"
#include "mystr.h"

// Виведення  рядка на екран різними командами
// залежно від визначення символічних констант у директивах
int Vyvod(char *s) {
 #ifdef IO_OUTPUT
        using namespace std;
        cout << "Виведення за допомогою cout" << endl;
        cout << s << endl;
 #else
        printf ("Виведення за допомогою printf \n %s \n",s);
 #endif
        return strlen(s);
}

// Повідомлення про спосіб копіювання рядків
void InfoStrCopy(int var) {
        printf("Do my strcpy (variant=%d) \n",var);
}

#ifndef LIB_STRCPY
#if (F_VARIANT==1)

// Копіювання одного рядка у інший - 1 варіант функції
char *mystrcpy(char*s, const  char*t) {
 while ((*s=*t)!=0) {s++; t++;};
```

```
  InfoStrCopy(F_VARIANT);
  return s;
}

#elif (F_VARIANT==2)
// Копіювання одного рядка у інший - 2 варіант функції
char *mystrcpy(char*s,  const  char*t) {
  while (*s++=*t++);
  InfoStrCopy(F_VARIANT);
  return s;
}

#else
// Копіювання одного рядка у інший - 3 варіант функції
char *mystrcpy(char s[], const  char t[]) {
  for (int i=0; (s[i]=t[i])!=0; i++) ;
  InfoStrCopy(F_VARIANT);
  return s;
}

#endif // для F_VARIANT
#else
 // Копіювання одного рядка у інший стандартною бібліотечною функцією
#endif // для LIB_STRCPY


// ЗАГОЛОВНИЙ ФАЙЛ ДЛЯ  РОЗРОБЛЕНИХ ФУНКЦІЙ
#ifndef __MYSTR__H
#define __MYSTR_H

#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <iostream>
#include <string.h>

//#define LIB_STRCPY      // використати бібліотечну функцію
```

```
//#define IO_OUTPUT       // спосіб виведення на екран (через б-ку IOS-
TREAM, якщо задана)
#define L_VARIANT    1  // варіант максимальної допустимої довжини ряд-
ка
#define F_VARIANT    2  // варіант функції копіювання рядків

#ifdef LIB_STRCPY
  #define copystr strcpy
#else
  #define copystr mystrcpy
#endif

// Виведення  рядка на екран
int Vyvod(char *s);

// Копіювання одного рядка в інший
char *mystrcpy(char*s, const char*t);

#endif
```

У результаті виконання програми на екрані з'явиться подібний результат, (рис. 3.1):

```
Введіть рядок без пропусків MAXLEN=10 : Груша_Good
Do my strcpy (variant=2)
Вивід за допомогою printf
 Груша_Good
Вивід за допомогою printf
 Груша_Good
Для продолжения нажмите любую клавишу . . .
```

Рис. 3.1. **Вікно з результатами роботи програми до завдання 3.1**

Як самостійне завдання необхідно проекспеременувати з виконанням програми, компілюючи її з різними умовами. У разі появи помилок виправити їх. Пояснити отримані результати.

Практичне заняття 4. Розробка програм обробки об'єднань та масивів структур

Структури становлять собою складені типи даних, побудовані з використанням інших типів. Вони об'єднують загальним ім'ям набір даних різних типів. Саме тим, що в структурах можуть зберігатися дані різних типів, вони і відрізняються від масивів, що зберігають дані одного типу. Окремі дані структури називаються *елементами* або *полями*.

Крім структур у С++ існує особлива конструкція, об’ява якої нагадує синтаксис структур, але вона має зовсім інший фізичний сенс – це об’єднання. Об’єднання використовується для розміщення в одній і тій же області пам’яті (за однією і тією ж адресою) даних різних типів.

Для того щоб досконально розібратися з теоретичним матеріалом і зрозуміти наведений приклад, необхідно дати відповіді на такі запитання:

1. Що таке структура та основні правила роботи із структурними типами даних?

2. Як описувати нові типи і створювати відповідні екземпляри структур; як здійснювати доступ до елементів даних (полів) структури?

3. Що таке масиви структур, покажчики на структури? Як здійснювати передачу по посиланню масивів структур?

4. Що таке об'єднання? Як описувати та працювати з об'єднаннями?

Завдання 4.1. Написати діалогову програму, яка створює динамічний масив структур з даними про студентів, вводить у нього дані про успішність навчання та пропуски занять й дозволяє отримувати відповіді на запитання про кращого студента курсу, загальний середній бал та здійснювати пошук даних студента за прізвищем. В якості додаткової функції показати на екрані, яким чином зберігаються дані у структурі.

```
// Приклад додатка з використанням структур та об'єднань
#include "stdafx.h"
#include <iostream>
```

```
#include <string.h>
#include <locale.h>
#include <windows.h>
#define MAX_STUD     100     // Максимальна кількість студентів (розмір масиву)
#define MAX_SCROLL_LINE   5     // Максимально допустима кількість рядків  для "прокрутки" екрана
#define NO_GREATER(x,y)    ((x)<=(y)?(x):(y))   // макрос порівняння "НЕ БіЛЬШЕ"

using namespace::std;
typedef unsigned char byte;     // визначення нового типу даних <byte>
// Визначення структури для збереження даних про студента
struct Stud {
        char fio[20];    // Прізвище І.Б.
        byte kurs;       // Курс, на якому навчається
};
// Визначення структури для обліку успішності навчання студента
struct Study {
        Stud  student;    // Дані про студента
        float s_ocenka;   // Середній бал
        int   propuski;   // Кількість пропусків занять
};
// Об'єднання: Масив символів та Структура даних про студента
union UStudy {
        unsigned char mem [30];
        Study        stud;
};
// Прокрутка екрана на n-рядків
void ScrollNLine(int n)
{
  n = NO_GREATER(n,MAX_SCROLL_LINE);  // на скільки рядків здвигаємо екран
  for (int i=0; i<n; i++) cout<<endl; // цикл виведення символа "новий рядок"
}

// Введення початкових даних у масив структур
```

```
// data - масив даних студента, cnt - кількість студентів
void Vvod(Study *data, int cnt)
{
        for (int i=0; i<cnt; i++) {
                cout << "*** Студент № " << i+1 << " ***" << endl;
                cout << "Прізвище І.Б. : "; cin >> (data+i)->student.fio;

        // Перетворення введеного рядка з правильними кодами символів
                OemToChar((data+i)->student.fio,(data+i)->student.fio);

                cout << "Курс         : "; cin >> (data+i)->student.kurs;
                cout << "Середній бал  : "; cin >> (data+i)->s_ocenka;
                cout << "Пропуски      : "; cin >> (data+i)->propuski;
                cout << endl;
        }
}

// Пошук кращого студента на курсі
// data - масив даних студента, cnt - кількість студентів
void  FindBestStudent(Study *data, int cnt)
{
 byte nkurs;
 UStudy ubuf;   // об'єднання для виведення способу зберігання даних
структури

 // ініціалізація  пошукових даних
 float gmark =0;     // краща оцінка
 int   gnomer=-1;    // номер (індек у масиві) кращого студента

 // Вводимо номер курсу, для якого шукаємо кращого студента
 cout << "Кращий студент для курсу? ";  cin >> nkurs;
 for (int i=0; i<cnt; i++) {
        if (data[i].student.kurs!=nkurs) continue;  // не аналізуємо інші курси
        // Знайшли кращого?
        if (data[i].s_ocenka > gmark)
          { gnomer=i; gmark=data[i].s_ocenka; }    // запам'ятовуємо кращого
 }
```

```
  // Чи знайшли кого-небудь?
  if (gnomer>=0){
          // Виводимо дані про знайденого студента
    cout << "Прізвище І.Б. - " <<data[gnomer].student.fio  << endl;
    cout << "Оцінка       - " <<data[gnomer].s_ocenka  << endl<<endl;

        cout << "Структура займає в пам'яті: "<<sizeof(Study) << "байт \n";
        cout << "Шістнадцятирічне представлення наступне: ";
        ubuf.stud = data[gnomer];
        for (int i=0; i<sizeof(Study); i++) printf("%02X ",ubuf.mem[i]);

  } else {
    cout << "Даних немає" << endl;
  }
}

// Пошук студента за прізвищем
void  FindStudent(Study *data, int cnt)
{       char fio[20];

   // Вводимо прізвище
        cout << "Введіть ПІБ студента: "; cin>>fio;
        // Організуємо перегляд масиву з даними
        for(int i=0; i<cnt; i++) {
                if (strcmp(fio,(data+i)->student.fio)) continue;       // прізвища не
співпадають
                // Виводимо дані про студента на екран
                cout << "Курс    : " <<data[i].student.kurs  << endl;
                cout << "Оцінка  : " <<data[i].s_ocenka      << endl;
                cout << "Пропуски: " <<data[i].propuski      << endl;
       return;
        }
        // Нікого не знайшли
        cout << "Такий студент відсутній" << endl;
}
// Визначення загального середнього бала успішності
```

```
// data - масив даних студента, cnt - кількість студентів
float AverageMark(Study data[], int cnt)
{
  // ініціалізація початкових даних
  float smark=0;   // середній бал
  int i=0;

  while(true) {
        smark += (data+i)->s_ocenka;   // накопичуємо суму балів
        if (++i>=cnt) break;                // перевірка на допустиму кількість
студентів
  }
  return smark/cnt;                // розраховуємо та повертаємо середній бал
успішності
}

// Управляюча процедура вирішення задачі (з меню)
void Obrabotka(Study *data, int cnt)
{
 bool cyc=true;  // ознака повторення основного циклу
 char ch;       // значення вибраного пункту меню

 // Основний цикл виконання програми з виведенням  меню
 while (cyc) {
        ScrollNLine(2);   // "прокрутка" екрана на 2 рядки
        // Виводимо пункти основного меню
        cout <<"*****************************"<<endl;
        cout <<"1 - Кращий студент курсу   " <<endl;
        cout <<"2 - Пошук даних  студента  "<<endl;
        cout <<"3 - Загальний середній бал  "  <<endl;
        cout <<"0 - Вихід"<<endl;
        cout <<"*****************************"<<endl;
        cin >> ch;      // Читаємо,  який пункт вибрали

        // Розгалудження програми залежно від вибраного пункту меню
  switch (ch) {
     case '1' : FindBestStudent(data, cnt); break;    // Пошук кращого студента
```

```
курсу
        case '2' : FindStudent(data, cnt);     break;   // Пошук даних  про студента
        case '3' : cout << " Загальний середній бал  = " <<       // Підрахунок  за-
гального середнього бала
                    AverageMark(data, cnt)<< endl;
                     break;
         case '0' : cyc=false;  break;        // Встановлюємо ознаку "вихід з циклу"
                  default  : ;
          } // switch

  }  // while
}

// Головна програма
int main(int argc, char* argv[])
{
  int N=0;
  Study *stdata;     // Покажчик на масив студентів
  setlocale(0,"Rus");

  // Вводимо кількіть студентів
  cout << "Введіть кількість студентів (але не більше ніж " << MAX_STUD
<< ") ";
  cin >> N;

  if (N>MAX_STUD){     // контроль правильності введення
          cout << "Ви ввели неправильне  значення!" << endl;
          return 0;
  }
  stdata = new Study[N];   // виділяємо пям'ять під масив
  Vvod(stdata,N);          // вводимо дані про студентів
  Obrabotka(stdata,N);     // здійснюємо обробку даних за вибором пункту
меню
  delete[] stdata;         // видаляємо динамічний масив
   return 0;
}
```

Результати роботи програми представлені на рис. 4.1.

```
Введіть кількість студентів (але не більше ніж 100) 3
*** Студент № 1 ***
Призвище і.Б. : Петров
Курс          : 1
Середній бал  : 3.5
Пропуски      : 20

*** Студент № 2 ***
Призвище і.Б. : Сидоров
Курс          : 2
Середній бал  : 4.1
Пропуски      : 15

*** Студент № 3 ***
Призвище і.Б. : Коробов
Курс          : 2
Середній бал  : 4
Пропуски      : 12



*****************************
1 - Кращий студент курсу
2 - Пошук даних  студента
3 - Загальний середній бал
0 - Вихід
*****************************
1
Кращий студент для курсу? 2
Призвище і.Б. - Сидоров
Оцінка        - 4.1

Структура займає в пам'яті: 32байт
Шістнадцятирічне представлення наступне: D1 E8 E4 EE F0 EE E2 00 CD CD CD CD CD
CD CD CD CD CD CD CD 32 CD CD CD 33 33 83 40 0F 00 00 00

*****************************
1 - Кращий студент курсу
2 - Пошук даних  студента
3 - Загальний середній бал
0 - Вихід
*****************************
```

Рис. 4.1. **Вікно з результатами роботи програми до завдання 4.1**

Поліпшіть програму таким чином, щоб вона:

1. Виконувала додаткові розрахунки на існуючих даних.
2. Зберігала введені дані у файлі та завантажувала їх звідти.
3. Проводила аналіз зв'язку між успішністю навчання та кількістю годин пропуску занять.

Практичне заняття 5. Програми роботи з контейнерами з використанням STL

STL – це бібліотека стандартних шаблонів, мета якої звільнити програміста від виконання рутинних, загальноприйнятих, часто використовуваних операцій. Ця бібліотека контейнерних класів містить

алгоритми обробки таких способів організації даних, як динамічні масиви, двонаправлені списки, стеки, черги та ін. Крім того, STL містить безліч алгоритмів що часто зустрічаються: сортування (як на всій безлічі, так і на частині), знаходження мінімального й максимального значень та ін. Кожен такий алгоритм працює з різними типами контейнерів.

Для поглибленого засвоєння матеріалу необхідно чітко знати відповіді на такі запитання:

1. Що таке шаблони функцій та класів? Чим вони відрізняються?
2. Яким чином здійснюється перевантаження шаблонів функцій?
3. Що таке стандартна бібліотека шаблонів (STL)? Призначення та склад STL. Контейнери. Робота з векторами, списками, стеками, чергами і т. д.

Завдання 5.1. Написати програму, яка працює з двома функціями, заданими своїми шаблонами. Перша – обчислює середнє значення двох чисел і змінює їх місцями. Друга виводить на екран значення двох довільних змінних у поле заданої ширини та визначає тип цих змінних.

```
// Приклад програми, яка працює з шаблонами функцій

#include "stdafx.h"
#include <iostream>
#include < typeinfo>
#include <string.h>

using namespace std;

// Визначення шаблону функції, яка виконує обмін
// двох числових змінних та повертає середнє їх значення.
template <class T>          // тип даних, з якими працює функція
T AvgSwap(T &var1, T &var2)  // функція приймає дві змінні якогось чи-
слового типу даних
{  T tmp;
   // здійснення обміну змінними
   tmp = var1;    var1 = var2;   var2 = tmp;
   // повернути середнє значення
```

```
  return (var1+var2)/2;
}
// Визначення шаблону функції, яка приймає 4 параметри:
// 1  - текст, що виводиться на екран
// 2,3- два параметри задовільного типу
// 4  - ширина поля, у який необхідно вивести значення 2 та 3 параметрів
// Додатково функція виводить типи параметрів
template <typename T1, typename T2>
void Print(char *text, T1 var1, T2 var2, int len)
{
  cout << text << ": ";                  // виводимо текст
  // виводимо значення 2 параметра в поле завширшки len-позицій
  cout.width (len);  cout <<"1) " <<var1 << " "<<typeid(var1).name();
  // виводимо значення 3 параметра в поле завширшки len-позицій
  cout.width (len);  cout <<"  2) " <<var2 << " "<<typeid(var2).name()<< endl;
  //cout <<"\n";
}
// Головна програма
int main()
{  int    iA=5, iB=9;                  // змінні цілого типу
   double dA=5.01, dB=8.17, dC=12.77;     // змінні дійсного типу
   char A[10]="table", B[10]="the";       // змінні текстового типу (рядки)
//   char s[]="Test text";

   // Виведення на екран значень двох цілих змінних - крок № 1
   Print("Integer before Swap", iA,iB,5);
   // Обмін змінними - крок №2
   cout << "Average= "<<AvgSwap(iA,iB) << endl;
   // Повторне виведення на екран значень двох цілих змінних - крок № 3
   Print("Integer after Swap ", iA,iB,5);

   // Виконання кроків №1,2,3 для дійсних змінних
   Print("\nDouble before Swap", dA,dB,10);
   cout << AvgSwap(dA,dB) << endl;
   Print("Double after Swap ", dA,dB,11);

   // Виведення на екран значень двох рядків
```

```
  Print("\nPrint Char", A,B,7);
  // Виведення на екран значень двох змінних різних типів
  Print("\nPrint Different type", A,dA,10);
  return 0;
}
```

Результати роботи програми представлені на рис. 5.1.

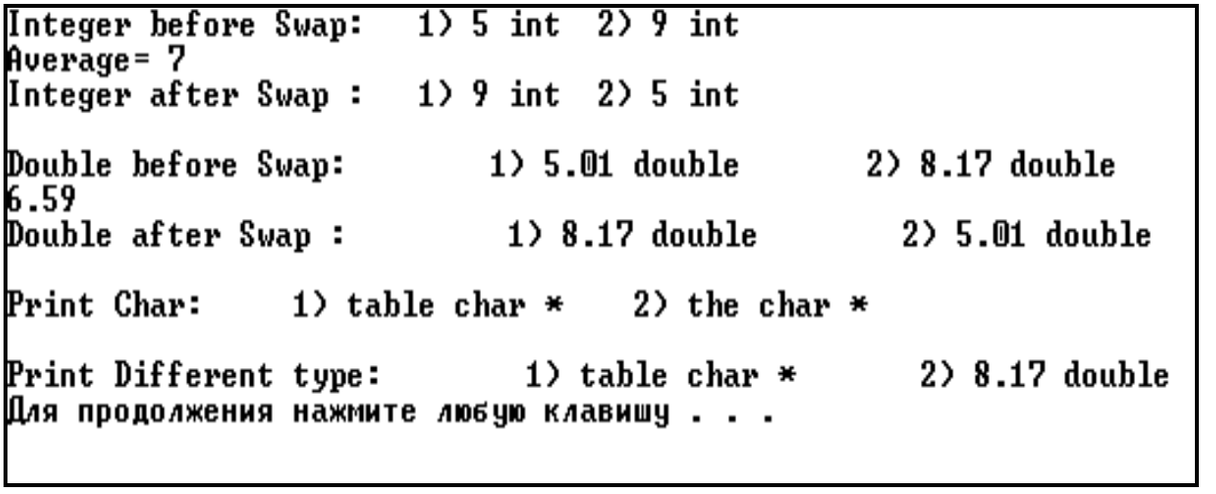

Рис. 5.1. **Вікно з результатами роботи програми до завдання 5.1**

Проаналізувати текст програми та визначити:

1. Чи завжди правильно працює програма? Якщо "ні", то що треба зробити, щоб позбутися помилок?

2. Чи можна використовувати шаблон функції AvgSwap для символьних масивів (рядків)? Відповідь аргументувати.

Завдання 5.2. Написати діалогову програму, яка дозволяє вводити та зберігати дані у файлі про продані та наявні товари. Використати у програмі стандартні контейнери бібліотеки STL вектор та список.

Примітка: у програмі використовується упаковка та розпаковка дати в одне число цілого типу. Доведіть правильність чи помилковість такої операції.

```
// Приклад програми, що використовує бібліотеку шаблонів STL

#include "stdafx.h"
#include <locale.h>
#include <stdio.h>
```

```
#include <vector>
#include <list>
// заборона виведення  попередження 4996
#pragma warning(disable: 4996)
using namespace std;

// Структура опису товару, що надійшов
typedef struct
{
  char name[15];    // Назва
  char marka[10];   // Марка
  float cena;          // Ціна
} GOODS;
// Структура опису інформації про проданий товар
typedef struct
{
  char name[15];    // Назва
  char marka[10];   // Марка
  long sdate;          // Дата продажу
} SELLS;

list <GOODS>  tovar;       // Список товарів у наявності
vector<SELLS> prodano;     // Дані про продані товари

//////////////////////////////////////////////////
// Функція порівняння елементів типу GOODS (за ціною)
bool operator<(const GOODS &x, const GOODS &y)
{
        return x.cena < y.cena;
}
//////////////////////////////////////////////////

// Виведення діагностичного повідомлення
void ShowMessage(char *mes)
{
        printf("!!! %s !!!\n",mes);
}
```

```
// Надходження товару до магазину
void ToShop()
{   GOODS ng;
        printf("Введіть дані про новий товар за форматом \n");
        printf("Назва    Марка Ціна : ");
        scanf("%s %s %f",&ng.name, &ng.marka, &ng.cena);
        tovar.push_back(ng);
}
// Продаж товару
void OutShop()
{   char marka[10]="";   // Змінна  для збереження марки товару
   long date=0;             // Змінна для збереження дати продажу
        SELLS sg;          // Змінна  для зберігання даних про продаж товару

        printf("Марка проданого товару и дата (РРММДД): ");
        scanf("%s %ld",&marka, &date);

        // Цикл перегляду наявних товарів
        for( list<GOODS>::iterator i=tovar.begin(); i!=tovar.end(); i++) {
                if (strcmp(i->marka,marka)) continue;    // порівняння марки то-
вару
                strcpy(sg.name,i->name);                // записуємо дані про продаж
                strcpy(sg.marka,i->marka);
                sg.sdate = date;
      prodano.push_back(sg);       // Заносимо у продаж
                tovar.erase(i);          // Видаляємо зі списку, що є у наявності
                return;
        }
        ShowMessage("Такий товар відсутній");
}
// Виведення даних про наявні товари
void GoodsInShop()
{
        tovar.sort();   // Впорядковуємо за ціною
        printf("\n /// СПИСОК ТОВАРІВ У НАЯВНОСТІ ///\n");
        printf("-----------------------------------\n");
        printf("|   Назва      | Марка    |   Ціна |\n");
```

```
        printf("----------------------------------\n");
        // Цикл перегляду наявних товарів
        for ( list<GOODS>::iterator i=tovar.begin(); i!=tovar.end(); i++) {
                printf("|%-15s|%-10s|%8.2f|\n",i->name,i->marka,i->cena );
        }
        printf("----------------------------------\n");
}

// Виведення даних про продані  товари
void SaleInfo()
{   int day, month, year;
        printf("\n ///  СПИСОК ПРОДАНИХ  ТОВАРІВ  ///\n");
        printf("----------------------------------\n");
        printf("|   Назва       | Марка    |  Дата  |\n");
        printf("----------------------------------\n");

        // Цикл перегляду проданих товарів
        for ( vector<SELLS>::iterator i=prodano.begin(); i!=prodano.end(); i++) {
      year =i->sdate/10000;
                month=(i->sdate-year*10000)/100;
                day  =i->sdate % 100;
                printf("|%-15s|%-10s|%02d/%02d/%02d|\n",i->name,i-
>marka,day,month,year);
        }
        printf("----------------------------------\n");
}
// Збереження даних у файлі
void SaveData(char *fname)
{ FILE *fn;
  // Відкриваємо файл для запису
  if ((fn=fopen(fname,"wt"))==NULL) return;

  // Збереження даних про наявні товари
  for (list<GOODS>::iterator i=tovar.begin(); i!=tovar.end(); i++) {
          fprintf(fn,"%s %s %f\n",i->name, i->marka, i->cena);
  }
```

```
  // Виведення  розділового рядка
  fprintf(fn,"//\n");
  // Збереження даних про продаж
  for ( vector<SELLS>::iterator i=prodano.begin(); i!=prodano.end(); i++) {
     fprintf(fn,"%s %s %ld\n",i->name,i->marka,i->sdate);
  }
  fclose(fn);
}
// Відновлення збережених даних
void RestoreData(char *fname)
{ FILE *fn;
  char s[80];
  GOODS ng;
  SELLS sg;
    // Відкриваємо файл для зчитування
  if ((fn=fopen(fname,"rt"))==NULL) return;

  //********* Для самостійного знаходження помилки *************
   tovar.clear() ;
   prodano.clear();
  //************************************************************
  // Відновлення даних про товари у наявності
  while (!feof(fn)){
    fgets(s,sizeof(s),fn);                  // читаємо рядок
         if (!strncmp(s,"//",2)) break;  // дійшли до рядка розподілу
         // розбираємо рядок на складові
    sscanf(s,"%s %s %f",&ng.name, &ng.marka, &ng.cena);
        tovar.push_back(ng);            // заносимо у список
  }
  // Відновлення даних про продажі
  while (!feof(fn)){
    if (fgets(s,sizeof(s),fn)==NULL) break; // читаємо рядок
          // розбираємо рядок на складові
         sscanf(s,"%s %s %ld",&sg.name, &sg.marka, &sg.sdate);
         prodano.push_back(sg);                // заносимо у вектор
  }
  fclose(fn);
```

```
}

// Вибір пунктів меню
int ChoiceMenu()
{   int nom=0;
   // Виведення меню
        printf("*****************************************\n");
        printf("*** 1 - Надходження нового товару   ***\n");
        printf("*** 2 - Продаж товару              ***\n");
        printf("*** 3 - Відомості про наявні товари ***\n");
        printf("*** 4 - Відомості про продаж        ***\n");
        printf("*** 5 - Зберегти дані               ***\n");
        printf("*** 6 - Відновити дані              ***\n");
        printf("*** 0 - Вихід                       ***\n");
        printf("*****************************************\n");
        scanf ("%d",&nom);  // вибір пункту меню
        return nom;
}
// Головна програма
int main()
{  bool cyc=true;            // Змінна циклу
  int vybor=0;              // Значення вибраного пункту меню
  char filename[]="data.db";  // ім'я файла зі збереженими даними
  setlocale(0,"Rus");
  while (cyc) {                                  // цикл вибору пунктів меню
          vybor=ChoiceMenu();                    // вибір пункту меню
          switch(vybor) {
                  case 1: ToShop(); break;           // Надходження нового товару
                  case 2: OutShop(); break;        // Продаж товару
                  case 3: GoodsInShop(); break;    // Відомості про наявні товари
                  case 4: SaleInfo(); break;         // Відомості про продаж
                  case 5: SaveData(filename); break;          // Зберегти дані
                  case 6: RestoreData(filename); break; // Відновити дані
                  case 0: cyc=false;                                  // Вихід
                  default :;
          } // switch
  } // while
```

```
  return 0;
}
```

Результати роботи програми представлені на рис. 5.2.

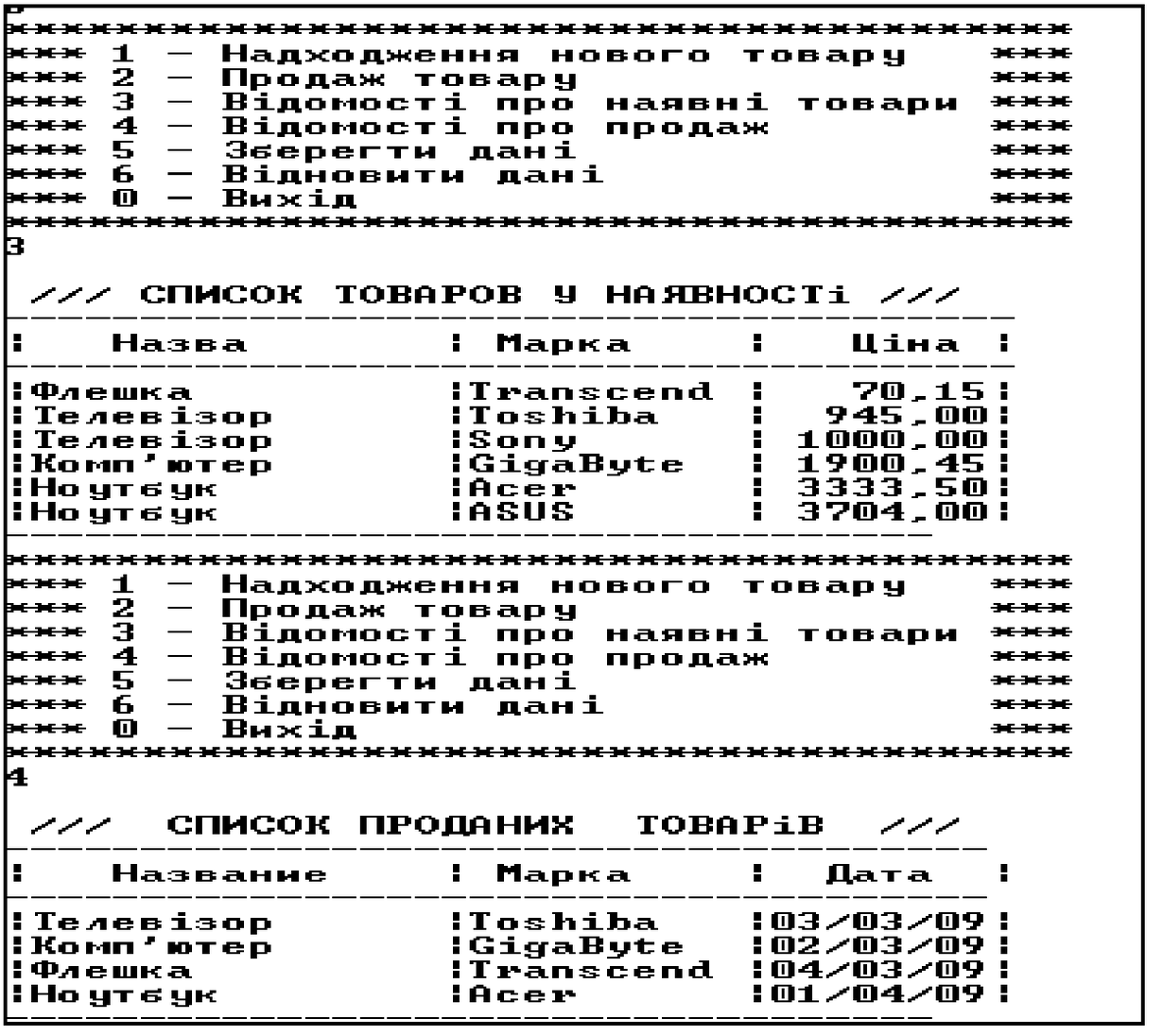

Рис. 5.2. **Вікно з результатами роботи програми до завдання 5.2**

Наведемо ще один приклад роботи з контейнерами, але вже іншого, асоціативного типу. Такі контейнери були розроблені для швидкого доступу до елементів за допомогою ключів. До таких контейнерів відноситься <map>, тобто карта.

Завдання 5.3. Розробити програму, яка ілюструє роботу з контейнерами типу <map>, з ключами та значеннями різного типу, а саме: <ціле число, ціле число>, <ціле чісло, рядок>, <рядок, рядок>.

```
#include "stdafx.h"
```

```
#include <locale.h>
#include <string>
#include <map>
#include <iostream>
#include <windows.h>
using namespace std;

// Вводимо нові типи даних для контейнерів <map>
typedef pair<int,int> Int_Pair;
typedef pair<int,string> Int_Str;
typedef pair<string,string> Str_Str;

int main( )
{
  setlocale(0,"RUS");
  // змінна контейнера <map> з цілими числами
  map <int, int> m1;
  // ітератор для контейнера m1
  map <int, int>::iterator m1_pIter;

  // змінна контейнера <map> з цілим числом та рядком
  map <int, string> m2;

  // ітератор для контейнера m2
  map <int, string>::iterator m2_pIter;

  // змінна контейнера <map> з рядками
  map <string, string> m3;
  // ітератор для контейнера m3
  map <string, string>::iterator m3_pIter;

  // Заносимо дані в карту m1
  m1.insert ( Int_Pair ( 1, 10 ) );
  m1.insert ( Int_Pair ( 2, 20 ) );
```

```
m1.insert ( Int_Pair ( 3, 30 ) );
m1.insert ( Int_Pair ( 4, 40 ) );
cout << "Перелік ключів контейнера m1..... = ";
for ( m1_pIter = m1.begin( ); m1_pIter != m1.end( ); m1_pIter++ )
cout << " " << m1_pIter -> first;
cout << "." << endl;

cout << "Перелік значень для ключів контейнера m1 =";
for ( m1_pIter = m1.begin(); m1_pIter != m1.end(); m1_pIter++ )
   cout << " " << m1_pIter -> second;

cout << "." << endl << endl;

// Нова карта для перевірки введення  даних
pair< map<int,int>::iterator, bool > pr;
// Спроба внести нове значення у карту
pr = m1.insert ( Int_Pair ( 1, 10 ) );

// Перевірка успішності
if( pr.second == true ) {
  cout << "Елемент 10 був успішно добавлений у m1." << endl;
}
else {
  cout << "Ключ з номером 1 вже існує у m1\n"
       << "та пов'язаний зі значенням= "
       << ( pr.first ) -> second
       << "." << endl << endl;
}

// Заносимо дані в карту m2
m2.insert ( Int_Str ( 1, "монітор" ) );
m2.insert ( Int_Str ( 2, "флешка" ) );
m2.insert ( Int_Str ( 3, "клавіатура" ) );
```

```
  cout << "Перелік ключів контейнера m2 у зворотному напрямі = \n";
   m2_pIter = --m2.end();
   while (true) {
     cout << " Ключ= " << m2_pIter->first << "  Значення= "<< m2_pIter-
>second<<endl;

          if (m2_pIter == m2.begin()) break;
          m2_pIter--;
   };
   cout << endl << endl;

   // Заносимо дані в карту m3
   m3.insert ( Str_Str ( "робота", "work" ));
   m3.insert ( Str_Str ( "яблуко", "apple" ));
   m3.insert ( Str_Str ( "ручка", "pen" ));
   m3.insert ( Str_Str ( "лампа", "lamp" ));
   m3.insert ( Str_Str ( "годинник", "clock" ));
   m3.insert ( Str_Str ( "собака", "dog" ));

   cout << "Українсько-англійський словник (для закінчення введіть 0)\n";
   string slovo;
   char   nslovo[80];
   while (true) {
          cout << "Введіть слово : "; cin >> slovo;
          if (slovo=="0") break;
          OemToChar(slovo.c_str(), nslovo);

          m3_pIter = m3.find(nslovo);
          if (m3_pIter != m3.end()) cout << "  Переклад : "<< m3_pIter-
>second<<endl;
          else cout << "  слово відсутнє \n";
   };
}
```

Результати роботи програми представлені на рис. 5.3.

```
Перелік ключів контейнера m1..... =  1 2 3 4.
Перелік значень для ключів контейнера m1 = 10 20 30 40.

Ключ з номером 1 вже існує у m1
та пов'язаний зі значенням= 10.

Перелік ключів контейнера m2 зворотньому напрямку =
 Ключ= 3  Значення= клавіатура
 Ключ= 2  Значення= флешка
 Ключ= 1  Значення= монітор


Українсько-англійський словник (для закінчення введіть 0)
Введіть слово : яблуко
  Переклад : apple
Введіть слово : собака
  Переклад : dog
Введіть слово : вода
  слово відсутне
Введіть слово : ручка
  Переклад : pen
Введіть слово : 0
Для продолжения нажмите любую клавишу . . .
```

Рис. 5.3. **Вікно з результатами роботи програми до завдання 5.3**

Практичне заняття 6. Розробка Windows-додатків

При написанні коду програми для Windows вона підпорядковується операційній системі і Windows керує нею. Програма для Windows не повинна безпосередньо взаємодіяти з обладнанням, і всі комунікації із зовнішнім світом повинні проходити через Windows. При використанні Windows-програми, користувач спочатку взаємодіє з Windows, а вона вже взаємодіє з конкретною прикладною програмою. Перша і найважливіша причина такого стану речей полягає в тому, що Windows-програма майже завжди поділяє комп'ютер з іншими програмами, які

можуть виконуватися в один і той же час. Windows постійно повинна зберігати контроль над розподільними ресурсами машини. Друга причина для збереження контролю за Windows полягає в тому, що Windows містить в собі стандартний користувальницький інтерфейс і має відповідати за дотримання стандарту. Програма може відображати інформацію на екрані, тільки використовуючи інструменти, передбачені в Windows, і тільки тоді, коли це дозволено Windows.

На відміну від раніше розглянутих програм, Windows-програми є керованими подіями, тому така програма, по суті, постійно очікує, коли що-небудь відбудеться. Значна частина коду, необхідного Windows-додатка, призначена для обробки подій, викликаних зовнішніми діями користувача. Внаслідок цього, написання подібного роду програм є досить специфічним і вимагає ґрунтовної теоретичної і практичної підготовки. Розробник Windows-програм повинен чітко уявляти собі відповіді на наступні запитання:

1. Особливості ОС Windows. Графічні елементи вікна Windows. Елементи призначеного для користувача інтерфейсу Windows. Поняття повідомлення. Джерела отримання повідомлень у Windows-програмах. Цикл обробки повідомлень. Графічна схема типової Windows-програми.

2. Структура каркасного Windows-додатка. Класи вікон, їх реєстрація та функції підтримки вікон.

3. Ресурси Windows-додатків. Структура файлів ресурсу. Підключення ресурсів до виконавчого файла.

4. Акселератори. Опис акселераторів у файлах ресурсів. Підключення таблиць акселераторів.

5. Діалогові вікна та їх елементи. Модальні та немодальні діалогові вікна. Клавіатурний інтерфейс діалогового вікна. Елементи управління діалогових вікон. Створення діалогових вікон.

6. Процеси та потоки. Таблиця процесів. Атрибути процесу. Сервіси процесу. Стан процесу. Атрибути потоку. Створення та завершення потоку. Волокна. Планування процесів і потоків. Системи пакетної обробки. Інтерактивні системи. Системи дійсного часу. Система пріоритетів. Засоби міжпроцесного обміну даними. Буфери обміну. Атоми. Канали передачі даних. Роздільна пам’ять.

Особливо слід звернути увагу на типи даних, які використовуються при написанні Windows-програм (додаток В).

Завдання 6.1. Розробити програму каркасного Windows-додатка, яка крім своїх базових функцій включає додаткові:

1. Перевірку на те, що програма вже запущена.

2. У зафарбованому вікні виводить три рядки тексту за різними координатами.

```
// Найпростіша Windows-програма для відображення тексту у вікні
#include <windows.h>
#include <stdafx.h>

// Прототип функції обробки повідомлень вікна
LRESULT WINAPI WindowProc(HWND hWnd, UINT message,WPARAM
wParam, LPARAM lParam);


//////////////////////////////////////////////////////////////////////////////
//              Головна програма WinMain
// Параметри:
// HINSTANCE hInstance     - Дескриптор примірника програми, що
виконується
// HINSTANCE hPrevInstance  -  Дескриптор перед. примірника програми  (
в Win32 = NULL)
// LPSTR lpCmdLine     - покажчик на командний рядок
// int nCmdShow           - зовнішній вигляд вікна при його створенні
//////////////////////////////////////////////////////////////////////////////

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
            LPSTR lpCmdLine, int nCmdShow)
{
  // Структура для зберігання атрибутів вікна
  WNDCLASSEX WindowClass;
  static LPCTSTR szAppName = "WinLab6_01";  // Визначити клас вікна
  HWND hWnd;                          // Дескриптор вікна
  MSG msg;                           // Структура повідомлення вікна

// HANDLE hevt = CreateEvent(NULL, FALSE, FALSE,
```

```
TEXT("ProgName_started"));
 HANDLE hevt = CreateEvent(NULL, FALSE, FALSE,szAppName);

if (GetLastError() != ERROR_ALREADY_EXISTS) {
// Перший раз запустили?
} else {
// Вже існує (запущена)
        MessageBox(NULL,"Програма вже запущена","",0);
        return 0;
}

  RECT screen_rect;
  GetWindowRect(GetDesktopWindow(),&screen_rect); // розподільна
здатність екрана

  WindowClass.cbSize = sizeof(WNDCLASSEX);      // Встановити розмір
структури
  WindowClass.style  = CS_HREDRAW | CS_VREDRAW; // Перемалювати
вікно при зміні розміру
  WindowClass.lpfnWndProc = WindowProc;        // Визначити функцію-
обробник повідомлень
  WindowClass.cbClsExtra = 0;   // Немає додаткових байтів  після структури
  WindowClass.cbWndExtra = 0;           // класу вікна чи примірника вікна
  WindowClass.hInstance = hInstance;    // Дескриптор екземпляру додатка

  WindowClass.hIcon = LoadIcon(0, IDI_APPLICATION); // Встановити
піктограму додатка  за замовчуванням
  WindowClass.hCursor = LoadCursor(0, IDC_ARROW);   // Встановити
стандартний курсор миші  у вигляді стрілочки

  // Встановити "сіру" кисть для кольору фону
  WindowClass.hbrBackground = stat-
ic_cast<HBRUSH>(GetStockObject(GRAY_BRUSH));

  WindowClass.lpszMenuName = 0;            // Немає  меню
  WindowClass.lpszClassName = szAppName;    // Встановити ім'я класу
  WindowClass.hIconSm = 0;            // Маленька піктограма за замовчуваням
```

```
 RegisterClassEx(&WindowClass);          // Зареєструвати клас вікна
 //  Створити вікно
 hWnd = CreateWindow(
       szAppName,                    // Ім'я класу вікна
       "Приклад простої Windows-програми",   // Заголовок вікна
       WS_OVERLAPPEDWINDOW,    // Стиль вікна,  що перекривається
       CW_USEDEFAULT,        // Позиція на екрані за замовчуванням ...
       CW_USEDEFAULT,        // лівого верхнього кута як x, y. ..
       CW_USEDEFAULT,        // Розмір вікна за замовчуванням - ширина
       CW_USEDEFAULT,        // .... і  висота
       0,                // Немає батьківського вікна
       0,                // Немає меню
       hInstance,       // Дескриптор примірника програми
       0                 // Ніяких даних для створення вікна ( додат. параметри)
     );

 ShowWindow(hWnd, nCmdShow);     // Відобразити вікно
 UpdateWindow(hWnd);          // Перемалювати клієнтську область вікна

 // Цикл обробки повідомлень
 while(GetMessage(&msg, 0, 0, 0) == TRUE)  // Отримати повідомлення
 {
  TranslateMessage(&msg);             // Трансляція повідомлення
  DispatchMessage(&msg);              // Диспетчеризація повідомлення
 }

 return static_cast<int>(msg.wParam);      // Кінець. Повертаємось у Windows
}

////////////////////////////////////////////////////////////////////////////////////
//          Функція вікна ( Обробка повідомлень головного вікна )
// Параметри :
// HWND hWnd - Дескриптор вікна, в якому відбулася подія
// UINT message - Код (ідентифікатор) повідомлення (32-бітне значення)
// WPARAM wParam - 32-бітове значення, що містить додатк. інформ., що
залежить від типу повідомлення
```

```
// LPARAM lParam - 32-бітове значення, що містить додатк. інформ., що залежить від типу повідомлення
////////////////////////////////////////////////////////////////////////////////

LRESULT WINAPI WindowProc(HWND hWnd, UINT message,
                  WPARAM wParam, LPARAM lParam)
{
 HDC hDC;                    // Дескриптор дисплейного контексту
 PAINTSTRUCT PaintSt;    // Структура, що визначає область малювання
 RECT aRect;                 // Робочий прямокутник

 switch(message)             // Обробка вибраних повідомлень
 {
  case WM_PAINT:                 // Повідомлення для перемальовування вікна
    hDC = BeginPaint(hWnd, &PaintSt);// Підготуватися до перемальовування вікна
      GetClientRect(hWnd, &aRect);    // Визначити лівий верхній та правий нижній  кути клієнтської області
    SetBkMode(hDC, TRANSPARENT);    // Встановити режим прозорого відображення фону для тексту

    // Відобразити текст у клієнтській області
    DrawText(
        hDC,              // Дескриптор контексту пристрою
        "Відображення тексту для першого прикладу Windows-програми",
        -1,               // Індикатор рядка, який закінчується null-символом
        &aRect,       // Прямокутник,  в якому відображається текст
        DT_SINGLELINE|     // Формат тексту - одна строка
        DT_CENTER|          //           - центрування в рядку
        DT_VCENTER);       //           - центрування за висотою  aRect

        aRect.top =50; aRect.left=100;
    SetBkMode(hDC, OPAQUE);    // Встановити режим непрозорого відображення фону для тексту
        DrawText(hDC,"Відображення тексту в вершині прямокутника",-
```

```
1,&aRect,DT_TOP);
        SetBkMode(hDC, TRANSPARENT);     // Встановити режим прозоро-
го відображення фону для тексту
    TextOut(hDC, 200,400, "Відображення тексту по координатам!",34);

    EndPaint(hWnd, &PaintSt);   // Завершити операцію перемальовування
вікна
    return 0;

  case WM_DESTROY:            // Вікно знищується
    PostQuitMessage(0);
    return 0;

  default:                    // Будь-які інші повідомлення нас не цікавлять, тому
                              // викликається обробка повідомлень за замовчуванням
    return DefWindowProc(hWnd, message, wParam, lParam);
 }
}
```

Загальний вигляд вікна програми такий (рис. 6.1):

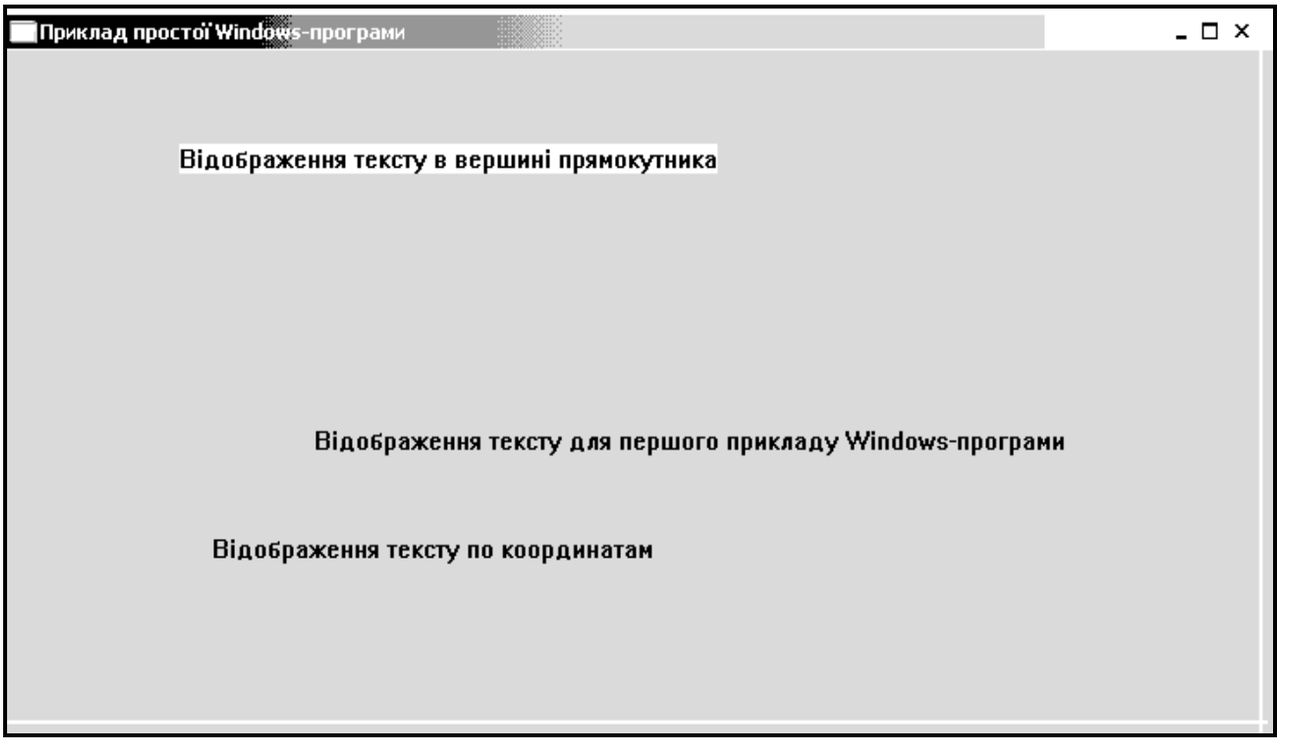

Рис. 6.1. **Вікно з результатами роботи програми до завдання 6.1**

Завдання 6.2. Розробити програму Windows-додатка, яка використовує меню, контекстне меню, діалогові вікна. Деякі елементи управління створюються до початку роботи програми (див. файл ресурсів), а деякі створюються динамічно під час виконання програми. Програма є багатофайловим проектом. Особливостями даної програми є:

1. Наявність статус-рядка з відображенням поточної дати і часу.
2. Наявність пунктів меню, які додавалися через редактор ресурсів.
3. Пункти "Товщини" відображаються ознаками "вибрано".
4. Зміна кольору фону при виборі пункту меню супроводжується виведенням "часу" у вікно.
5. Пункти меню вибору кольору фону мають "гарячі клавіші".
6. Включено додавання пунктів меню в головне меню і підменю "Фігури".
7. Передбачена обробка зміни мови пунктів меню.
8. Створюється нове діалогове вікно для виконання розрахунків (меню - Calculation).
9. У діалоговому вікні радіокнопки і чекбокси додані в редакторі ресурсів, поля для введення, кнопки і меню додаються в програмі.

```
// ТЕКСТ ГОЛОВНОГО МОДУЛЯ ПРОГРАМИ
#include "stdafx.h"
#include "LAB-6-11.h"
#include <stdio.h>
#include <commctrl.h>
#include <math.h>

#define MAX_LOADSTRING 100   // макровизначення для максимальної довжини рядка
#define ID_STATUS 200        // макровизначення ідентифікатора статусного рядка

// Коди пунктів контекстного меню
#define CM_CLEAR_WINDOW      5001  // очистити вікно
#define CM_DRAW_LINE         5002  // намалювати лінію
```

```
#define CM_ADD_MENUITEM      5003  // добавити пункт меню
#define CM_ADD_SUBMENU_ITEM  5004  // добавити пункт підменю
#define CM_CHANGE_LANG       5005  // змінити мову меню
#define CM_ENG_LANG          5051  // мова меню англійська
#define CM_UKR_LANG          5052  // мова меню українська

// Пункти меню, що дадаємо
#define CM_NEW_MENUITEM      5101  // Новий пункт меню
#define CM_NEW_MENUSUBITEM   5102  // Новий пункт підменю

// Коди для малювання фігур
#define FDRAW_RECTANGLE      1  // Прямокутник
#define FDRAW_CIRCLE         2  // Круг
#define FDRAW_CROSS          3  // Хрест
#define FDRAW_LINE           4  // Лінія
#define FDRAW_TRIANGLE       5  // Трикутник

// Глобальні змінні
HINSTANCE hInst;                    // Дескриптор поточного додатка
TCHAR szTitle[MAX_LOADSTRING];  // Текст заголовка
TCHAR szWindowClass[MAX_LOADSTRING]; // Ім'я класу головн. вікна

// Прототипи функцій, що викликаються
ATOM        MyRegisterClass(HINSTANCE hInstance);  // Реєстрація класу
вікна
BOOL        InitInstance(HINSTANCE, int);        /
LRESULT  CALLBACK       WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR  CALLBACK       About(HWND, UINT, WPARAM, LPARAM);
VOID        SetCheckMenuItem(HMENU,LONG);
VOID         ChangeLanguage(HMENU, CHAR);

// Вікно користувача
INT_PTR CALLBACK        CalcDlg(HWND, UINT, WPARAM, LPARAM);

// Головна програма
int APIENTRY _tWinMain( HINSTANCE hInstance,
                HINSTANCE hPrevInstance,
                LPTSTR    lpCmdLine,
```

```
                int       nCmdShow)
{
        UNREFERENCED_PARAMETER(hPrevInstance);
        UNREFERENCED_PARAMETER(lpCmdLine);

        MSG msg;            // змінна, що застосовується для Windows-
повідомлень
        HACCEL hAccelTable;   // Дескриптор таблиці прискорювачів (accele-
rator table).

        // Ініціалізація глобальних рядків
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_LAB611, szWindowClass,
MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        // Виконати ініціалізацію додатка
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }
        // Завантажити таблицю акселераторів ( "гарячих клавіш")
        hAccelTable = LoadAccelerators(hInstance, MAKEINTRE-
SOURCE(IDC_LAB611));

        // Цикл обробки повідомлень
        while (GetMessage(&msg, NULL, 0, 0))   // Отримати повідомлення
        {
           // Переведення  акселераторів клавіатури у повідомлення команд
меню
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                  TranslateMessage(&msg);       // Трансляція повідомлення
                   DispatchMessage(&msg);       // Диспетчеризація повідомлення
                }
        }
        return (int) msg.wParam;  // Кінець, повертаємось у Windows
```

```
}

//////////////////////////////////////////////////////////////////////
//  Функція:     MyRegisterClass()
//  Призначення: Реєстрація класу вікна
ATOM MyRegisterClass(HINSTANCE hInstance)
{
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);  // встановити розмір структури

        wcex.style              = CS_HREDRAW | CS_VREDRAW; // стиль вікна
        wcex.lpfnWndProc  = WndProc;   // функція  обробки  повідомлення
        wcex.cbClsExtra     = 0;       // Немає додаткових байтів після структури
        wcex.cbWndExtra           = 0;         // класа вікна у екземплярі вікна
        wcex.hInstance       = hInstance;  // Дескриптор екземпляра додатка
        // Встановити піктограму  додатка
        wcex.hIcon  = LoadIcon(hInstance, MAKEINTRE-
SOURCE(IDI_ICON1));
        // Встановити стандартний курсор миші у вигляді стрілочки
        wcex.hCursor  = LoadCursor(NULL, IDC_ARROW);
        // Встановити пензель для малювання фону
        wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName = MAKEINTRESOURCE(IDC_LAB611);  // задати
меню
        wcex.lpszClassName = szWindowClass;         // встановити ім'я класу
        // Встановити маленьку піктограму
        wcex.hIconSm= LoadIcon(wcex.hInstance, MAKEINTRESOURCE
(IDI_SMALL) );
   //wcex.hIconSm                = NULL;
        return RegisterClassEx(&wcex);
}
//////////////////////////////////////////////////////////////////////
//   Функція:    InitInstance(HINSTANCE, int)
// Призначення: Зберегти дескриптор програми та створити головне вікно
//
// Коментар:
```

```
//     У цій функції зберігається дескриптор програми в глобальній змінній
//     і створюється і відображається головне вікно програми
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;
  hInst = hInstance; // Збереження дескриптора в глобальній змінній

  // Створити вікно програми
  hWnd = CreateWindow(szWindowClass, szTitle,
WS_OVERLAPPEDWINDOW,
     CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance,
NULL);
  // перевіряємо успішніть створення вікна
  if (!hWnd) { return FALSE; }

  ShowWindow(hWnd, nCmdShow); // відобразити вікно
  UpdateWindow(hWnd);        // перемалюваи клієнтську частину вікна

  return TRUE;
}
////////////////////////////////////////////////////////////////////
// Функція: WndProc (HWND, UINT, WPARAM, LPARAM)
// Призначення: Обробка повідомлень головного вікна
//
// WM_COMMAND - обробка меню програми
// WM_PAINT   - відмальовування головного вікна
// WM_DESTROY - Знищення вікна (завершення роботи і вихід)

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam)
{
  INT wmId, wmEvent;          // Елементи повідомлень WM_COMMAND
  HDC hdc;                    // Дескриптор дисплейного контексту
   PAINTSTRUCT PaintSt;    // Структура, що визначає область малювання
   RECT  aRect;                // Робочий прямокутник
```

```
  HBRUSH hBrush;                // "пензель" для замальовування вікна
        HBITMAP hPic;                // Дескриптор картинки
        static HMENU hMenu;          // Дескриптор меню головного вікна
        static LONG        bColor=RGB(200,200,200);   // Поточний колір вікна
        static BYTE  newFigure=0;       // Код фігури, що малюється
        static BYTE  curThick=1;      // Поточна товщина для малювання фігур
        static SHORT cx;      // Допоміжна змінна (обчислення розміру ділянки
статусного рядка)
        static INT pParts[3]; // масив частин статусного рядка
        static HWND hStatus;  // ідентифікатор вікна (статусний рядок)
        SYSTEMTIME st;       // екземпляр структури системної дати та часу
        static CHAR Buf[60]="";              // строковий буфер
        INT day, month, year, sec, min, hour; // змінні дати та часу

        switch (message)   // Обробка вибраних повідомлень
        {
        case WM_CREATE:    // Створення вікна
                // Створюємо статусний рядок
                hStatus=CreateStatusWindow(WS_CHILD|WS_VISIBLE, "Вико-
нано", hWnd, ID_STATUS);
                SendMessage(hStatus, SB_SETTEXT, 0, (LONG)"Виконано");  //
Відображаємо текст у статус-рядку
                SetTimer(hWnd, 1, 1000, NULL);            // Встановлюємо таймер
                hMenu=GetMenu(hWnd);  // Визначаємо дескриптор меню го-
ловного вікна
      // Додаємо картинки до пунктів меню
       hPic = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAP_RED));
                SetMenuItemBitmaps(hMenu,ID_BACKCOLOR_RED,
MF_BYCOMMAND,hPic, hPic);
                hPic = LoadBitmap(hInst, MAKEINTRE-
SOURCE(IDB_BITMAP_BLUE));
                SetMenuItemBitmaps(hMenu,ID_BACKCOLOR_BLUE,
MF_BYCOMMAND,hPic, hPic);
                hPic = LoadBitmap(hInst, MAKEINTRE-
SOURCE(IDB_BITMAP_GREEN));
                SetMenuItemBitmaps(hMenu,ID_BACKCOLOR_GREEN,
MF_BYCOMMAND,hPic, hPic);
```

```
            hPic = LoadBitmap(hInst, MAKEINTRE-
SOURCE(IDB_BITMAP_YELLOW));
            SetMenuItemBitmaps(hMenu,ID_BACKCOLOR_YELLOW,
MF_BYCOMMAND,hPic, hPic);
            hPic = LoadBitmap(hInst, MAKEINTRE-
SOURCE(IDB_BITMAP_GRAY));
            SetMenuItemBitmaps(hMenu,ID_BACKCOLOR_GRAY,
MF_BYCOMMAND,hPic, hPic);
            break;
        case WM_TIMER:    // Повідомлення таймера
            GetLocalTime(&st);    // Отримати поточну дату та час
            day=st.wDay;      month=st.wMonth;      year=st.wYear;
            hour=st.wHour;  min=st.wMinute;         sec=st.wSecond;
            sprintf(Buf, "%02d/%02d/%02d", day, month, year);
           // Відобразити дату
            SendMessage(hStatus, SB_SETTEXT, 1, (LONG)Buf);
            sprintf(Buf, "%02d:%02d:%02d", hour, min, sec);
           // Відобразити час
            SendMessage(hStatus, SB_SETTEXT, 2, (LONG)Buf);
            break;

        case WM_SIZE: // Зміна розмірів вікна
           // Посилаємо повідомлення WM_SIZE
            MoveWindow(hStatus, 0,0,0,0, TRUE);
            // Перерахунок розмірів частин статусного рядка
            cx=LOWORD(lParam);
            pParts[0]=cx-200;       pParts[1]=cx-100;       pParts[2]=cx;
            // Встановлюємо розміри частин статусного рядка
            SendMessage(hStatus, SB_SETPARTS, 3, (LPARAM)pParts);
            break;
        case WM_COMMAND:
           // ID елемента, який послав поточне повідомлення
            wmId    = LOWORD(wParam);
            wmEvent = HIWORD(wParam); // інформація про тип події

            // Визначення вибраного пункту меню
            switch (wmId)
```

```
        {
        case ID_BACKCOLOR_RED :    // Пункт  "колір червоний"
                bColor= RGB(255,0,0);  // Встановити колір
                InvalidateRect (hWnd, NULL, TRUE);    // Робить
недостовірною область користувача у вікні. Фактично викликає перема-
лювання фону
                break;
        case ID_BACKCOLOR_BLUE :   // Пункт  "колір синій"
                bColor= RGB(0,0,255);
                InvalidateRect (hWnd, NULL, TRUE);
                break;
        case ID_BACKCOLOR_GREEN:   // Пункт  "колір зелений"
                bColor= RGB(0,255,0);
                InvalidateRect (hWnd, NULL, TRUE);
                break;
        case ID_BACKCOLOR_YELLOW : // Пункт  "колір жовтий"
                bColor= RGB(255,255,0);
                InvalidateRect (hWnd, NULL, TRUE);
                break;
        case ID_BACKCOLOR_GRAY :   // Пункт "колір сірий"
                bColor= RGB(200,200,200);
                InvalidateRect (hWnd, NULL, TRUE);
                break;

        // Обробка пунктів меню для малювання фігур //////////////////////
        case ID_FIGURE_RECTANGLE:              // Прямокутник
      newFigure=FDRAW_RECTANGLE;
                InvalidateRect (hWnd, NULL, TRUE);
                break;
        case ID_FIGURE_CIRCLE   :              // Круг
      newFigure=FDRAW_CIRCLE;
                InvalidateRect (hWnd, NULL, TRUE);
                break;
        case ID_FIGURE_CROSS    :              // Хрест
      newFigure=FDRAW_CROSS;
                InvalidateRect (hWnd, NULL, TRUE);
                break;
       case CM_NEW_MENUSUBITEM :   // Добавлений пункт підменю
```

```
        newFigure=FDRAW_TRIANGLE;
                    InvalidateRect (hWnd, NULL, TRUE);
                    break;
      ////////////////////////////////////////////////////////////////
              case ID_THICKNESS_LARGE:   // Пункт "Товщина велика"
                   // Встановити відмітку
                    SetCheckMenuItem(hMenu,ID_THICKNESS_LARGE);
                    curThick=7;
                    break;
              case ID_THICKNESS_AVERAGE: // Пункт "Товщина середня"
                    // Встановити відмітку
                    SetCheckMenuItem(hMenu,ID_THICKNESS_AVERAGE);
                    curThick=4;
                    break;
              case ID_THICKNESS_SMALL:   // Пункт "Товщина маленька"
                   // Встановити відмітку
                    SetCheckMenuItem(hMenu,ID_THICKNESS_SMALL);
                    curThick=2;
                    break;
      // Обробка пунктів спливаючого (контекстного) меню
////////////////////////////////
              case CM_CLEAR_WINDOW:   // Очистити вікно
                    bColor= RGB(255,255,255);  newFigure=0;
                    InvalidateRect (hWnd, NULL, TRUE);
                    break;
              case CM_DRAW_LINE:    // Намалювати круг
        newFigure=FDRAW_LINE;
                    InvalidateRect (hWnd, NULL, TRUE);
                    break;
              case CM_ADD_MENUITEM:   // Добавити пункт меню
                    if (GetMenuItemCount(hMenu)<=6) { // Есть такой пункт ?
                      AppendMenu(hMenu,
MF_ENABLED|MF_STRING,CM_NEW_MENUITEM, "New Menu Ele-
ment");
                      DrawMenuBar(hWnd);     // Перемалювати меню
                    } else {
                 MessageBox(hWnd,"Такий пункт меню вже існує","", MB_OK);
```

```
                }
                break;

            case CM_ADD_SUBMENU_ITEM:  // Добавити пункт підменю
                {
                  // Визначаємо дескриптор підменю
                  HMENU hSubMenu = GetSubMenu( hMenu, 1 );
                 // Чи є вже такий пункт ?
                  if ( GetMenuItemCount(hSubMenu)<=3) {
                   // Добавляємо пункт
                   AppendMenu(hSubMenu,
MF_ENABLED|MF_STRING,CM_NEW_MENUSUBITEM, "Triangle");
                    DrawMenuBar(hWnd);     // Перемальовуємо меню
                  } else {
                        MessageBox(hWnd,"Такий пункт підменю вже
існує","",MB_OK);
                  }
                  }
                  break;
            case CM_ENG_LANG:       // Вибір англійської мови
                  ChangeLanguage(hMenu, 'E');
                  DrawMenuBar(hWnd);
                  break;
            case CM_UKR_LANG:       // Вибір української мови
                  ChangeLanguage(hMenu, 'U');
                  DrawMenuBar(hWnd);
                  break;
     ////////////////////////////////////////////////////////////////////////
            case ID_CALCULATION:      // Переходимо у діалогове вікно
                  DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1),
hWnd, CalcDlg);
                  break;
            case IDM_ABOUT:           // Пункт "Про програму" (About)
                  DialogBox(hInst, MAKEINTRE-
SOURCE(IDD_ABOUTBOX), hWnd, About);
                  break;
            case IDM_EXIT:            // Пункт "Вихід" (Exit)
```

```
            DestroyWindow(hWnd);
            break;
        default:                // Інші повідомлення
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
        break;
case WM_PAINT:              // Малювання у клієнтській області
        hdc = BeginPaint(hWnd, &PaintSt);   // починаємо малювати
        // визначаємо координати клієнтської області
        GetClientRect (hWnd, &aRect);
        // Малюємо фігуру ?
            {
             HPEN hPen;         // дескриптор нашого "олівця"
             POINT Point;       // структура точки
             // створюємо олівець
             hPen = CreatePen(PS_SOLID,curThick,RGB(0,0,0));
             SelectObject (hdc,hPen);            // вибираємо наш олівець
             // створюємо "пензель"
             hBrush = CreateSolidBrush (bColor);
             FillRect (hdc, &aRect, hBrush);     // Заповнюємо ним вікно
            // Встановлюємо режим відображення фону для тексту
            SetBkMode(hdc, bColor);
             switch (newFigure){ // Вибір фігури для малювання
                case FDRAW_RECTANGLE :    // ПРЯМОКУТНИК
                    MoveToEx(hdc,150,150,&Point);
                    LineTo(hdc,400,150); // малюємо лінію
                    LineTo(hdc,400,350); // малюємо лінію
                    LineTo(hdc,150,350); // малюємо лінію
                    LineTo(hdc,150,150); // малюємо лінію
                    break;
                case FDRAW_CIRCLE :       // КРУГ
                    Ellipse(hdc,100,100,200,200);
                    break;
                case FDRAW_CROSS :        // ХРЕСТ
                    MoveToEx(hdc,150,150,&Point);
                    LineTo(hdc,350,350); // малюємо лінію
                    MoveToEx(hdc,150,350,&Point);
```

```
                    LineTo(hdc,350,150); // малюємо лінію
                    break;
                case FDRAW_LINE :        // ЛІНІЯ
                    MoveToEx(hdc,200,200,&Point);
                    LineTo(hdc,500,200); // малюємо лінію
                    break;
                case FDRAW_TRIANGLE :    // ТРИКУТНИК
                    MoveToEx(hdc,250,150,&Point);
                    LineTo(hdc,350,350); // малюємо лінію
                    LineTo(hdc,150,350); // малюємо лінію
                    LineTo(hdc,250,150); // малюємо лінію
                    break;
            }
          // Встановлюємо режим прозорого відображення фону для тексту
           SetBkMode(hdc, TRANSPARENT);
           TextOut(hdc,300,250,Buf,strlen(Buf));  // Виводимо "час" у вікно
            DeleteObject(hPen);      // видаляємо олівець
            DeleteObject (hBrush);   // видаляємо "пензель"
            }
            EndPaint(hWnd, &PaintSt);     // закінчуємо малювати
            break;
      case WM_RBUTTONDOWN :   // Натискання правої кнопки миші
(визов контекстного меню)
            {
     DWORD xyPos=GetMessagePos();  // Визначаємо координати "миші"
      WORD  xPos=LOWORD(xyPos);
      WORD  yPos=HIWORD(xyPos);
      // Створюємо вспливаюче  меню
      HMENU hFloatMenu=CreatePopupMenu();();
     // Створити підменю для пункту "Вибір мови"
      HMENU hLangMenu=CreatePopupMenu
       // Добавити пункти у підменю
        AppendMenu(hLangMenu,
MF_ENABLED|MF_STRING|MF_UNCHECKED,CM_ENG_LANG, "Eng-
lish");
        AppendMenu(hLangMenu,
MF_ENABLED|MF_STRING|MF_UNCHECKED,CM_UKR_LANG,
```

```
"Ukraine");
                    // Добавити пункти у вспливаюче меню

                    AppendMenu(hFloatMenu,
MF_ENABLED|MF_STRING,CM_CLEAR_WINDOW, "Clear Window");
                    AppendMenu(hFloatMenu,
MF_ENABLED|MF_STRING,CM_DRAW_LINE, "Draw Line");
                    AppendMenu(hFloatMenu,
MF_ENABLED|MF_STRING,CM_ADD_MENUITEM, "Add MenuItem");
                    AppendMenu(hFloatMenu,
MF_ENABLED|MF_STRING,CM_ADD_SUBMENU_ITEM, "Add SubMenuI-
tem");
                    AppendMenu(hFloatMenu,
MF_ENABLED|MF_POPUP,(UINT)hLangMenu, "Change Language");
                    // Відобразити вспливаюче меню
                    TrackPopupMenu(hFloatMenu,TPM_LEFTALIGN|
TPM_TOPALIGN, xPos, yPos, 0, hWnd,NULL);
                }
                break;

        case WM_DESTROY:        // Вікно знищується
      PostQuitMessage(0);
                break;
        default:    // Інші повідомлення не цікаві,  визов обробника за
замовчуванням
                return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
// Обробник повідомлення для пункту меню "About"
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wPa-
ram, LPARAM lParam)
{
   static INT  PicSize=0;    // Ознака - як виводити малюнок (0 - не виводити,
                             //    1 - оригінальний розмір,  2 - на всю форму)

   HDC hDC;   // контекст пристрою, необхідний для будь-якого малювання
```

```
   PAINTSTRUCT PaintStruc;   // інформація про пошкоджені області Вікна
   static RECT Rect;                 // структура прямокутника Вікна
   static HBITMAP hBitmap;   // дескриптор зображення
   static BITMAP BitmapInfo; // характеристики зображення
   HDC hCompatibleDC;        // Сумісний контекст в пам'яті
   HBITMAP hOldBitmap;     // дескриптор зображення для сумісного контексту

   UNREFERENCED_PARAMETER(lParam);
       switch (message)
       {
       case WM_INITDIALOG:
               // Встановлюємо новий заголовок вікна
               SetWindowText(hDlg, "About. Подвійне клацання для завантаження зображення");
               return (INT_PTR)TRUE;
       case WM_COMMAND:
            if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
               {
                       EndDialog(hDlg, LOWORD(wParam));
                       return (INT_PTR)TRUE;
               }
               break;
       case WM_LBUTTONDBLCLK :  // Обробка подвійного натискання миші
       {
          // Завантажуємо малюнок і визначаємо його характеристики
            hBitmap = (HBITMAP) LoadImage
(NULL,"Pic.bmp",IMAGE_BITMAP,0,0,LR_LOADFROMFILE);
            GetObject(hBitmap,sizeof(BITMAP),&BitmapInfo);
            // Встановлюємо ознаку  "розмір"  зображення
            PicSize=(PicSize==1)? 2:1;
            InvalidateRect(hDlg,NULL,true); // Перемальовуємо вікно
             }
             break;
```

```
        case WM_PAINT:
    // завжди починаємо малювання з виклику функції BeginPaint
    hDC = BeginPaint(hDlg,&PaintStruc); // контекст пристрою для малювання
         GetClientRect(hDlg,&Rect);          // Визначаємо координати вікна.
    if (hBitmap != 0)                        // Чи завантажений малюнок?
    {
     hCompatibleDC = CreateCompatibleDC(hDC); // створення сумісного
контексту в пам'яті
     hOldBitmap = (HBITMAP) SelectObject(hCompatibleDC,hBitmap); // за-
вантаження нашого Bitmap
          if (PicSize==1){
        // копіювання bitmap ( оригінальний розмір)

BitBlt(hDC,0,0,BitmapInfo.bmWidth,BitmapInfo.bmHeight,hCompatibleDC,0,0
,SRCCOPY);
          } else {
                // трансформація малюнка за розмірами вікна
                StretchBlt(hDC,0,0,Rect.right,Rect.bottom,hCompatibleDC,
                0,0,BitmapInfo.bmWidth,BitmapInfo.bmHeight,SRCCOPY);
          }
     SelectObject(hCompatibleDC,hOldBitmap);
     DeleteDC(hCompatibleDC);
    }
         EndPaint(hDlg,&PaintStruc); // закінчуємо малювання
    return 0; // повертаємо 0,  якщо ми обробили повідомлення
       }
       return (INT_PTR)FALSE;
}

// Встановити відмітку на вибраному пункті меню  ////////////////////////////
VOID   SetCheckMenuItem(HMENU hMenu, LONG pMenu)
{
       CheckMenuItem(hMenu,ID_THICKNESS_LARGE,MF_UNCHECKED);
       CheckMenuI-
tem(hMenu,ID_THICKNESS_AVERAGE,MF_UNCHECKED);
       CheckMenuItem(hMenu,ID_THICKNESS_SMALL,MF_UNCHECKED);
       CheckMenuItem(hMenu,pMenu,MF_CHECKED);
```

```
}
// Змінити мову пунктів меню  ////////////////////////////////////////////

VOID  ChangeLanguage(HMENU hm, CHAR c)
{
	switch (c) {
		case 'E':
			ModifyMenu(hm, IDM_EXIT,  MF_BYCOMMAND |
MF_STRING, IDM_EXIT,"Exit" );
			ModifyMenu(hm, ID_FIGURE_RECTANGLE,
MF_BYCOMMAND | MF_STRING, ID_FIGURE_RECTANGLE,"Rectangle"
);
			ModifyMenu(hm, ID_FIGURE_CIRCLE,
MF_BYCOMMAND | MF_STRING, ID_FIGURE_CIRCLE,"Circle" );
			ModifyMenu(hm, ID_FIGURE_CROSS,
MF_BYCOMMAND | MF_STRING, ID_FIGURE_CROSS,"Cross" );
			ModifyMenu(hm, ID_BACKCOLOR_RED,
MF_BYCOMMAND | MF_STRING, ID_BACKCOLOR_RED,"Red" );
			ModifyMenu(hm, ID_BACKCOLOR_BLUE,
MF_BYCOMMAND | MF_STRING, ID_BACKCOLOR_BLUE,"Blue" );
			ModifyMenu(hm, ID_BACKCOLOR_GREEN,
MF_BYCOMMAND |  MF_STRING, ID_BACKCOLOR_GREEN,"Green" );
			ModifyMenu(hm, ID_BACKCOLOR_YELLOW,
MF_BYCOMMAND | MF_STRING, ID_BACKCOLOR_YELLOW,"Yellow"
);
			ModifyMenu(hm, ID_BACKCOLOR_GRAY,
MF_BYCOMMAND | MF_STRING, ID_BACKCOLOR_GRAY,"Gray" );
			ModifyMenu(hm, ID_THICKNESS_LARGE,
MF_BYCOMMAND | MF_STRING, ID_THICKNESS_LARGE,"Large" );
			ModifyMenu(hm, ID_THICKNESS_AVERAGE,
MF_BYCOMMAND | MF_STRING, ID_THICKNESS_AVERAGE,"Average"
);
			ModifyMenu(hm, ID_THICKNESS_SMALL,
MF_BYCOMMAND | MF_STRING, ID_THICKNESS_SMALL,"Small" );
			ModifyMenu(hm, ID_CALCULATION,  MF_BYCOMMAND
| MF_STRING, ID_CALCULATION,"Calculation" );
			ModifyMenu(hm, IDM_ABOUT,  MF_BYCOMMAND |
```

```
MF_STRING, IDM_ABOUT,"About" );

                    ModifyMenu(hm,(UINT)GetSubMenu(hm,0),
MF_BYCOMMAND | MF_STRING, (UINT)GetSubMenu(hm,0),"File" );
                    ModifyMenu(hm,(UINT)GetSubMenu(hm,1),
MF_BYCOMMAND | MF_STRING, (UINT)GetSubMenu(hm,1),"Figure" );
                    ModifyMenu(hm,(UINT)GetSubMenu(hm,2),
MF_BYCOMMAND | MF_STRING, (UINT)GetSubMenu(hm,2),"BackColor"
);
                    ModifyMenu(hm,(UINT)GetSubMenu(hm,3),
MF_BYCOMMAND | MF_STRING, (UINT)GetSubMenu(hm,3),"Thickness" );
                    ModifyMenu(hm,(UINT)GetSubMenu(hm,5),
MF_BYCOMMAND | MF_STRING, (UINT)GetSubMenu(hm,5),"Help" );
                    break;

               case 'U':
                    ModifyMenu(hm, IDM_EXIT,  MF_BYCOMMAND |
MF_STRING, IDM_EXIT,"Вихід" );
                    ModifyMenu(hm, ID_FIGURE_RECTANGLE,
MF_BYCOMMAND | MF_STRING,
ID_FIGURE_RECTANGLE,"Прямокутник" );
                    ModifyMenu(hm, ID_FIGURE_CIRCLE,
MF_BYCOMMAND | MF_STRING, ID_FIGURE_CIRCLE,"Коло" );
                    ModifyMenu(hm, ID_FIGURE_CROSS,
MF_BYCOMMAND | MF_STRING, ID_FIGURE_CROSS,"Хрест" );
                    ModifyMenu(hm, ID_BACKCOLOR_RED,
MF_BYCOMMAND | MF_STRING, ID_BACKCOLOR_RED,"Червоний" );
                    ModifyMenu(hm, ID_BACKCOLOR_BLUE,
MF_BYCOMMAND | MF_STRING, ID_BACKCOLOR_BLUE,"Синій" );
                    ModifyMenu(hm, ID_BACKCOLOR_GREEN,
MF_BYCOMMAND | MF_STRING, ID_BACKCOLOR_GREEN,"Зелений" );
                    ModifyMenu(hm, ID_BACKCOLOR_YELLOW,
MF_BYCOMMAND | MF_STRING, ID_BACKCOLOR_YELLOW,"Жовтий"
);
                    ModifyMenu(hm, ID_BACKCOLOR_GRAY,
MF_BYCOMMAND | MF_STRING, ID_BACKCOLOR_GRAY,"Сірий" );
                    ModifyMenu(hm, ID_THICKNESS_LARGE,
```

```
MF_BYCOMMAND | MF_STRING, ID_THICKNESS_LARGE,"Велика" );
                    ModifyMenu(hm, ID_THICKNESS_AVERAGE,

MF_BYCOMMAND | MF_STRING, ID_THICKNESS_AVERAGE,"Середня"
);
                    ModifyMenu(hm, ID_THICKNESS_SMALL,
MF_BYCOMMAND | MF_STRING, ID_THICKNESS_SMALL,"Маленька"
);
                    ModifyMenu(hm, ID_CALCULATION,  MF_BYCOMMAND
| MF_STRING, ID_CALCULATION,"Обчислення" );
                    ModifyMenu(hm, IDM_ABOUT,  MF_BYCOMMAND |
MF_STRING, IDM_ABOUT,"Про програму" );

                    ModifyMenu(hm,(UINT)GetSubMenu(hm,0),
MF_BYCOMMAND | MF_STRING, (UINT)GetSubMenu(hm,0),"Файл" );
                    ModifyMenu(hm,(UINT)GetSubMenu(hm,1),
MF_BYCOMMAND | MF_STRING, (UINT)GetSubMenu(hm,1),"Фігури" );
                    ModifyMenu(hm,(UINT)GetSubMenu(hm,2),
MF_BYCOMMAND | MF_STRING, (UINT)GetSubMenu(hm,2),"Колір фону"
);
                    ModifyMenu(hm,(UINT)GetSubMenu(hm,3),
MF_BYCOMMAND | MF_STRING, (UINT)GetSubMenu(hm,3),"Товщина" );
                    ModifyMenu(hm,(UINT)GetSubMenu(hm,5),
MF_BYCOMMAND | MF_STRING, (UINT)GetSubMenu(hm,5),"Допомога"
);
                    break;
               default: ;
        }
 }

// ТЕКСТ ПРОГРАМИ, ЯКА ВІДОБРАЖАЄ ЕЛЕМЕНТИ ДІАЛОГОВОГО
ВІКНА
#include "stdafx.h"
#include "stdio.h"
#include "LAB-6-11.h"

// Коди  елементів управління, що додаються
```

```
#define ID_MATR11          6001   // Елемент матриці [1,1]
#define ID_MATR12          6002   // Елемент матриці [1,2]
#define ID_MATR21          6003   // Елемент матриці [2,1]
#define ID_MATR22          6004   // Елемент матриці [2,2]
#define ID_OTVET           6010   // Поле для виведення  відповіді
#define ID_CALC_BTN        6011   // Кнопка для виконання обчислень
#define ID_CLEAR_BTN       6012   // Кнопка для очистки
#define ID_EXIT_BTN        6013   // Кнопка для виходу

#define ID_ZAGOLOVOK   6100   // Елемент для виведення  заголовка

// Коди алгоритмів розрахунку
#define     ALG_OPRED          1   // розрахунок визначника матриці
#define     ALG_SUMMA          2   // Розрахунок суми елементів
#define     ALG_MINMAX         3   // Розрахунок мінімаксу

// Пункти меню у вікні
#define CM_PUNCT_OBRAM        9002 // Пункт меню для виведення /
очищення обрамлення відповіді
#define CM_PUNCT_EXIT     9003 // Пункт меню "Вихід"
#define CM_PUNCT_HELP     9004 // Пункт меню для виклику довідки

// Координати і розміри вікна для виведення відповіді
#define     X_OTV                  30  // Координата X
#define     Y_OTV                  150  // Координата Y
#define     XSIZE_OTV              200  // Розмір по X
#define     YSIZE_OTV               30  // Розмір по Y

extern HINSTANCE hInst;      // Дескриптор поточного додатка
HWND hDlgGlobal;        // Для збереження дескриптора діалогового вікна

// Прототипи функцій для виконання розрахунків над елементами матриці
VOID DoRaschet(INT cod);        // Виконання розрахунків за обраним алго-
ритмом
VOID ShowOtvet(char *str);      // Виведення відповіді у вікно заданим
шрифтом
```

```
// Оброблювач повідомлень для пункту меню "Calculation"
INT_PTR CALLBACK CalcDlg(HWND hDlg, UINT message, WPARAM wPa-
ram, LPARAM lParam)
{
        INT wmId, wmEvent;       // Елементи повідомлення WM_COMMAND
        INT cod_raschet;            // Код алгоритму розрахунку

        hDlgGlobal=hDlg;               // Зберігаємо дескриптор діалогового вікна
        static HMENU hCMenu;       // Дескриптор меню поточного вікна
    static HMENU hSubMenu1;       // Дескриптор підменю "Меню"
        static BYTE DrawObram=0;    // Чи малювати обрамлення відповіді
(ознака)

        HDC hdc;                  // Дескриптор дисплейного контексту
    PAINTSTRUCT PaintSt;    // Структура, яка визначає область малювання
    RECT  aRect;                 // Робочий прямокутник
        HBRUSH hBrush;        // "Пензель" для зафарбовування вікна
        CHAR stroka[40];        // Рядок для отримання значення з елемента
управління

        UNREFERENCED_PARAMETER (lParam);

        switch (message)
        {
        case WM_INITDIALOG:
       // ДИНАМІЧНЕ ДОДАВАННЯ ЕЛЕМЕНТІВ НА ФОРМУ
       // Створення тексту-заголовка для матриці 2x2
       CreateWindow("static", "Елементи матриці: ",

WS_CHILD|WS_VISIBLE|SS_LEFT|WS_EX_TRANSPARENT,30,10,150,30,
                          hDlg, (HMENU)ID_ZAGOLOVOK, hInst, NULL);

       // Створення полів для елементів матриці 2x2
                CreateWindow("edit", NULL,
WS_CHILD|WS_VISIBLE|ES_LEFT|WS_BORDER,30,40,60,30,
                           hDlg, (HMENU)ID_MATR11, hInst, NULL);
                CreateWindow("edit", NULL,
```

```
WS_CHILD|WS_VISIBLE|ES_LEFT|WS_BORDER,100,40,60,30,
                        hDlg, (HMENU)ID_MATR12, hInst, NULL);

            CreateWindow("edit", NULL,
WS_CHILD|WS_VISIBLE|ES_LEFT|WS_BORDER,30,80,60,30,
                        hDlg, (HMENU)ID_MATR21, hInst, NULL);
            CreateWindow("edit", NULL,
WS_CHILD|WS_VISIBLE|ES_LEFT|WS_BORDER,100,80,60,30,
                        hDlg, (HMENU)ID_MATR22, hInst, NULL);

      // Створення тексту-відповіді
       CreateWindow("static", "",
            WS_CHILD|WS_VISIBLE|SS_LEFT|WS_EX_TRANSPARENT,
            X_OTV,Y_OTV,XSIZE_OTV,YSIZE_OTV,
            hDlg, (HMENU)ID_OTVET, hInst, NULL);

         // Створення кнопок "Обчислити", "Очистити" і "Вихід"
            CreateWindow("button", "Обчислити",
WS_CHILD|WS_VISIBLE|BS_DEFPUSHBUTTON,30,230,90,25,
                        hDlg, (HMENU)ID_CALC_BTN, hInst, NULL);
            CreateWindow("button", "Очистити",
WS_CHILD|WS_VISIBLE|BS_DEFPUSHBUTTON,130,230,90,25,
                        hDlg, (HMENU)ID_CLEAR_BTN, hInst, NULL);
            CreateWindow("button", "Вихід",
WS_CHILD|WS_VISIBLE|BS_DEFPUSHBUTTON,350,230,90,25,
                        hDlg, (HMENU)ID_EXIT_BTN, hInst, NULL);

            // Створюємо підменю і додаємо до нього пункти
            hSubMenu1=CreatePopupMenu();
            AppendMenu(hSubMenu1, MF_ENABLED | MF_STRING ,
CM_PUNCT_OBRAM, "Обрамлення");
       AppendMenu(hSubMenu1, MF_ENABLED | MF_SEPARATOR, 0, 0);
       AppendMenu(hSubMenu1, MF_ENABLED | MF_STRING,
CM_PUNCT_EXIT, "Вихід");

      // Створюємо головне меню вікна
      hCMenu=CreateMenu();      // Створити меню для вікна
```

```
      SetMenu(hDlg, hCMenu);   // Встановити меню для вікна ( зв'язати ме-
ню з вікном)
               // Додати в меню вікна  підменю та один пункт "Довідка"
            AppendMenu(hCMenu,
MF_ENABLED|MF_POPUP,(UINT)hSubMenu1, "Меню");
            AppendMenu(hCMenu,
MF_ENABLED|MF_STRING,CM_PUNCT_HELP, "Довідка");
               DrawMenuBar(hDlg);     // Перемалювати  меню
               return (INT_PTR)TRUE;

         case WM_PAINT:
               hdc = BeginPaint(hDlg, &PaintSt);
               GetClientRect (hDlg, &aRect);
               if (DrawObram){
                  HPEN hPen;                        // дескриптор нашого олівця));
                   // створюємо олівець
                   hPen = CreatePen(PS_SOLID,5,RGB(0,0,0
                  SelectObject (hdc,hPen);              // вибираємо наш олівець
                  // Встановити режим прозорого відображення фону для тексту
                   SetBkMode(hdc, TRANSPARENT);
                  // Малюємо прямокутник
                  Rectangle(hdc,X_OTV-1,Y_OTV-1,
X_OTV+XSIZE_OTV+1,Y_OTV+YSIZE_OTV+1);
           DeleteObject(hPen);                  // видаляємо олівець
               }
      EndPaint(hDlg, &PaintSt);
               break;

         case WM_COMMAND:
              // ID елемента, який послав поточне повідомлення
               wmId    = LOWORD(wParam);
               wmEvent = HIWORD(wParam); // інформація про тип події

               // Визначення натиснутої кнопки
               switch (wmId)
               {
                  case CM_PUNCT_EXIT:  // Вибраний пункт меню "Вихід"
```

```
                    case ID_EXIT_BTN  :   // Нажата кнопка "Вихід"
                      // Скидання ознаки малювання обрамлення
                       DrawObram=0;
                      // Закінчити работу з діалоговим вікном
                       EndDialog(hDlg, LOWORD(wParam));
                            return (INT_PTR)TRUE;

                    case ID_CLEAR_BTN:  // Натиснута кнопка "Очистити"
                    // Очищаємо поля введення елементів матриці і відповіді
                            SetDlgItemText(hDlg,ID_MATR11,"");
                            SetDlgItemText(hDlg,ID_MATR12,"");
                            SetDlgItemText(hDlg,ID_MATR21,"");
                            SetDlgItemText(hDlg,ID_MATR22,"");
                            SetDlgItemText(hDlg,ID_OTVET,"");
                            break;
                    case ID_CALC_BTN:   // Натиснута кнопка "Обчислити"
                    // Визначаємо,  який  алгоритм обраний для розрахунку
                            cod_raschet=0;
          if (SendDlgItemMessage(hDlg,IDC_RADIO1,BM_GETCHECK,0,0))
                             cod_raschet=ALG_OPRED;
          else if (SendDlgItemMes-
sage(hDlg,IDC_RADIO2,BM_GETCHECK,0,0))
                             cod_raschet=ALG_SUMMA;
          else if (SendDlgItemMes-
sage(hDlg,IDC_RADIO3,BM_GETCHECK,0,0))
                              cod_raschet=ALG_MINMAX;
                            // Виконуємо розрахунок за обраним алгоритмом
                             DoRaschet(cod_raschet);
                                 break;
                           // Вибраний пункт меню "Обрамлення"
                    case CM_PUNCT_OBRAM:
                            // Змінюємо ознаку "обрамлення" на зворотну
                            DrawObram=!DrawObram;
CheckMenuItem (hCMenu, CM_PUNCT_OBRAM, (DrawObram) ?
MF_CHECKED : MF_UNCHECKED);
                           // Визначаємо текст відповіді
                            GetDlgItemText(hDlgGlobal,ID_OTVET,stroka,40);
```

```
                        // Визиваємо перемалювання вікна
                        InvalidateRect (hDlg, NULL, TRUE);
                        ShowOtvet(stroka);     // Відображаємо відповідь
                        break;
                    // Вибраний пункт меню "Довідка"
                    case CM_PUNCT_HELP:
                        MessageBox(hDlg,"Вибраний пункт меню
\"Довідка\"","",0);
                        break;
                }
                break;
        }
        return (INT_PTR)FALSE;
}
// Виведення тексту Відповіді вибраним шрифтом ////////////////////////////////
VOID ShowOtvet(char *str)
{
  int c1, c2, c3;        // статус контрольних кнопок
  HFONT hFont;     // об'єкт-шрифт
  LOGFONT lf =    // екземпляр структури LOGFONT - параметри шрифту
                  {20, 0, 0, 0, 100, 0, 0, 0, DEFAULT_CHARSET,
                  OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
DRAFT_QUALITY, DEFAULT_PITCH, "Times New Roman"};
  // Задаємо шрифт для виведення  відповіді
  // Отримуємо статус контрольних кнопок
  c1=SendDlgItemMessage(hDlgGlobal, IDC_CHECK1, BM_GETCHECK,0,0);
  c2=SendDlgItemMessage(hDlgGlobal, IDC_CHECK2, BM_GETCHECK,0,0);
  c3=SendDlgItemMessage(hDlgGlobal, IDC_CHECK3, BM_GETCHECK,0,0);

  lf.lfWeight = (c1)? 1000:100;     // ознака "товщини" шрифту
  lf.lfItalic = (c2)? 1:0;               // ознака "наклону" шрифту
  lf.lfHeight = (c3)? 25:20;            // значення "висоти" шрифту
  hFont = CreateFontIndirect(&lf);  // створюємо фонт

  // Для елемента  "Відповідь" встановлюємо шрифт
  SendDlgItemMessage(hDlgGlobal, ID_OTVET, WM_SETFONT, (WPA-
RAM)hFont, MAKELPARAM(TRUE,0));
```

```
  SetDlgItemText(hDlgGlobal,ID_OTVET,str);   // Задаємо текст елемента управління "Відповідь"
  DeleteObject(hFont);                       // Видаляємо об'єкт "фонт"
}

// Виконати розрахунок  за обраним алгоритмом  ////////////////////////////////////
VOID DoRaschet(INT cod)
{
    INT i,j;
    CHAR stroka[40];    // Рядок для отримання значення з елемента управління
    FLOAT matr[2][2];   // Матриця для зберігання значень введених елементів
        FLOAT rez=0,rez2=0;
        for(i=0; i<2; i++)
                for (j=0; j<2; j++){
                        GetDlgItemText(hDlgGlobal,ID_MATR11+2*i+j,stroka,20);
                        matr[i][j]=atof(stroka);
                }
        switch (cod) {
                case ALG_OPRED :
                        rez=matr[0][0]*matr[1][1]-matr[1][0]*matr[0][1];
                        break;
                case ALG_SUMMA :
                        rez=matr[0][0]+matr[0][1]+matr[1][0]+matr[1][1];
                        break;
                case ALG_MINMAX:
                        rez = (matr[0][0]<matr[0][1])?matr[0][0]: matr[0][1];
                        rez2= (matr[1][0]<matr[1][1])?matr[1][0]: matr[1][1];
                        rez = (rez>rez2) ? rez: rez2;
                        break;
                default: MessageBox(hDlgGlobal,"Не заданий алгоритм розрахунку","Помилка",MB_OK);
                        return;
        }
  sprintf(stroka,"Відповідь=%7.*f",2,rez);  // Формуємо рядок відповіді
  ShowOtvet(stroka);                        // Відображаємо відповідь
```

}

// ТЕКСТ RESOURCE.H ЗГЕНЕРОВАНОГО VISUAL STUDIO C++

//{{NO_DEPENDENCIES}}

// Microsoft Visual C++ generated include file.

// Used by LAB-6-11.rc

//

```
#define IDC_MYICON                      2
#define IDD_LAB611_DIALOG               102
#define IDS_APP_TITLE                   103
#define IDD_ABOUTBOX                    103
#define IDM_ABOUT                       104
#define IDM_EXIT                        105
#define IDI_LAB611                      107
#define IDI_SMALL                       108
#define IDC_LAB611                      109
#define IDR_MAINFRAME                   128
#define IDB_BITMAP_RED                  133
#define IDB_BITMAP_BLUE                 134
#define IDB_BITMAP_GRAY                 137
#define IDB_BITMAP_GREEN                138
#define IDB_BITMAP_YELLOW               139
#define IDI_ICON1                       140
#define IDD_DIALOG1                     141
#define IDC_RADIO1                      1000
#define IDC_RADIO2                      1001
#define IDC_RADIO3                      1002
#define IDC_CHECK1                      1003
#define IDC_CHECK2                      1004
#define IDC_CHECK3                      1005
#define ID_FIGURE_RECTANGLE             32771
#define ID_FIGURE_CIRCLE                32772
#define ID_FIGURE_CROSS                 32773
#define ID_BACKCOLOR_RED                32774
#define ID_BACKCOLOR_BLUE               32775
#define ID_BACKCOLOR_GREEN              32776
#define ID_BACKCOLOR_YELLOW             32777
#define ID_BACKCOLOR_GRAY               32778
```

```
#define ID_THICKNESS_LARGE              32779
#define ID_THICKNESS_AVERAGE            32780
#define ID_THICKNESS_SMALL              32781
#define ID_CALCULATION                  32783
#define ID_COLOR                        32791
#define ID_THICK_POPUP                  32792
#define ID_COLOR_POPUP                  32793
#define ID_FILE_POPUP                   32794
#define ID_FIGURE_POPUP                 32795
#define ID_HELP_POPUP                   32796
#define ID_Menu                         32797
#define IDC_STATIC                      -1

// Next default values for new objects
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NO_MFC                     1
#define _APS_NEXT_RESOURCE_VALUE        142
#define _APS_NEXT_COMMAND_VALUE         32798
#define _APS_NEXT_CONTROL_VALUE         1006
#define _APS_NEXT_SYMED_VALUE           110
#endif
#endif
```

// ФАЙЛ РЕСУРСІВ ПРОГРАМИ //
// Microsoft Visual C++ generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
//
//
// Generated from the TEXTINCLUDE 2 resource.
//
#define APSTUDIO_HIDDEN_SYMBOLS
#include "windows.h"
#undef APSTUDIO_HIDDEN_SYMBOLS

```
//////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS
//////////////////////////////////////////////////////////////////////////////
// Russian resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_RUS)
#ifdef _WIN32
LANGUAGE LANG_RUSSIAN, SUBLANG_DEFAULT
#pragma code_page(1251)
#endif //_WIN32

//////////////////////////////////////////////////////////////////////////////
// Icon
//
// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDI_LAB611              ICON                    "LAB-6-11.ico"
IDI_SMALL               ICON                    "small.ico"
IDI_ICON1               ICON                    "icon1.ico"

//////////////////////////////////////////////////////////////////////////////
// Menu
//
IDC_LAB611 MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit",                       IDM_EXIT
    END
    POPUP "Figure"
    BEGIN
        MENUITEM "Rectangle",                   ID_FIGURE_RECTANGLE
        MENUITEM "Circle",                      ID_FIGURE_CIRCLE
        MENUITEM "Cross",                       ID_FIGURE_CROSS
    END
    POPUP "BackColor"
    BEGIN
```

```
      MENUITEM "Gray     Alt+W",          ID_BACKCOLOR_GRAY

      MENUITEM "Red      Alt+R",          ID_BACKCOLOR_RED
      MENUITEM "Blue     Alt+B",          ID_BACKCOLOR_BLUE
      MENUITEM "Green    Alt+G",          ID_BACKCOLOR_GREEN
      MENUITEM "Yellow   Alt+Y",          ID_BACKCOLOR_YELLOW
   END
   POPUP "Thickness"
   BEGIN
      MENUITEM "Large",                   ID_THICKNESS_LARGE
      MENUITEM "Average",                 ID_THICKNESS_AVERAGE
      MENUITEM "Small",                   ID_THICKNESS_SMALL
   END
   MENUITEM "Calculation",                ID_CALCULATION
   POPUP "&Help"
   BEGIN
      MENUITEM "&About ...",              IDM_ABOUT
   END
END

////////////////////////////////////////////////////////////////////////////
// Accelerator
//
IDC_LAB611 ACCELERATORS
BEGIN
   "/",          IDM_ABOUT,            ASCII,  ALT, NOINVERT
   "?",          IDM_ABOUT,            ASCII,  ALT, NOINVERT
   "R",          ID_BACKCOLOR_RED,     VIRTKEY, ALT, NOINVERT
   "B",          ID_BACKCOLOR_BLUE,    VIRTKEY, ALT, NOINVERT
   "G",          ID_BACKCOLOR_GREEN,   VIRTKEY, ALT, NOINVERT
   "Y",          ID_BACKCOLOR_YELLOW,  VIRTKEY, ALT, NOINVERT
   "W",          ID_BACKCOLOR_GRAY,    VIRTKEY, ALT, NOINVERT
END

////////////////////////////////////////////////////////////////////////////
// Dialog
//
```

```
IDD_ABOUTBOX DIALOG  22, 17, 230, 75
STYLE DS_SETFONT | DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "About"
FONT 8, "System"
BEGIN
    ICON            IDI_LAB611,IDC_MYICON,14,9,16,16
    LTEXT           "LAB-6-11 Version
1.0",IDC_STATIC,49,10,119,8,SS_NOPREFIX
    LTEXT           "Copyright (C) 2009",IDC_STATIC,49,20,119,8
    DEFPUSHBUTTON   "OK",IDOK,195,6,30,11,WS_GROUP
END

IDD_DIALOG1 DIALOGEX 0, 0, 320, 182
STYLE DS_SETFONT | DS_MODALFRAME | DS_FIXEDSYS | WS_POPUP |
WS_CAPTION | WS_SYSMENU
FONT 8, "MS Shell Dlg", 400, 0, 0x1
BEGIN
    CONTROL
"Визначник",IDC_RADIO1,"Button",BS_AUTORADIOBUTTON,181,33,66,10
    CONTROL         "Сума
елементів",IDC_RADIO2,"Button",BS_AUTORADIOBUTTON,181,47,74,10
    CONTROL
"Мінімакс",IDC_RADIO3,"Button",BS_AUTORADIOBUTTON,181,61,49,10
    CONTROL         "Жирний",IDC_CHECK1,"Button",BS_AUTOCHECKBOX |
WS_TABSTOP,182,95,44,10
    CONTROL         "Похилий",IDC_CHECK2,"Button",BS_AUTOCHECKBOX |
WS_TABSTOP,182,107,54,10
    CONTROL         "Крупний",IDC_CHECK3,"Button",BS_AUTOCHECKBOX |
WS_TABSTOP,182,120,46,10
    GROUPBOX        "Шрифт відповіді",IDC_STATIC,172,84,133,53
    GROUPBOX        "Алгоритм розрахунку",IDC_STATIC,172,20,133,56
END

#ifdef APSTUDIO_INVOKED
//////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
```

```
//
1 TEXTINCLUDE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE
BEGIN
    "#define APSTUDIO_HIDDEN_SYMBOLS\r\n"
    "#include ""windows.h""\r\n"
    "#undef APSTUDIO_HIDDEN_SYMBOLS\r\n"
    "\0"
END

3 TEXTINCLUDE
BEGIN
    "\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED

/////////////////////////////////////////////////////////////////////////////
//
// Bitmap
//

IDB_BITMAP_RED          BITMAP                "red.bmp"
IDB_BITMAP_BLUE         BITMAP                "blue.bmp"
IDB_BITMAP_GRAY         BITMAP                "gray.bmp"
IDB_BITMAP_GREEN        BITMAP                "bitmap1.bmp"
IDB_BITMAP_YELLOW       BITMAP                "Yellow.bmp"

/////////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
```

```
//
#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO
BEGIN
    IDD_DIALOG1, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 313
        TOPMARGIN, 7
        BOTTOMMARGIN, 175
    END
END
#endif    // APSTUDIO_INVOKED


////////////////////////////////////////////////////////////////////////////
//
// String Table
//
STRINGTABLE
BEGIN
    IDS_APP_TITLE          "LAB-6-11"
    IDC_LAB611             "LAB611"
END
#endif    // Russian resources

////////////////////////////////////////////////////////////////////////////
#ifndef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

Головне вікно програми наведене на рис. 6.2. Діалогове вікно, яке з'являється при виборі пункту меню Calculation наведене на рис. 6.3.

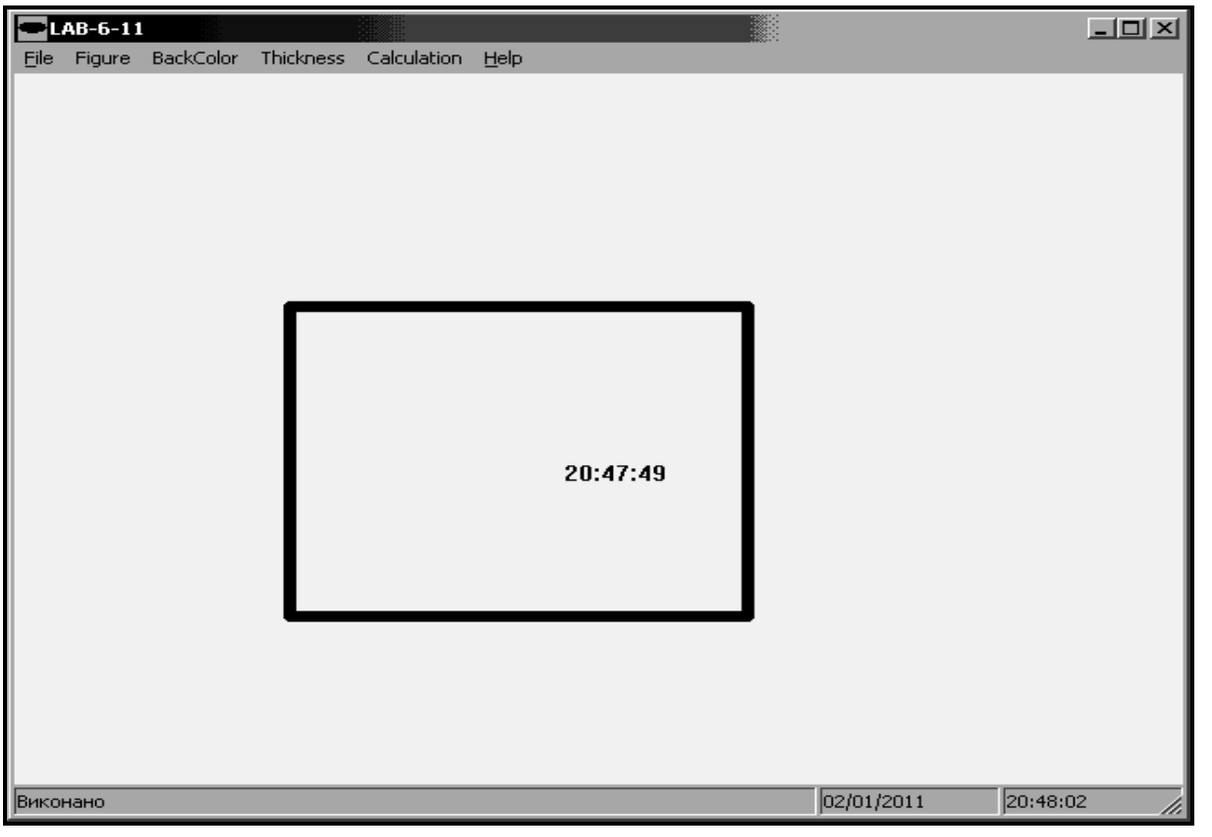

Рис. 6.2. **Головне вікно програми до завдання 6.2**

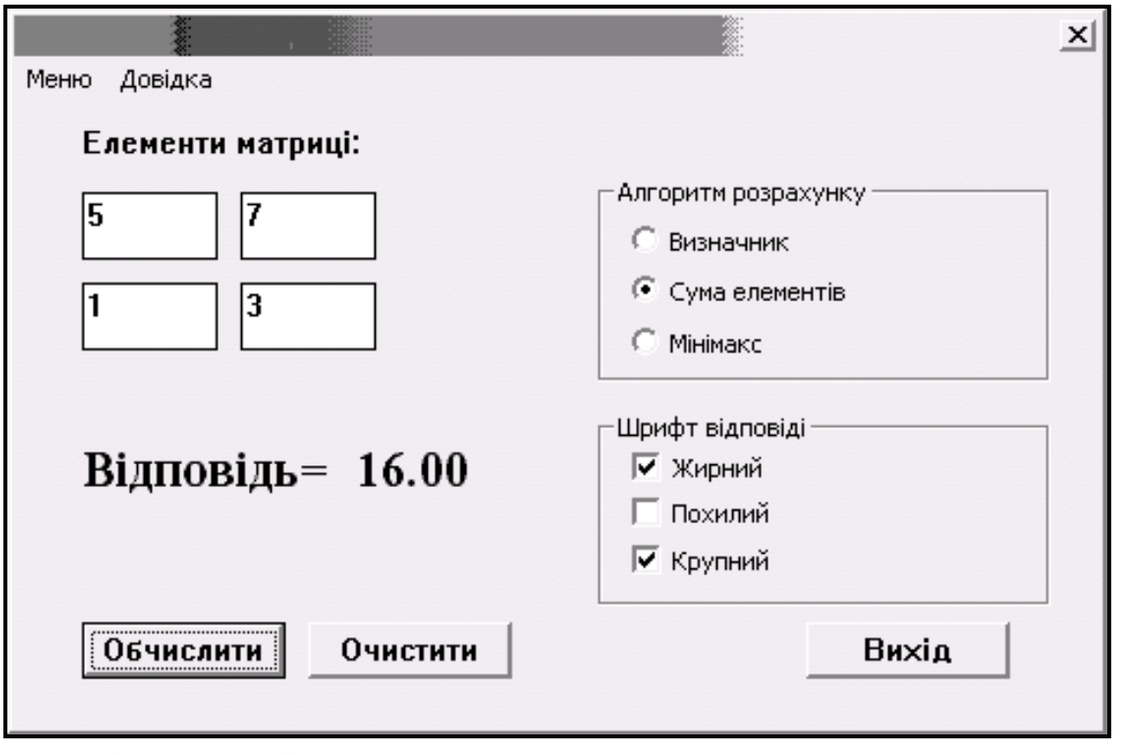

Рис. 6.3. **Діалогове вікно програми до завдання 6.2**

Практичне заняття 7. Робота з процесами і потоками

В операційній системі Windows підтримуються багатопотокові процеси. Процес (process) є об'єктом, якому належать ресурси додатка. Потік (thread) – це суть усередині процесу, яку ядро Windows направляє на виконання. Без нього програма процесу не може виконуватися. Потік розділяє разом з процесом загальний адресний простір, код і глобальні дані. У кожного потоку є власні регістри, стек і механізми введення, у тому числі і черга прихованих повідомлень.

Багатозадачність (multitasking) – це можливість операційної системи виконувати декілька програм одночасно. Основою цього принципу є використання операційною системою апаратного таймера для виділення відрізків часу для будь-якого з одночасно виконуваних процесів. Якщо ці відрізки часу досить маленькі, і процесор не переобтяжений занадто великим числом програм, то користувачеві здасться, що усі ці програми виконуються паралельно.

Многопотоковість - це можливість програми бути багатозадачною. Програма може бути розділена на окремі потоки виконання, яке, як здається, виконуються паралельно.

7.1. Процеси

Функції управління процесами і потоками наведені в табл. 7.1.

Таблиця 7.1

Функції управління процесами і потоками

Функція	***Призначення***
1	2
AttachThreadInput	Переадресує введення з одного потоку в інший
CancelWaitableTlmer	Відміняє таймер очікування, встановлюючи його в неактивний стан
CreateProcess	Створює об'єкт процесу і потоку й запускає процес
CreateProcessAsUser	Створює об'єкт процесу і потоку в контексті захисту, встановленого для цього користувача

Продовження табл. 7.1.

1	2
CreateRemoteThreed	Створює потік, який виконується в адресному просторі іншого процесу
CreateThread	Створює об'єкт потоку і запускає його
DuplicateHandle	Копіює дескриптор будь-якого об'єкта з тим, щоб він міг використовуватися іншим процесом
ExitProcess	Завершує процес і усі потоки процесу і повертає код завершення
ExitThread	Завершує один потік процесу
GetCurrentProcess	Повертає хендл поточного процесу
GetCurrentProcessID	Повертає ідентифікатор поточного процесу
GetCurrentThread	Повертає хендл поточного потоку
GetCurrentThreadID	Повертає ідентифікатор поточного потоку
GetExitCodeProcess	Повертає код завершення вказаного процесу. Ця функція може також використовуватися для встановлення факту завершення процесу
GetExitCodeThread	Повертає код завершення вказаного потоку. Ця функція може також використовуватися для встановлення факту завершення потоку
GetPriorityClass	Визначає клас пріоритету процесу
GetProcessHeap	Повертає дескриптор купи процесу
GetProcessHeaps	Повертає масив дескрипторів, які містять усі купи, що належать процесу
GetProcessPriorityBoost	Повертає підвищення пріоритету для процесу
GetProcessShutdownParameters	Повертає параметри закриття процесу
GefProcessTimes	Повертає тимчасові характеристики виконання процесу
GefProcessVersion	Повертає очікувану процесом версію Windows
GetQueueStatus	Повертає вхідний стан черги потоку
GetThreedContext	Використовується відладчиком для збереження машинного контексту потоку
GetThreadPriority	Повертає клас пріоритету цього потоку
GetThreadPriorityBoost	Повертає підвищення пріоритету потоку
OpenProcess	Повертає дескриптор вказаного об'єкта
ReadProcessMemory	Читає вміст пам'яті з адресного простору процесу

Закінчення табл. 7.1

1	2
RegisterHotKey	Створює оперативну клавішу для цього потоку
ResumeThread	Поновлює виконання припиненого потоку
SetPriorityClass	Встановлює клас пріоритету процесу
SetProcessPriorityBoost	Встановлює підвищення пріоритету для процесу
SetProcessShutdownParameters	Встановлює параметри закриття процесу
SetThreadPriority	Встановлює клас пріоритету потоку
SetThreadPriorityBoost	Встановлює підвищення пріоритету для потоку
Sleep	Припиняє виконання потоку на заданий проміжок у мілісекундах. При цьому може бути заплановане виконання іншого потоку
SuspendThread	Припиняє виконання процесу
TerminateProcess	Завершує процес
TerminateThread	Завершує виконання потоку

Окремо розглянемо функцію CreateProces(), яка створює об'єкт процесу і потоку, і запускає процес.

BOOL CreateProcess(

LPCTSTR lpApplicationName, // ім'я виконуваного модуля

LPTSTR lpCoramandLine, // командний рядок

LPSECURITY_ATTRIBUTES lpProcessAttributes, // захист процесу

LPSECURITY_ATTRIBUTES lpThreadAttributes, // захист потоку

BOOL blnheritHandles, // ознака спадкоємства дескриптора

DWORD dwCreationFlags, // прапори створення процесу

LPVOID IpEnvironment, // блок нового середовища оточення

LPCTSTR lpCurrentDirectory, // поточний каталог

LPSTARTUPINFO IpStartUpInfo, // вид головного вікна

LPPROCESS_INFORMATION lpProcInfo //інформація про процес

);

Повертаєме значення: у разі успішного створення процесу і потоку – TRUE, в іншому випадку – FALSE.

Параметри:

lpApplicationName і **lpComrnandLine** - використовуються разом для вказівки виконуваної програми аргументів командного рядка.

lpsaProcess і lpsaThread – покажчики на структури атрибутів захисту процесу і потоку. Значенням NULL відповідає використання атрибутів захисту, заданих за замовчуванням.

bInheritHandles – показує чи наслідує новий процес успадковані відкриті дескриптори із поточного процесу.

dwCreationFlags – може об'єднувати в собі безліч значень прапорів, наприклад такі:

CREATE_SUSPENDED – вказує на те, що основний потік буде створений у припиненому стані і почне виконуватися лише після виклику функції ResumeThread;

DETACHED_PROCESS і CREATE_NEW_CONSOLE – взаємовиключні значення, які не повинні встановлюватися обидва одночасно. Перший прапор означає створення нового процесу, у якого відсутня консоль, другий – процес з консоллю.

lpEnvironment –- вказує на блок параметрів налаштування оточення нового процесу. Якщо задано значення NULL, то новий процес використовуватиме значення параметрів оточення батьківського процесу.

lpCurDir – покажчик на рядок, що містить шлях до поточного каталогу нового процесу. Якщо задано значення NULL, то як поточний каталог використовуватиметься робочий каталог батьківського процесу.

lpStartupInfo – покажчик на структуру STARTUPINFO, яка описує зовнішній вигляд основного вікна і містить дескриптори стандартних пристроїв нового процесу.

lpProInfo – покажчик на структуру, в яку будуть поміщені повертані функцією значення дескрипторів і глобальних ідентифікаторів процесу і потоку.

Коли потік у додатку викликає CreateProcess, система створює об'єкт ядра "процес" з початковим значенням лічильника числа його користувачів, рівним 1. Цей об'єкт – не сам процес, а компактна структура даних, через яку операційна система управляє процесом. (Об'єкт ядра "процес" слід розглядати як структуру даних із статистичною інформацією про процес), Потім система створює для нового процесу віртуальний адресний простір і завантажує в нього код і дані як для виконуваного файла, так і для будь-яких DLL (якщо такі потрібні).

Далі система формує об'єкт ядра "потік" (з лічильником, рівним 1)

для первинного потоку нового процесу. Як і в першому випадку, об'єкт ядра "потік" – це компактна структура даних, через яку система управляє потоком. Первинний потік починає з виконання стартового коду з бібліотеки C/C++, який урешті-решт викликає функцію main у вашій програмі.

При створенні нового процесу Windows-функціями використовуються елементи структури STARTUPINFO.

STARTUPINFO:

```
typedef struct _STARTUPINFO {
DWORD cb;
PSTH lpReserved;
PSTR lpDesktop;
PSTR lpTitle;
DWORD dwX;
DWORD dwY;
DWORD dwXSize;
DWORD dwYSize;
DWORD dwXCountChars;
DWORD dwYCountChars;
DWORD dwFillAttribute;
DWORD dwFlags;
WORD wShowWindOw;
WORD cbReserved2;
PBYTE lpReserved2;
HANDLE hStdInput;
HANDLE hStdOutput;
HANDLE hStdError;
} STARTUPINFO, *LPSTARTUPINFO;
```

Слід сказати, що більшість додатків породжують процеси з атрибутами за замовчуванням, але і в цьому випадку потрібно ініціалізувати усі елементи структури STARTUPINFO хоч би нульовими значеннями, а в перший елемент cb структури заносити розмір цієї структури:

```
STARTUPINFO si = { sizeof(si)};
CreateProcess(., &si, ..};
```

Щоб отримати копію структури STARTUPINFO, що ініціалізувала

батьківським процесом, додаток може викликати:

VOID GetStartupInfo(LPSTARTUPINFO LpStartupInfo);

Аналізуючи цю структуру, дочірній процес може змінювати свою поведінку залежно від значень її елементів. У документації на Windows про це чітко не сказано, але перед викликом GetStartupInfo треба ініціалізувати перший елемент структури STARTUPINFO:

```
STARTUPINFO si = { sizeof(si)};
GetStartupInfo(&si);
```

Розглянемо приклад, в якому створюється новий процес для виклику текстового блокнота notepad.exe:

```
STARTUPINFO si = { sizeof(si)};
PROCESS_INFORMATION pi;
TCHAR szCommandLine[] = TEXT ("NOTEPAD");
CreateProcess(NULL, szCommandLine, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi);
```

Параметр szCommandLme дозволяє вказати повний командний рядок, використовуваний функцією CreateProcess при створенні нового процесу. Розбираючи цей рядок, функція вважає, що першим компонентом в ній є ім'я виконуваного файла, який ви хочете запустити. Якщо в імені цього файла не вказано розширення, вона вважає його EXE. Далі функція приступає до пошуку заданого файла і робить це в такому порядку:

1. Каталог, що містить EXE-файл батьківського процесу.
2. Поточний каталог батьківського процесу.
3. Системний каталог Windows.
4. Основний каталог Windows.
5. Каталоги, перераховані в змінній оточення PATH.

Звичайно, якщо в імені файла вказаний повний шлях доступу, система відразу звертається туди і не переглядає ці каталоги. Знайшовши потрібний виконуваний файл, вона створює новий процес і проектує код і дані виконуваного файла на адресний простір цього процесу.

Приклад 7.1. Розглянемо приклад, в якому при натисненні "1" відбувається створення дочірнього процесу і запуск "WINWORD", а при натисненні "2" відбувається створення дочірнього процесу і запуск "EXCEL".

```
#include "stdafx.h"
#include <iostream>
#include <windows.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    int a;
        do
        {
                cout << "Введіть 1 для  Word або 2 для Excel\n";
                cin >> a;
        } while( (a!=1) && (a!=2));
        if (a==1)
        {
        /* в перший елемент cb структури STARTUPINFO заносимо розмір
        цієї структури */
                STARTUPINFO si = { sizeof(si)};
                PROCESS_INFORMATION pi;
                /* ініціалізуємо масив символів szCommandLine рядком, який
        містить шлях до додатка WINWORD MS Office 2007 (для іншої версії
        MS Office папка Office12 буде мати іншу назву або інший шлях )*/
                TCHAR szCommandLine[] = TEXT("c:\\Program Files\\Microsoft
Office\\Office12\\WINWORD");
                /* створює об'єкт процесу */
                CreateProcess(NULL, szCommandLine, NULL, NULL, FALSE, 0,
NULL, NULL, &si, &pi);
        }
        if (a==2)     {
                /* в перший елемент структури STARTUPINFO заносимо розмір
        цієї структури */
                STARTUPINFO si = { sizeof(si)};
                PROCESS_INFORMATION pi;
        /* ініціалізуємо масив символів szCommandLine рядком, який містить
        шлях до додатка EXCEL MS Office 2007 (для іншої версії MS Office
        папка Office12 буде мати іншу назву або інший шлях )*/
                TCHAR szCommandLine[] = TEXT("c:\\Program Files\\Microsoft
```

```
Office\\Office12\\EXCEL");
        /* створює об'єкт процесу */
        CreateProcess(NULL, szCommandLine, NULL, NULL, FALSE, 0,
NULL, NULL, &si, &pi);
    }
}
```

Приклад 7.2. Розглянемо приклад, в якому пропонується аналогічний вибір калькулятора, Excel або Notepad, проте в даному випадку напишемо функцію MyCreateProcess () для створення нового процесу, і змусимо його виконати відкриття одного із запропонованих вище застосувань (рис. 7.1).

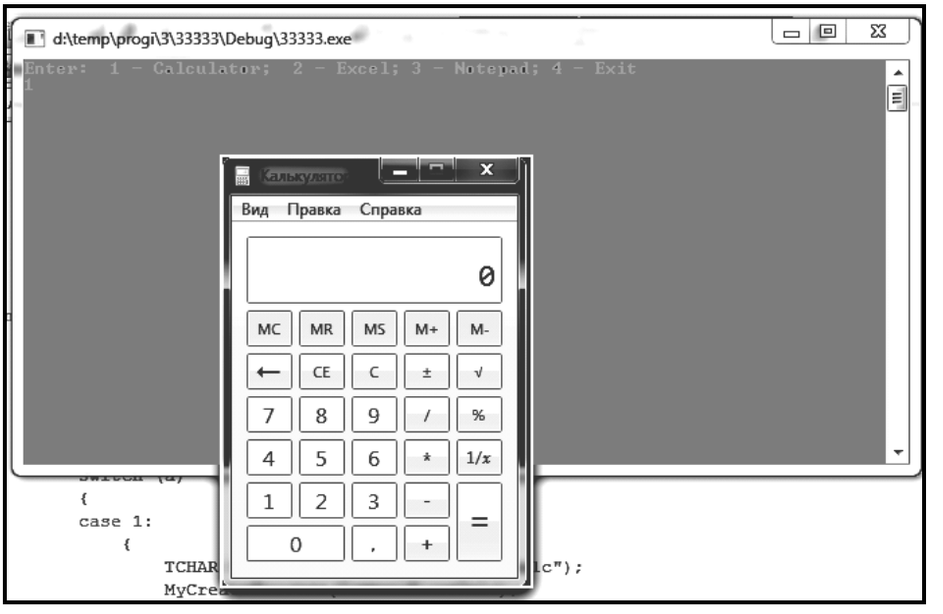

Рис. 7.1. **Дочірній процес відкриває вибране застосування**

Програма повинна дочекатися закриття запущеного застосування, після чого видасть повідомлення про його закриття (рис. 7.2).

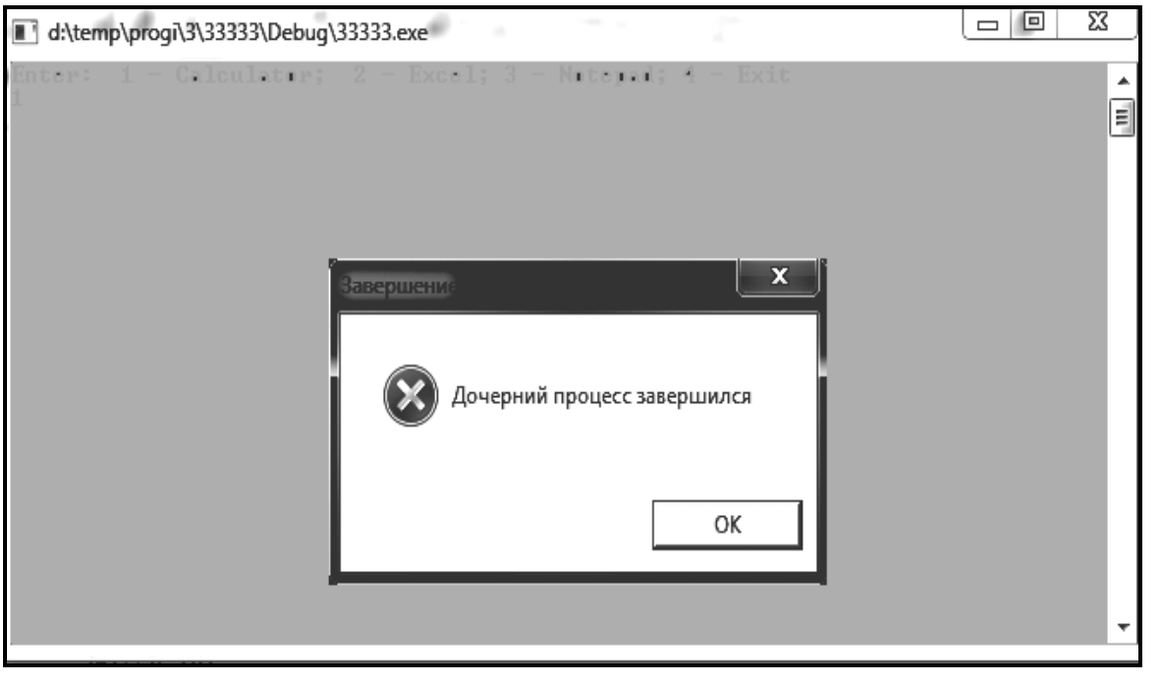

Рис. 7.2. **Дочірній процес завершився**

```
#include "stdafx.h"
#include <iostream>
#include <windows.h>

using namespace std;
// функція створення нового процесу
BOOL MyCreateProcess (TCHAR CommandLine[])
{
	STARTUPINFO si = { sizeof(si)};
	PROCESS_INFORMATION pi;
	// породжуємо дочірній процес
	BOOL Rez = CreateProcess(NULL, CommandLine, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi);
	if (Rez)
	{
	  /* припиняємо виконання батьківського процесу, поки не завершиться дочірній процес */
		WaitForSingleObject(pi.hProcess, INFINITE);

		// дочірній процес завершився;
		LPCWSTR Caption = TEXT("Завершення");
		LPCWSTR Text = TEXT("Дочірній процес завершився");
```

```
            MessageBox(NULL, Text, Caption, MB_ICONSTOP);

            // закриваємо описувач процесу
            CloseHandle(pi.hProcess);
        }
        return Rez;
}

int _tmain(int argc, _TCHAR* argv[])
{
        int a;
        do
        {
            // вибір додатка
            cout << "Enter:  1 - Calculator;  2 - Excel; 3 - Notepad; 4 - Exit\n";
            cin >> a;

            switch (a)
            {
            case 1: // калькулятор
            {
                /*ініціалізуємо масив символів CommandLineCalc рядком,
            який містить ім'я додатка або шлях */
                TCHAR CommandLineCalc[] = TEXT("calc");
                // запуск функції створення нового процесу
                MyCreateProcess(CommandLineCalc);
            }
            break;
            case 2: //EXCEL MS Office 2007
            {
                /*ініціалізуємо масив символів CommandLineCalc рядком,
            який містить ім'я додатка або шлях */
                TCHAR CommandLineExcel[] = TEXT("c:\\Program
Files\\Microsoft Office\\Office12\\EXCEL");
                // запуск функції створення нового процесу
                MyCreateProcess(CommandLineExcel);
            }
```

```
            break;
            case 3: // блокнот
            {
                /*ініціалізуємо масив символів CommandLineCalc рядком,
            який містить ім'я додатка або шлях */
                TCHAR CommandLineNotepad[] = TEXT("notepad");
                // запуск функції створення нового процесу
                MyCreateProcess(CommandLineNotepad);
            }
            break;
            case 4: // вихід
            {
                LPCWSTR Caption = TEXT("Вихід");
                LPCWSTR Text = TEXT("Вихід з програми");
                MessageBox(NULL, Text, Caption, MB_ICONSTOP);
            }
            break;
            }
        } while( a!=4 );
}
```

У функції MyCreateProcess () створили новий процес і, якщо створення пройшло успішно, викликали функцію WaitForSingleQbject().

Функція DWORD WaitForSingleObject(HANDLE hObject, DWORD dwTimeOut) затримує виконання коду до тих пір, поки об'єкт, визначуваний параметром bObject, не перейде у вільний (незайнятий) стан. Об'єкт "процес" переходить у такий стан при його завершенні, при цьому виклик WaitForSingleObject припиняє виконання потоку батьківського процесу до завершення породженого їм процесу.

Звернення до CloseHandle в наведеному вище фрагменті коду примушує систему зменшити значення лічильників об'єктів "потік" і "процес" до нуля і тим самим звільнити пам'ять, займану цими об'єктами.

7.2 Потоки

Процес нічого не виконує, він просто служить контейнером потоків. Потоки завжди створюються в контексті якого-небудь процесу, і усе їх

життя проходить тільки в його межах. На практиці це означає, що потоки виконують код і маніпулюють даними в адресному просторі процесу. Тому, якщо два і більше потоків виконується в контексті одного процесу, усі вони ділять один адресний простір. Потоки можуть виконувати один і той же код і маніпулювати одними і тими ж даними, а також спільно використовувати описувачі об'єктів ядра, оскільки таблиця описувачів створюється не в окремих потоках, а в процесах.

Потік (thread) визначає послідовність виконання коду в процесі. При ініціалізації процесу система завжди створює первинний потік. Починаючись зі стартового коду з бібліотеки C/C++, який, у свою чергу викликає вхідну функцію (main) з вашої програми, він живе до того моменту, коли вхідна функція повертає управління стартовому коду і той викликає функцію ExitProcess. Більшість додатків обходяться єдиним, первинним потоком. Проте процеси можуть створювати додаткові потоки, що дозволяє їм ефективніше виконувати свою роботу.

Функція CreateThread

Ми вже говорили, як при виклику функції CreateProcess з'являється на світ первинний потік процесу. Якщо ви хочете створити додаткові потоки, треба викликати з первинного потоку функцію CreateThread:

```
HANDLE WINAPI CreateThread (
LPSECURITY_ATTRIBUTES lpThreadAttributes, //атрибути захисту
SIZE_T dwStackSize, // розмір стека в байтах
LPTHREAD_START_ROUTINE lpStartAddress, // адреса функції
LPVOID lpParameter, // адреса параметра
DWORD dwCreationFlags, // прапорці створення потоку
LPDWORD lpThreadId // ідентифікатор потоку
);
```

При кожному виклику цієї функції система створює об'єкт ядра "потік". Це не сам потік, а компактна структура даних, яка використовується операційною системою для управління потоком і зберігає статистичну інформацію про потік. Отже, об'єкт ядра "потік" – повний аналог об'єкта ядра "процес".

Параметр **lpThreadAttributes** є покажчиком на структуру SECURITY_ATTRIBUTES. Якщо ви хочете, щоб об'єкту ядра "потік" були присвоєні атрибути захисту за замовчуванням (що найчастіше і

буває), передайте в цьому параметрі NULL. А щоб дочірні процеси змогли наслідувати описувач цього об'єкту, визначите структуру SECURITY_ATTRIBUTES.

Параметр **dwStackSize** визначає, яку частину адресного простору потік зможе використовувати під свій стек. Кожному потоку виділяється окремий стек. Функція Create Process, запускаючи додаток, викликає CreateThread, яка ініціалізує первинний потік процесу. При цьому CreateProcess заносить в параметр dwStackSize значення, що зберігається у виконуваному файлі.

Параметр **lpStartAddress** визначає адреса функції потоку, з якою повинен буде почати роботу створюваний потік, а параметр **lpParameter** ідентичний параметру pvParam функції потоку. Функція потоку повинна мати такий прототип:

DWORD WINAPI ім'я_функції_потоку(LPVOID lpParameters);

Видно, що функції потоку може бути переданий єдиний параметр lpParameter, який є покажчиком на порожній тип. Це обмеження виходить з того, що функція потоку викликається операційною системою, а не прикладною програмою. Програми операційної системи є виконуваними модулями і тому вони повинні викликати тільки функції, сигнатура яких заздалегідь визначена. Тому для потоків визначили найпростіший список параметрів, який містить тільки покажчик.

Оскільки функції потоків викликаються операційною системою, то вони також дістали назву функції зворотного виклику.

Параметр **dwCreationFlags** визначає додаткові прапори, що управляють створенням потоку. Він приймає одне з двох значень:

0 – виконання потоку починається негайно;

CREATE_SUSPENDED – система створює потік, ініціалізував його, і припиняє до наступних вказівок. Прапор CREATE_SUSPENDED дозволяє програмі змінити які-небудь властивості потоку перед тим, як він почне виконувати код. Правда, необхідність у цьому виникає досить рідко.

Параметр **lpThreadId** функції CreateThread – це адреса змінної типу DWORD, в якій функція повертає ідентифікатор, приписаний системою новому потоку.

Синхронізація потоків у призначеному для користувача режимі

ОС Windows краще всього працює, коли усі потоки можуть

займатися своєю справою, не взаємодіючи один з одним. Проте така ситуація дуже рідкісна. Зазвичай потік створюється для виконання певної роботи, про завершення якої, ймовірно, захоче дізнатися інший потік.

Усі потоки в системі повинні мати доступ до системних ресурсів – купи, послідовних портів, файлів, вікон і т. д. Якщо один з потоків запросить монопольний доступ до якого-небудь ресурсу, то іншим потокам, яким теж потрібний цей ресурс, не вдасться виконати свої завдання. А з іншого боку, просто неприпустимо, щоб потоки безконтрольно користувалися ресурсами. Інакше може вийти так, що один потік пише в блок пам'яті, з якого інший щось прочитує.

Атомарний доступ: сімейство Inferlockect-функцій

Велика частина синхронізації потоків пов'язана з атомарним доступом (atomic access) – монопольним захопленням ресурсу, потоком, що звертається до нього. Наведемо простий приклад.

```
// визначаємо глобальну змінну long g_x = 0;
DWORD WINAPI ThreadFunc1(LPVOID lpParameters)
{
        g_x++;
        return(0);
}
DWORD WINAPI ThreadFunc2(LPVOID lpParameters}
{
        g_x++;
        return(0);
}
```

Ми оголосили глобальну змінну g_x і ініціалізували її нульовим значенням. Тепер уявіть, що ми створили два потоки: один виконує ThreadFunc1, інший – ThreadFunc2. Код цих функцій ідентичний: обидві збільшують значення глобальної змінної g_x на 1. Тому, напевно, коли обидва потоки завершать свою роботу, значення g_x дорівнюватиме 2. Чи це так? Можливо. При такому коді заздалегідь сказати, яким буде кінцеве значення g_x, не можна. І ось чому.

Значення з g_x поміщається в регістр.

Значення регістра збільшується на 1.

Значення з регістра поміщається назад в g_x.

Навряд чи обидва потоки виконуватимуть цей код в один і той же час. Якщо вони робитимуть це по черзі - спочатку один, потім інший, тоді ми отримаємо таку картину:

Потік 1 в регістр поміщається 0.
Значення регістра збільшується на 1.
Потік 1 значення 1 поміщається в g_x.
Потік 2 в регістр поміщається 1.
Потік 2 значення регістра збільшується до 2.
Потік 2 значення 2 поміщається в g_x.

Після виконання обох потоків значення g_x дорівнюватиме 2. Це якраз те, що ми чекали: узявши змінну з нульовим значенням, двічі збільшили її на 1 і отримали в результаті 2. Але Windows - це середовище, яке підтримує багатопоточність і витісняючу багатозадачність. Значить, процесорний час у будь-який момент може бути відібраний у одного потоку і переданий іншому. Тоді код, наведений вище, може виконуватися і таким чином:

У регістр поміщається 0.
Потік 1 значення регістра збільшується на 1.
Потік 2 в регістр поміщається 0.
Потік 2 значення регістра збільшується на 1.
Потік 2 значення 1 поміщається в g_x.
Значення 1 поміщається в g_x.

А якщо код виконуватиметься саме так, кінцеве значення g_x виявиться рівним 1, а не 2.

Проте в Windows є ряд функцій, які гарантують коректні результати виконання коду.

Вирішення цієї проблеми має бути простим. Усе, що нам потрібне, – це спосіб, що гарантує приріст значення змінної на рівні атомарного доступу, тобто без переривання іншими потоками. Сімейство Interlocked-функцій якраз і дає нам ключ до рішення подібних проблем. Усі функції з цього сімейства маніпулюють змінними на рівні атомарного

доступу. Розглянемо InterlockedExchangeAdd.

LONG InterlockedExchangeAdd (PLONG plAddend, LONG lIncrement)

При виклику цієї функції, передаючи адреса змінної типу LONG і вказуючи значення що додається, InterlockedExchangeAdd гарантує, що операція буде виконана атомарно. Перепишемо код:

```
// визначаємо глобальну змінну long g_x = 0;
DWORD WINAPI ThreadFunc1(LPVOID lpParameters)
        {
      InterlockedExchangeAdd(&g_x, 1);
      return(0);
}
DWORD WINAPI ThreadFunc2(LPVOID lpParameters)
{
      InterlockedExchangeAdd(&g_x, 1);
      return(0);
}
```

У будь-якому потоці, де треба модифікувати значення змінної типу LONG, що розділяється (тобто є загальною), слід користуватися лише Interlocked-функціями і ніколи не удаватися до стандартних операторів мови C/C++:

```
//  змінна типу LONG, використовувана декількома потоками
LONG g_x;
//  неправильний спосіб збільшення змінної типу LONG
g_x++;
// правильний спосіб збільшення змінної типу LONG
InterlockedExchangeAdd(&g_x, 1);
```

Ще дві функції з цього сімейства:

```
LONG InterlockedExchange( PLONG plTarget, LONG lValue);
PVOTD InterlockedExchangePointer(PVOID* ppvTarget, PVOID* pvValue);
```

InterlockedExchange і ***InterlockedExchangePointer*** монопольно замінюють поточне значення змінної типу LONG, адреса якої передається в першому параметрі, на значення, що передається в другому параметрі. У 32-розрядному застосуванні обидві функції працюють з 32-розрядними значеннями, але в 64-розрядній програмі перша оперує з 32-розрядними значеннями, а друга – з 64-розрядними.

Складніші методи синхронізації потоків

Interlocked-функції хороші, коли вимагається монопольно змінити всього одну змінну. Але реальні програми мають справу із структурами даних, які набагато складніші ніж єдина 32, - або 64-бітова змінна. Щоб дістати доступ на атомарному рівні до таких структур даних, забудьте про Interlocked-функції і використовуйте інші механізми, пропоновані Windows.

7.3. Критичні секції

Критична секція (critical section) – це невелика ділянка коду, що вимагає монопольного доступу до якихось загальних даних. Вона дозволяє зробити так, щоб одноразово тільки один потік діставав доступ до певного ресурсу. Природно, система може у будь-який момент витіснити ваш потік і підключити до процесора інший, але жоден з потоків, яким потрібний зайнятий вами ресурс, не отримає процесорний час до тих пір, поки ваш потік не вийде за межі критичної секції.

Приклад 7.3. Приклад коду, який демонструє, що може статися без критичної секції:

```
const int MAX_TIMES = 1000; // максимальне значення лічильника
int g_nIndex = 0; // лічильник
DWORD g_dwTimes[MAX_TIMES]; /* масив значень свідчень системного часу*/

DWORD WINAPI FirstThread(LPVOID lpParameters)
{
    while (g_nIndex < MAX_TIMES)
```

```
        {
            // GetTickCount() свідчення системного часу
            g_dwTimes[g_nIndex] = GetTickCount();
            g_nIndex++; // збільшення лічильника
        }
        return(0),
    }
    DWORD WINAPI SecondThread(LPVOID lpParameters)
    {
        while (g_nIndex < MAX_TIMES)
        {
            g_nIndex++; // збільшення лічильника
            g_dwTimes[g_nIndex - 1] = GetTickCount();
        }
        return(0);
    }
```

Тут передбачається, що функції обох потоків дають однаковий результат, хоч вони і закодовані з невеликими відмінностями. Якби виконувалася тільки функція FirstThread, вона заповнила б масив g_dwTimes набором чисел зі зростаючими значеннями. Це правильно і відносно SecondThread, якби вона теж виконувалася незалежно. В ідеалі обидві функції навіть при одночасному виконанні повинні як і раніше заповнювати масив тим же набором чисел. Але в нашому коді виникає проблема: масив g_dwTimes не буде заповнений, як належить, тому що функції обох потоків одночасно звертаються до одних і тих же глобальних змінних. От як це може статися.

Допустимо, ми тільки що почали виконання обох потоків у системі з одним процесором. Першим включився в роботу другий потік, тобто функція SecondThread, і тільки вона встигла збільшити лічильник g_nIndex на 1, як система витіснила її потік і перейшла до виконання FirstThread. Та заносить в g_dwTimes[1] свідчення системного часу GetTickCount(), і процесор знову перемикається на виконання другого потоку. SecondThread тепер привласнює елементу g_dwTtmes[1 - 1] нові свідчення системного часу. Оскільки ця операція виконується пізніше, нові свідчення, природно, вище, ніж записані в елемент g_dwTimes[1] функцією FirstThread. Відмітьте також, що спочатку заповнюється перший елемент масиву і тільки потім нульовий. Таким чином, дані в

масиві опиняються помилковими.

Візьмемо приклад з управлінням пов'язаним списком об'єктів. Якщо доступ до пов'язаного списку не синхронізований, один потік може додати елемент в список у той момент, коли інший потік намагається знайти в ньому якийсь елемент. Ситуація стане ще загрозливішою, якщо обидва потоки одночасно додадуть у список нові елементи. Отже, використовуючи критичні секції, можна і треба координувати доступ потоків до структур даних.

Приклад 7.4. Тепер поправимо цей фрагмент коду за допомогою критичної секції, використовуючи виклики функцій **EnterCrtticalSection()** і **LeaveCriticalSection()**:

```
const int MAX_TIMES = 1000; // максимальне значення лічильника
int g_nIndex = 0; // лічильник

DWORD g_dwTimes[MAX_TIMES]; /* масив значень свідчень системного часу*/

CRITICAL_SECTION g_cs; // об‘єкт критичної секції
DWORD WINAPI FirstThread(LPVOID lpParameters)
{
    for (BOOL fContinue = TRUE; fContinue; )
    {
        EnterCriticalSection(&g_cs);
        if (g_nIndex < MAX_TIMES)
        {
        // GetTickCount() свідчення системного часу
            g_dwTimes[g_nlndex] = GetTickCount();
            g_nIndex++;
        }
        else
            fContinue = FALSE;
        LeaveCriticalSection(&g_cs);
    }
    return(0);
}
```

```
DWORD WINAPI SecondThread(LPVOID lpParameters)
{
    for (BOOL fContinue = TRUE; fContinue; )
    {
        EnterCriticalSection(&g_cs);
        if (g__nIndex < MAX_TIMES)
        {
            g_nIndex++;
            g_dwTimes[g_nIndex - 1] = GetTickCount();
        }
        else
            fContinue = FALSE;
        LeaveCriticalSecLion(&g_cs);
    }
    return(0);
}
```

У нашому прикладі функція EnterCriticalSection() блокує потік, якщо на цій критичній ділянці коду присутній інший потік. Очікуючий потік розблоковується після того, як інший потік виконає функцію LeaveCriticalSection(). Тому був створений екземпляр структури даних CRITICAL_SECTION - g_cs, причому увесь код, який працює з ресурсом (у нашому прикладі це рядки з g_nIndex і g_dwTimes), що розділяється, помістили між викликами EnterCriticalSection і LeaveCriticalSection. Зверніть увагу, що при викликах цих функцій ми передаємо адреса g_cs.

Якщо є ресурс, що розділяється декількома потоками, ви повинні створити екземпляр структури CRITICAL_SECTION. Тепер у кожній ділянці коду, де ви звертаєтеся до ресурсу, що розділяється, викликайте EnterCriticaSection(), передаючи їй адреса структури CRITICAL_SECTION, яка виділена для цього ресурсу. Іншими словами, потік, бажаючи звернутися до ресурсу, повинен спочатку переконатися, чи не зайнятий він. Структура CRITICAL_SECTION ідентифікує ресурс, з яким хоче працювати потік, а функція EnterCriticalSection допустить потік, що викликав її, до ресурсу, якщо визначить, що той вільний. У іншому випадку (ресурс зайнятий) EnterCriticalSection змусить чекати, поки не звільниться ресурс.

Потік, покидаючи ділянку коду, де він працював із захищеним ресурсом, повинен викликати функцію LeaveCriticalSection. Тим самим

він повідомляє систему про те, що цей ресурс звільнився. Якщо ви забудете це зробити, система вважатиме, що ресурс все ще зайнятий, і не дозволить звернутися до нього іншим потокам, які чекають.

Зазвичай структури CRITICAL_SECTION створюються як глобальні змінні, доступні усім потокам процесу. Але ніщо не заважає нам створювати їх як локальні змінні або змінні, що динамічно розміщуються в купі. Є тільки дві умови, яких потрібно дотримуватися. По-перше, усі потоки, яким може знадобитися ресурс, повинні знати адреса структури CRITICAL_SECTION, яка захищає цей ресурс. Ви можете отримати її адреса, використовуючи будь-який з існуючих механізмів. По-друге, елементи структури CRITICAL_SECTION слід ініціалізувати до звернення якого-небудь потоку до захищеного ресурсу. Структура ініціалізувалася викликом:

VOID InitializeCriticalSection(PCRITICAL_SECTION pcs);

Ця функція ініціалізувала елементи структури CRITICAL_SECTION, на яку вказує параметр pcs. Оскільки уся робота цієї функції полягає в ініціалізації декількох змінних-членів, вона не дає збоїв і тому нічого не повертає (void). InitializeCriticalSection має бути викликана до того, як один з потоків звернеться до EnterCriticalSection.

Якщо ви знаєте, що структура CRITICAL_SECTION більше не знадобиться жодному потоку, видалите її, викликавши DeleteCriticalSection:

VOID DeleteCriticalSection(PCRITICAL__SECTION pcs);

Вона скидає усі змінні-члени усередині цієї структури. Природно, не можна видаляти критичну секцію у той момент, коли нею все ще користується який-небудь потік.

Ділянка коду, що працює з ресурсом, який розділяється, починається викликом:

VOID EnterCriticalSection(PCRITICAL_SECTION pcs);

Перше, що робить EnterCriticalSection, – досліджує значення елементів структури CRITICAL_SECTION. Якщо ресурс зайнятий, у них містяться відомості про те, який потік користується ресурсом.

У кінці ділянки коду, що використовує ресурс, який розділяється, має бути присутній виклик.

VOID LeaveCriticalSection(PCRITICAL_SECTION pcs);

Ця функція переглядає елементи структури CRITICAL_SECTION і зменшує лічильник числа захоплень ресурсу батьківським потоком на 1. Якщо його значення більше 0, LeaveCriticalSection нічого не робить і просто повертає управління.

Приклад 7.5. У консольному вікні при натисненні 1 створюється потік **FirstThread**, в якому через 300 мс збільшується глобальна змінна m_count на 5 одиниць і якщо, був створений до цього потік Second-Thread, то закриває його. Аналогічно при натисненні 2 створюється потік **SecondThread**, в якому через 500 мс зменшується глобальна змінна m_count на 7 одиниць і якщо, був створений до цього потік FirstThread, то закриває його.

На екрані (рис. 7.3) відображається процес збільшення або зменшення в різних потоках глобальної змінної m_count, яка захищена критичною секцією.

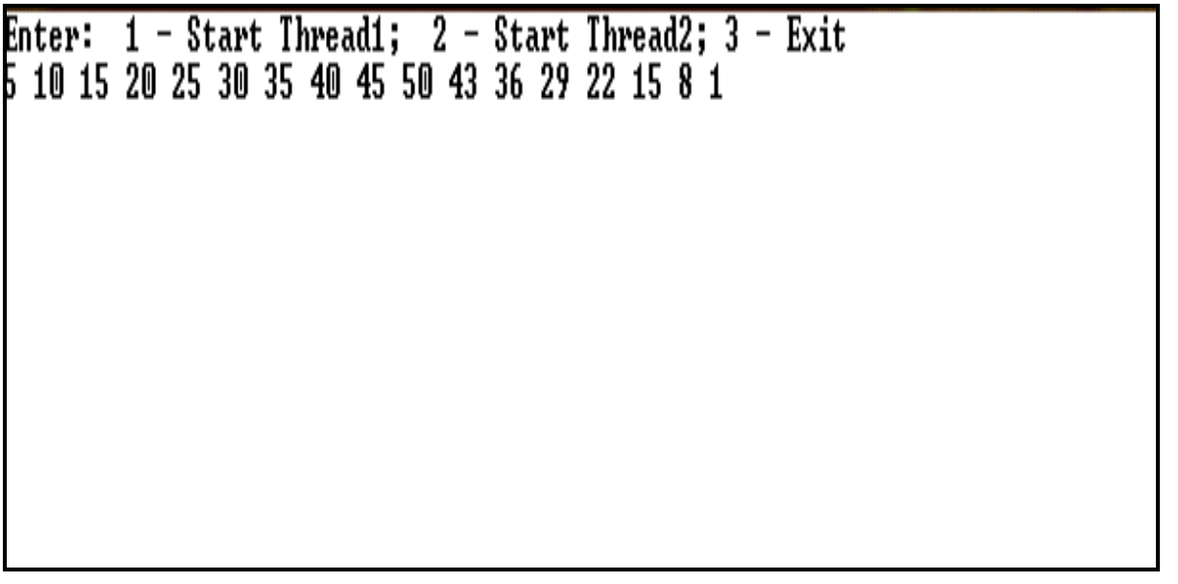

Рис. 7.3. **Приклад роботи програми до завдання 7.5**

```
#include "stdafx.h"
#include <windows.h>
#include <conio.h>

int m_count = 0, /*Лічильник – глобальна змінна, до якої отримують доступ два потоки через критичну секцію */
```

```
int a;
BOOL fContinue1 = FALSE;      // Прапорець запуску потоку 1
BOOL fContinue2 = FALSE;      // Прапорець запуску потоку 2
HANDLE hThread1,  hThread2; // Визначник потоків 1 та 2

/*Як правило структури CRITICAL_SECTION створюються як глобальні змінні, доступні усім потокам процесу*/
CRITICAL_SECTION g_cs;

/* створюємо глобальні функції FirstThread() і SecondThread() з критичною секцією  g_cs для розмежування доступу між двома потоками до глобальної змінної m_count */
DWORD WINAPI FirstThread(LPVOID lpParameters)
{
	while(1)
	{
		if (fContinue1)
		{
			Sleep(300); // інтервал 300 мс
			// критична секція
			EnterCriticalSection(&g_cs);
			m_count+=5;
			LeaveCriticalSection(&g_cs);

			_cprintf( "%d ", m_count); //виведення на екран
		}
		else
			break;
	}
	return(0);
}

DWORD WINAPI SecondThread(LPVOID lpParameters)
{
	while(1)
	{
		if (fContinue2)
```

```
		{
			Sleep(500); // інтервал 500 мс
			// критична секція
			EnterCriticalSection(&g_cs);
			m_count-=7;
			LeaveCriticalSection(&g_cs);

			_cprintf( "%d ", m_count); //виведення на екран
		}
		else
			break;
	}
	return(0);
}

int _tmain(int argc, _TCHAR* argv[])
{
	/*Елементи структури CRITICAL_SECTION слід ініціалізувати до звернення якого-небудь потоку до захищеного ресурсу*/
	InitializeCriticalSection(&g_cs);
	BOOL stop = FALSE; // прапорець зупинки програми

	_cputs( "Enter:  1 - Start Thread1;  2 - Start Thread2; 3 - Exit\n");
	while (!stop)
	{
		a =  _getch_nolock();  /*Отримує символ з консолі без відлуння і блокування потоку*/

		switch (a)
		{
		case '1':
			{
				// якщо другий потік не завершений
				if (fContinue2)
				{
					fContinue2 = FALSE;
					//закриваємо визначник потоку 2
```

```
                        CloseHandle(hThread2);
                    }
                    DWORD dwThreadID;
                    fContinue1 = TRUE;

                    // створюємо новий потік 1
                    hThread1 = CreateThread (NULL, 0, FirstThread,
NULL, 0, &dwThreadID);
                }
                break;
            case '2':
                {
                    // якщо перший потік не завершений
                    if (fContinue1)
                    {
                        fContinue1 = FALSE;
                        //закриваємо визначник потоку 1
                        CloseHandle(hThread1);
                    }
                    DWORD dwThreadID;
                    fContinue2 = TRUE;

                    // створюємо новий потік 2
                hThread2 = CreateThread (NULL, 0, SecondThread, NULL,
0, &dwThreadID);
                }
                break;
            case '3':
                stop=TRUE;
                break;
            }
    }
    // видалення критичної секції
    DeleteCriticalSection(&g_cs);

    return 0;
}
```

Практичне заняття 8. Організація взаємодії між процесами

8.1. Способи передачі даних між процесами

Під обміном даними між паралельними процесами розуміється пересилка даних від одного потоку до іншого, припускаючи, що ці потоки виконуються в контекстах різних процесів.

Якщо потоки виконуються в одному процесі, то для обміну даними між ними можна використовувати глобальні змінні і засоби синхронізації потоків. Справа йде складніше у тому випадку, якщо потоки виконуються в різних процесах – потоки не можуть звертатися до загальних змінних і для обміну даними між ними існують спеціальні засоби операційної системи. Якщо говорити концептуально, то для обміну даними між процесами створюється канал передачі даних, організація якого схематично показана на рис. 8.1.

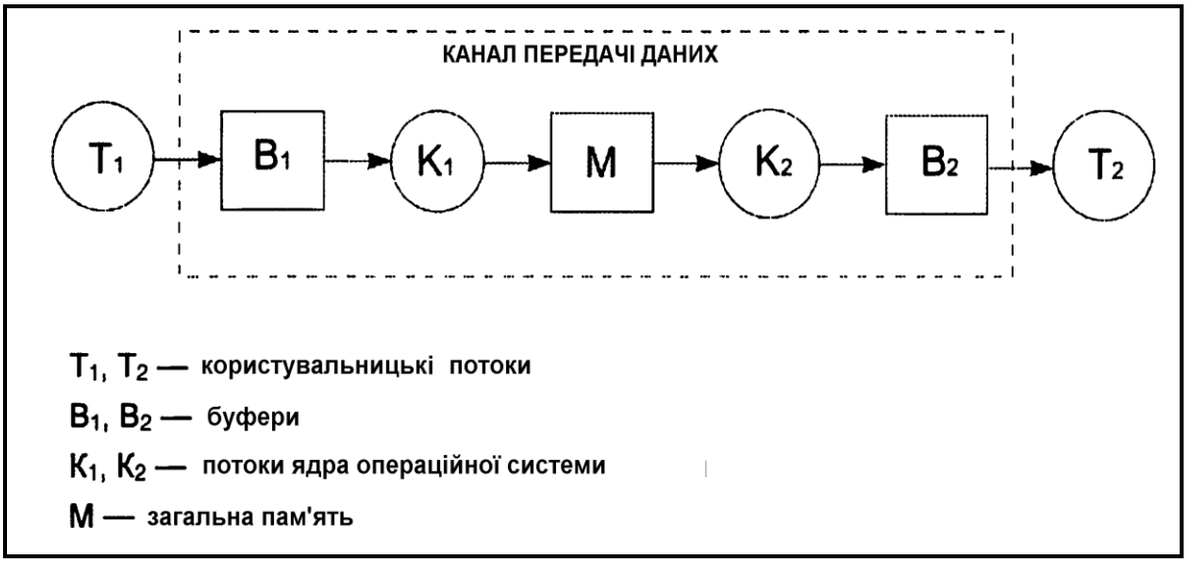

Рис. 8.1. **Схема каналу передачі даних**

Канал даних включає вхідний і вихідний буфери пам'яті, потоки ядра операційної системи і загальну пам'ять, доступ до якої мають обидва потоки ядра.

Працює канал передачі даних таким чином:

перший потік ядра операційної системи читає дані з вхідного буфера B1 і записує їх в загальну пам'ять M;

другий потік ядра читає дані із загальної пам'яті M і записує їх в буфер B2.

Призначені для користувача потоки T1 і T2 за допомогою виклику функцій ядра операційної системи мають доступ до буферів B1 і B2 відповідно. Тому пересилка даних з потоку T1 у потік T2 відбувається таким чином:

● призначений для користувача потік T1 записує дані в буфер B1, використовуючи спеціальну функцію ядра операційної системи;

● потік K1 ядра операційної системи читає дані з буфера B1 і записує їх у загальну пам'ять M;

● потік K2 ядра операційної системи читає дані із загальної пам'яті M і записує їх у буфер B2;

● призначений для користувача потік T2 читає дані з буфера B2.

8.2. Анонімні канали

Анонімним каналом називається об'єкт ядра операційної системи, який забезпечує передачу даних між процесами, що виконуються на одному комп'ютері. Процес, який створює анонімний канал, називається сервером анонімного каналу. Процеси, які зв'язуються з анонімним каналом, називаються клієнтами анонімного каналу. Іншими словами можна сказати, що анонімний – це такий канал передачі даних між процесами, який не має імені. Отже, доступ до такого каналу мають тільки батьківський процес-сервер і дочірні процеси-клієнти цього каналу.

Порядок роботи з анонімними каналами:

● створення анонімного каналу сервером;

● з'єднання клієнтів з каналом;

● обмін даними по каналу;

● закриття каналу.

Анонімні канали створюються процесом-сервером за допомогою функції **CreatePipe**, яка має такий прототип:

BOOL CreatePipe(

PHANDLE hReadPipe , // дескриптор для читання з каналу
PHANDLE hWritePipe, // дескриптор для запису в канал
LPSEOTRITY_ATTRIBUTES lpPipeAttributes, // атрибути захисту
DWORD dwSize // розмір буфера в байтах
);

При вдалому завершенні функція CreatePipe повертає ненульове значення, а у разі невдачі – FALSE. Розглянемо коротко призначення параметрів цієї функції.

У разі успішного завершення функція CreatePipe створює два дескриптори анонімного каналу – один для читання даних з каналу, а другий для запису даних у канал. Ці дескриптори повертаються в змінних, на які вказують параметри **hReadPipe** і **hWritePipe**. Дескриптор, на який вказує параметр hReadPipe, надалі використовується у функціях читання даних з каналу, а дескриптор hWritePipe - у функціях запису даних у канал.

Параметр **lpPipeAttributes** визначає атрибути захисту анонімного каналу.

Параметр **dwsize** визначає розмір буфера введення-виведення анонімного каналу. Значення цього параметра можна встановити рівним 0, тоді операційна система вибере розмір буфера за замовчуванням.

Оскільки анонімні канали не мають імені, то для з'єднання процесу – клієнта з таким каналом необхідно передати йому один з дескрипторів анонімного каналу. При цьому переданий дескриптор має бути успадкованим, а сам процес-клієнт має бути дочірнім процесом процесу сервера анонімного каналу і наслідувати успадковані дескриптори процесу-сервера.

Спадкоємство дескрипторів анонімного каналу визначається значенням поля bInheritHandie в структурі типу SECURITY_ATTRIBUTES, на яку вказує параметр lpPipeAttributes функції CreatePipe.

Для того щоб процес-клієнт наслідував дескриптори анонімного каналу він має бути створений функцією CreateProcess у процесі-сервері анонімного каналу і параметр bInheritHandies цієї функції має бути встановлений у TRUE.

Явна передача успадкованого дескриптора процесу-клієнтові анонімного каналу може виконуватися одним з таких способів:

● через командний рядок;

● через поля HSTDLNPUT, HSTDOUTPUT і HSTDERROR структури STARTUPINFO;

● за допомогою повідомлення WM_COPYDATA;

● через файл.

Обмін даними по анонімному каналу

Для обміну даними по анонімному каналу в операційних системах Windows використовуються ті ж функції, що для запису/читання даних у файл. Для запису даних в анонімний канал використовується функція **WriteFile**, яка має такий прототип:

```
BOOL WriteFile(
HANDLE hAnonymousPipe, // дескриптор анонімного каналу
LPCVOID lpBuffer, // буфер даних
DWORD dwNumberOfBytesToWrite, // кількість байтів для запису
LPDWORD lpNumberOfBytesWritten,//кількість записаних байтів
LPOVERLAPPED lpOverlapped // асинхронне введення
);
```

Функція WriteFile записує в анонімний канал кількість байтів, заданих параметром **dwNumberOfBytesToWrite**, з буфера даних, на який вказує параметр **lpBuffer**. Дескриптор виведення цього анонімного каналу має бути заданий першим параметром функції WriteFile. При успішному завершенні функція WriteFile повертає ненульове значення, а у разі невдачі – FALSE. Кількість байтів, записаних функцією WriteFile в анонімний канал, повертається в змінній, на яку вказує параметр **lpNumberOfBytesWritten**. Параметр **lpOverlapped** призначений для виконання асинхронної операції виведення, оскільки анонімні канали підтримують тільки синхронну передачу даних, то в нашому випадку цей параметр завжди буде рівний NULL.

Для читання даних з анонімного каналу використовується функція **ReadFile**, яка має такий прототип:

```
BOOL ReadFiie(
HANDLE hAnonymousPipe, // дескриптор анонімного каналу
LPCVOID lpBuffer, // буфер даних
DWORD dwNumberOfBytesToRead, // кількість байтів для читання
LPDWORD lpNumberOfBytesRead, // кількість прочитаних байтів
```

LPOVERLAPPED lpOverlapped // асинхронне введення
);

Функція ReadFile читає з анонімного каналу кількість байтів, заданих параметром **dwNumberOfBytesToRead**, у буфер даних, на який вказує параметр lpBuffer. Дескриптор введення цього анонімного каналу має бути заданий першим параметром функції ReadFile. При успішному завершенні функція ReadFile повертає ненульове значення, а у разі невдачі – FALSE. Кількість байтів, прочитаних функцією ReadFile з анонімного каналу, повертається в змінній, на яку вказує параметр **lpNumberOfBytesRead**. Також, як і у разі запису в анонімний канал, параметр **lpOverlapped** має бути рівний NULL.

Відмітимо, що обмін даними по анонімному каналу здійснюється тільки відповідно до призначення дескриптора цього каналу. Дескриптор для запису в анонімний канал має бути параметром функції WriteFile, а дескриптор для читання з анонімного каналу – параметром функції ReadFiie. У цьому і полягає сенс передачі даних по анонімному каналу тільки в одному напрямі. Проте це не означає, що один процес може використовувати анонімний канал тільки для запису або тільки для читання. Один і той же процес може як писати в анонімний канал, так і читати дані з нього, використовуючи дескриптори цього каналу.

Після завершення обміну даними по анонімному каналу потоки повинні закрити дескриптори запису і читання анонімного каналу, використовуючи функцію CloseHandle.

Приклад 8.1. Розглянемо приклад, в якому процес-сервер створює анонімний канал і дочірній процес, якому передає один з дескрипторів цього анонімного каналу. Для передачі дескриптора використовується командний рядок і, в цьому випадку, дочірній процес є клієнтом анонімного каналу. Для визначеності передамо клієнтові дескриптор для запису в анонімний канал і залишимо серверу дескриптор для читання.

Програма процесу-клієнта анонімного каналу:

```
#include "stdafx.h"
#include <windows.h>
#include <conio.h>
```

```
int _tmain(int argc, _TCHAR* argv[])
{
	// дескриптор каналу для запису
	HANDLE hWritePipe;
	// перетворимо символьне представлення дескриптора в число
	hWritePipe = (HANDLE) atoi(argv[1]);
	// чекаємо команди про початок запису в анонімний канал
	_cputs("Press any key to start communication.\n");
	_getch();
	// пишемо в анонімний канал
	for (int i = 0; i < 5; i++)
	{
		DWORD dwBytesWritten; // кількість записаних байтів
		//запис даних в анонімний канал
		if (!WriteFile(
			hWritePipe, //дескриптор анонімного каналу
			&&i, // буфер даних
			sizeof(i), // кількість байтів для запису
			&&dwBytesWritten, // кількість записаних байтів
			NULL)) // асинхронне введення
		{
			_cputs("Write to file failed.\n");
			_cputs("Press any key to finish.\n");
			_getch();
			return GetLastError();
		}
		_cprintf("The number %d is written to the pipe.\n", i);
		Sleep(500); //зупинка потоку на 500 мс
	}
	// закриваємо дескриптор каналу
	CloseHandle(hWritePipe);
	_cputs("The process finished writing to the pipe.\n");
	_cputs("Press any key to exit.\n");
	_getch();
	return 0;
}
```

Програма процесу-сервера анонімного каналу, який запускає клієнта і передає йому дескриптор запису в анонімний канал через командний рядок.

```
#include "stdafx.h"
#include <windows.h>
#include <conio.h>

int _tmain(int argc, _TCHAR* argv[])
{
	char lpszComLine[80]; // для командного рядка
	STARTUPINFO si;
	PROCESS_INFORMATION pi;

	// дескриптори для запису і читання
	HANDLE hWritePipe, hReadPipe, hlnWritePipe;

	// створюємо анонімний канал
	if(!CreatePipe(
		&hReadPipe, // дескриптор для читання
		&hWritePipe, // дескриптор для запису
		NULL, /* атрибути зашиті за замовчуванням, в цьому випадку
		дескриптори hReadPipe і hWritePipe неуспадковані*/
		0)) // розмір буфера за замовчуванням
	{
		_cputs("Create pipe failed.\n");
		_cputs("Press any key to finish.\n");
		_getch();
		return GetLastError();
	}
	// робимо успадкований дублікат дескриптора hWritePipe
	if(!DuplicateHandle(
		GetCurrentProcess(), //дескриптор процесу джерела
		hWritePipe, //початковий дескриптор каналу
		GetCurrentProcess(), //дескриптор процесу приймача
		&hlnWritePipe, //дублікат початкового дескриптора
```

```
        0, //прапори доступу до об'єкта ігноруються
        TRUE, // новий дескриптор успадкований
        DUPLICATE_SAME_ACCESS )) // доступ не змінюваний
{
        _cputs("Duplicate handle failed.\n");
        _cputs("Press any key to finish.\n");
        _getch();
        return GetLastError();
}
// закриваємо непотрібний дескриптор
CloseHandle(hWritePipe);
// встановлюємо атрибути нового процесу
// усі поля структури si типу startupinfo заповнюємо нулями
ZeroMemory(&si, sizeof(STARTUPINFO));
si.cb = sizeof(STARTUPINFO);

// формуємо командний рядок з іменем файла
wsprintf(lpszComLine, "D:\\pr.exe %d", (int) hlnWritePipe);
// запускаємо новий консольний процес
if (!CreateProcess(
        NULL, // ім'я процесу
        lpszComLine, // командний рядок
        NULL, // атрибути захисту процесу за замовчуванням
        NULL,//атрибути захисту первинного потоку за замовчуванням
        TRUE, /* успадковані дескриптори поточного процесу
        наслідують новим процесом*/
        CREATE_NEW_CONSOLE, // нова консоль
        NULL, //використовуємо середовище оточення процесу-предка
        NULL, // поточний диск і каталог, як і в процесі-предка
        &si, // вид головного вікна – за замовчуванням
        &pi // тут будуть дескриптори і ідентифікатори
        // нового процесу та його первинного потоку
        ))
{
        _cputs("Create process failed.\n");
        _cputs("Press any key to finish.\n");
        _getch();
```

```
            return GetLastError();
        }
        // закриваємо дескриптори нового процесу
        CloseHandle(pi.hProcess);
        CloseHandle(pi.hThread);
        // закриваємо непотрібний дескриптор каналу
        CloseHandle(hlnWritePipe);
        // читаємо з анонімного каналу
        for (int i = 0; i < 5; i++)
        {
            int nData;
            DWORD dwBytesRead;
            //читання даних з анонімного каналу
            if (!ReadFile(
                hReadPipe, // дескриптор анонімного каналу
                &&nData, // буфер даних
                sizeof(nData), // кількість байтів для запису

                &&dwBytesRead, // кількість записаних байтів
                NULL)) // асинхронне введення
            {
                _cputs("Read from the pipe failed. \n");
                _cputs("Press any key to finish.\n");
                _getch();
                return GetLastError();
            }
            _cprintf("The number %d is read from the pipe.\n", nData);
        }
        // закриваємо дескриптор каналу
        CloseHandle(hReadPipe);
        _cputs("The process finished reading from the pipe.\n");
        _cputs("Press any key to exit.\n");
        _getch();
        return 0;
}
```

Результати роботи програми представлені на рис. 8.2.

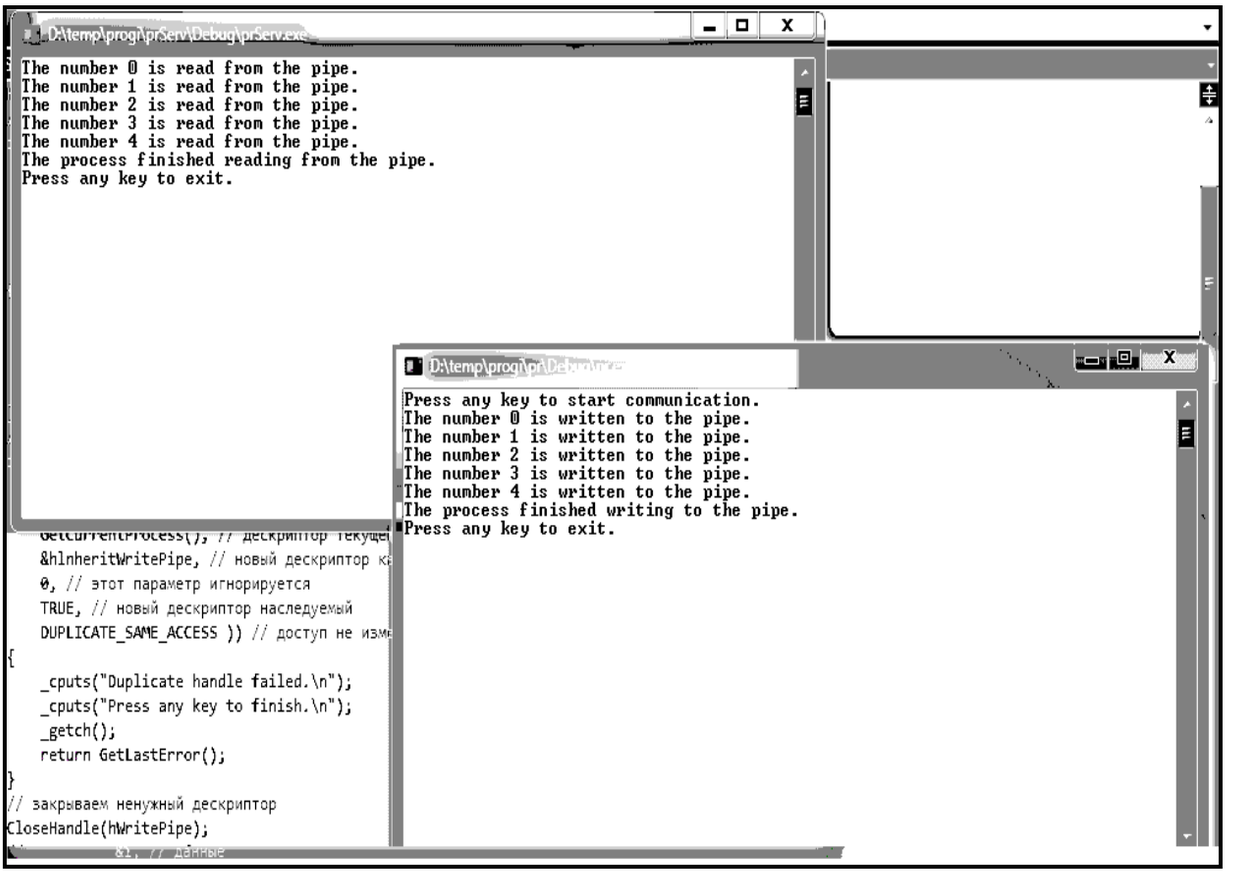

Рис. 8.2. **Ілюстрація роботи програми до приклада 8.1**

8.3. Іменовані канали

Іменованим каналом називається об'єкт ядра операційної системи, який забезпечує передачу даних між процесами, що виконуються на комп'ютерах в одній локальній мережі. Процес, який створює іменований канал, називається сервером іменованого каналу. Процеси, які зв'язуються з іменованим каналом, називаються клієнтами іменованого каналу.

Іменовані канали створюються процесом-сервером за допомогою функції CreateNamedPipe, яка має такий прототип:

```
HANDLE CreateNamedPipe(
LPCTSTR lpName,          // ім'я каналу
DWORD dwOpenMode, // атрибути каналу
DWORD dwPipeMode,   // режим передачі даних
DWORD nMaxInstances, //максимальна кількість екземплярів каналу
DWORD nOutBufferSize, // розмір вихідного буфера
DWORD nInBufferSize,   // розмір вхідного буфера
```

DWORD nDefaultTimeOut, // час очікування зв'язку з клієнтом
LPSECURITY_ATTRIBUTES lpPipeAttributes // атрибути безпеки
);

При вдалому завершенні функція CreateNamedPipe повертає дескриптор іменованого каналу, а у разі невдачі – одне з двох значень:

INVALID_HANDLE_VALUE – невдале завершення;

ERROR_INVALID_PARAMETR – значення параметра nMaxInstances більше, ніж величина pipe_unlimited_instances.

Опишемо параметри цієї функції.

Параметр lpName вказує на рядок, який повинен мати вигляд:

\\.\pipe\pipe_name

Тут "**.**" означає локальну машину, оскільки новий іменований канал завжди створюється на локальній машині, слово pipe – фіксоване, а pipe_name означає ім'я каналу, яке задається користувачем і нечутливе до регістра.

Параметр dwOpenMode задає прапори, які визначають напрям передачі даних, буферизацію, синхронізацію обміну даними і права доступу до іменованого каналу. Для визначення напряму передачі даних використовуються прапори:

PIPE_ACCESS_DUPLEX – читання і запис у канал;

PIPE_ACCESS_INBOUND – клієнт пише, а сервер читає дані;

PIPE_ACCESS_OUTBOUND – сервер пише, а клієнт читає дані.

Параметр dwPipeMode задає прапори, що визначають спосіб передачі даних по іменованому каналу. Для визначення способів читання і запису даних в іменований канал використовуються прапори:

PIPE_TYPE_BYTE – запис даних потоком;

PIPE_TYPE_MESSAGE – запис даних повідомленнями;

PYPE_READMODE_BYTE – читання даних потоком;

PYPE_READMODE_MESSAGE – читання даних повідомленнями.

За замовчуванням дані по іменованому каналу передаються потоком. Прапори, що визначають спосіб читання і запису даних в іменований канал, повинні співпадати для усіх екземплярів одного і того ж каналу. Для визначення синхронізації доступу до іменованого каналу використовуються прапори:

PIPE_WAIT – синхронний зв'язок з каналом і обмін даними;

PIPE_NOWAIT – асинхронний зв'язок з каналом і обмін даними.

Параметр nMaxinstances визначає максимальне число екземплярів іменованого каналу, яке може знаходитися в межах від 1 до pipe_unlimited_instances. Кожен екземпляр каналу призначений для обміну даними по каналу між сервером і окремим клієнтом.

Параметри nOutBufferSize і nInBufferSize визначають відповідно розміри вихідного і вхідного буферів для обміну даними по іменованому каналу. Проте ці значення розглядаються операційними системами Windows тільки як побажання користувача, а сам вибір розмірів буферів залишається за операційною системою.

Параметр nDefaultTimeOut встановлює час очікування клієнтом зв'язку з сервером. Цей час використовується при виклику клієнтом функції WaitNamedPipe, в якій параметр nTimeOut має значення NMPWAIT_USE_DEFAULT_WAIT.

Для зв'язку сервера з декількома клієнтами по одному іменованому каналу сервер повинен створити декілька екземплярів цього каналу. Кожен екземпляр іменованого каналу створюється викликом функції createNamedPipe, яка повертає дескриптор екземпляра іменованого каналу. Відмітимо, що в цьому випадку потік, що створює екземпляр іменованого каналу, повинен мати право доступу file_create_pipe_instance до іменованого каналу. Це право за замовчуванням має власник іменованого каналу, тобто той процес, який створив цей іменований канал.

З'єднання сервера з клієнтом

Після того, як сервер створив іменований канал, він повинен дочекатися з'єднання клієнта з цим каналом. Для цього сервер викликає функцію connectNamedPipe, яка має такий прототип:

```
BOOL ConnectNamedPipe (
HANDLE hNamedPipe, // дескриптор каналу
LPOVERLAPPED lpOverlapped // асинхронний зв'язок
);
```

У разі успішного завершення ця функція повертає ненульове значення, а у разі невдачі – значення FALSE. Сервер використовує цю функцію для зв'язку з клієнтом по кожному вільному екземпляру іменованого каналу.

Після закінчення обміну даними з клієнтом сервер може викликати функцію DisconnectNamedPipe, яка має такий прототип:

```
BOOL DisconnectNamedPipe (
    HANDLE hNamedPipe // дескриптор каналу
);
```

З'єднання клієнтів з іменованим каналом

Перш ніж з'єднуватися з іменованим каналом, клієнт повинен визначити, чи доступний який-небудь екземпляр цього каналу для з'єднання. З цією метою клієнт повинен викликати функцію WaitNamedPipe, яка має такий прототип:

```
BOOL WaitNamedPipe (
  LPCTSTR lpNamedPipeName, // покажчик на ім'я каналу
  DWORD nTimeOut // інтервал очікування
);
```

Параметр lpNamedPipeName вказує на рядок, який повинен мати вигляд:

\\server_name\pipe\pipe_name

Тут server_name означає ім'я комп'ютера, на якому виконується сервер іменованого каналу, слово pipe фіксоване, а pipe_name задає ім'я іменованого каналу.

Параметр nTimeOut задає часовий інтервал, протягом якого клієнт чекає зв'язок з сервером.

Після того, як виявлений вільний екземпляр каналу, щоб встановити зв'язок з цим каналом клієнт повинен викликати функцію CreateFile, яка має такий прототип:

```
HANDLE CreateFile (
  LPCTSTR lpFileName, // покажчик на ім'я каналу
  DWORD dwDesiredAccess, // читання або запис у канал
  DWORD dwShareMode, // режим спільного використання
  LPSECURITY_ATTRIBUTES lpSecurityAttributes, //атрибути безпеки
  DWORD dwCreationDisposition, // прапор відкриття каналу
  DWORD dwFlagsAndAttributes, // прапори і атрибути
```

HANDLE hTemplateFile // додаткові атрибути

);

У разі успішного завершення ця функція повертає дескриптор іменованого каналу, а у разі невдачі – зарезервоване значення INVALID_HANDLE_VALUE.

Якщо функція використовується для відкриття іменованого каналу, то її параметри можуть набувати наступних значень.

Параметр lpFileName повинен вказувати на ім'я каналу, яке має бути задане в тому ж форматі, що і у функції WaitNamedPipe. Відмітимо, що якщо клієнт працює на тій же машині, що і сервер, і використовує для відкриття іменованого каналу у функції createFile ім'я каналу як:

\\.\pipe\pipe_name, то файлова система іменованих каналів (Named Pipe File System, NPFS) відкриває цей іменований канал у режимі передачі даних потоком.

Щоб відкрити іменований канал у режимі передачі даних повідомленнями треба задавати ім'я каналу у вигляді:

\\server_name\pipe\pipe_name

Параметр dwDesiredAccess може приймати одне з таких значень:

0 – дозволяє отримати атрибути каналу;

GENERIC_READ – дозволяє читання з каналу;

GENERIC_WRITE – дозволяє запис у канал.

Слід зазначити, що функція CreateFile завершується невдачею, якщо доступ до іменованого каналу, заданий цими значеннями, не відповідає значенням параметра dwOpenMode у функції CreateNamedPipe.

Обмін даними по іменованому каналу

Як і у випадку з анонімним каналом, для обміну даними по іменованому каналу використовуються функції ReadFile і WriteFile, але з однією відмінністю, яка полягає в наступному. Оскільки у разі іменованого каналу дозволений асинхронний обмін даними, то у функціях ReadFile і WriteFile може використовуватися параметр lpOverLapped за тієї умови, що у виклику функції CreateNamedPipe в параметрі dwOpenMode був встановлений прапор FILE_FLAG_OVERLAPPED. Максимально в іменований канал може бути записано до 65 535 байтів однією операцією WriteFile. Для асинхронного

обміну даними по іменованому каналу можуть використовуватися також функції ReadFileEx і WriteFileEx.

Після завершення обміну даними по іменованому каналу потоки повинні закрити дескриптори екземплярів іменованого каналу, використовуючи функцію CloseHandle.

Розглянемо приклад, в якому процес-сервер створює іменований канал, а потім чекає, поки клієнт не з'єднається з іменованим каналом. Після цього сервер читає з іменованого каналу десять чисел і виводить їх на консоль.

Приклад 8.2. Програма процесу-сервера іменованого каналу.

```
#include "stdafx.h"
#include <windows.h>
#include <iostream>

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{

	HANDLE hNamedPipe;
	// створюємо іменований канал для читання
	hNamedPipe = CreateNamedPipe (
		"\\\\.\\pipe\\demo_pipe", // ім'я каналу
		PIPE_ACCESS_INBOUND, // читаємо з каналу
		PIPE_TYPE_MESSAGE | PIPE_WAIT, /*синхронна передача повідомлень*/
		1, // максимальна кількість екземплярів каналу
		0, // розмір вихідного буфера за замовчуванням
		0, // розмір вхідного буфера за замовчуванням
		INFINITE, // клієнт чекає зв'язок нескінченно довго
		NULL // захист за замовчуванням
		);
	// перевіряємо на успішне створення
	if (hNamedPipe == INVALID_HANDLE_VALUE)
	{
```

```
        cerr << "Create named pipe failed". << endl
        << "The last error code: " << GetLastError() <<endl;
        cout << "Press any key to exit".;
        cin.get();
        return 0;
}
// чекаємо, поки клієнт зв'яжеться з каналом
cout<<"The server is waiting for connection with a client"<<endl;
if(!ConnectNamedPipe(
        hNamedPipe, // дескриптор каналу
        NULL // зв'язок синхронний
        ))
{
        cerr << "The connection failed". << endl
        << "The last error code: " << GetLastError() << endl;

        // закриваємо дескриптор каналу
        CloseHandle(hNamedPipe);
        cout << "Press any key to exit".;
        cin.get();
        return 0;
}
// читаємо дані з каналу
for (int i = 0; i < 10; i++)
{
        int nData;
        DWORD dwBytesRead;
        if (!ReadFile(
                hNamedPipe, // дескриптор каналу
                &nData, // адреса буфера для введення даних
                sizeof(nData), // число читаних байтів
                &dwBytesRead, // число прочитаних байтів
                NULL // передача даних синхронна
                ))
        {
                cerr << "Read file failed". << endl
                <<"The last error code:"<< GetLastError() << endl;
```

```
                    // закриваємо дескриптор каналу
                    CloseHandle(hNamedPipe);

                    cout << "Press any key to exit".;
                    cin.get();
                    return 0;
               }
               // виводимо прочитані дані на консоль
               cout << "The number"<<nData << "was read by the server" <<
endl;
        }
        // закриваємо дескриптор каналу
        CloseHandle(hNamedPipe);

        // завершуємо процес
        cout << "The data are read by the server, " << endl;
        cout << "Press any key to exit".;
        cin.get();

        return 0;
}
```

Програма процесу-клієнта іменованого каналу, який спочатку зв'язується з іменованим каналом, а потім записує в нього десять чисел.

```
#include "stdafx.h"
#include <windows.h>
#include <iostream>

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
        HANDLE hNamedPipe;
        char pipeName [] = "\\\\.\\pipe\\demo_pipe";
        // зв'язуємося з іменованим каналом
        hNamedPipe = CreateFile(
```

```
pipeName, // ім'я каналу
GENERIC_WRITE, // записуємо в канал
FILE_SHARE_READ, // дозволяємо одночасне читання з каналу
NULL, // захист за замовчуванням
OPEN_EXISTING, // відкриваємо існуючий канал
0,// атрибути за замовчуванням
NULL // додаткових атрибутів немає
);
// перевіряємо зв'язок з каналом
if (hNamedPipe == INVALID_HANDLE_VALUE)
{
	cerr << "Connection with the named pipe failed" << endl
	<< "The last error code: " << GetLastError() << endl;
	cout << "Press any key to exit".;
	cin.get();
	return 0;
}
// пишемо в іменований канал
for (int i = 0; i < 10; i++)
{
	DWORD dwBytesWritten;
	if (!WriteFile(
		hNamedPipe, // дескриптор каналу
		&i, // дані
		sizeof(i), // розмір даних
		&dwBytesWritten, // кількість записаних байтів
		NULL // синхронний запис
		))

	{
		// помилка запису
		cerr <<"Writing to the named pipe failed:" << endl
		<<"The last error code:" << GetLastError()<< endl;

		// закриваємо дескриптор каналу
		CloseHandle(hNamedPipe);
```

```
                cout << "Press any key to exit".;
                cin.get();
                return 0;
            }
            // виводимо число на консоль
            cout << "The number" << i << "is written to the named pipe" <<
endl;
            Sleep(1000);
        }

        // закриваємо дескриптор каналу
        CloseHandle(hNamedPipe);
        // завершуємо процес
        cout << "The data are written by the client" << endl
        << "Press any key to exit".;
        cin.get();

        return 0;
}
```

Консольні вікна двох процесів наведені на скриншотах (рис. 8.3).

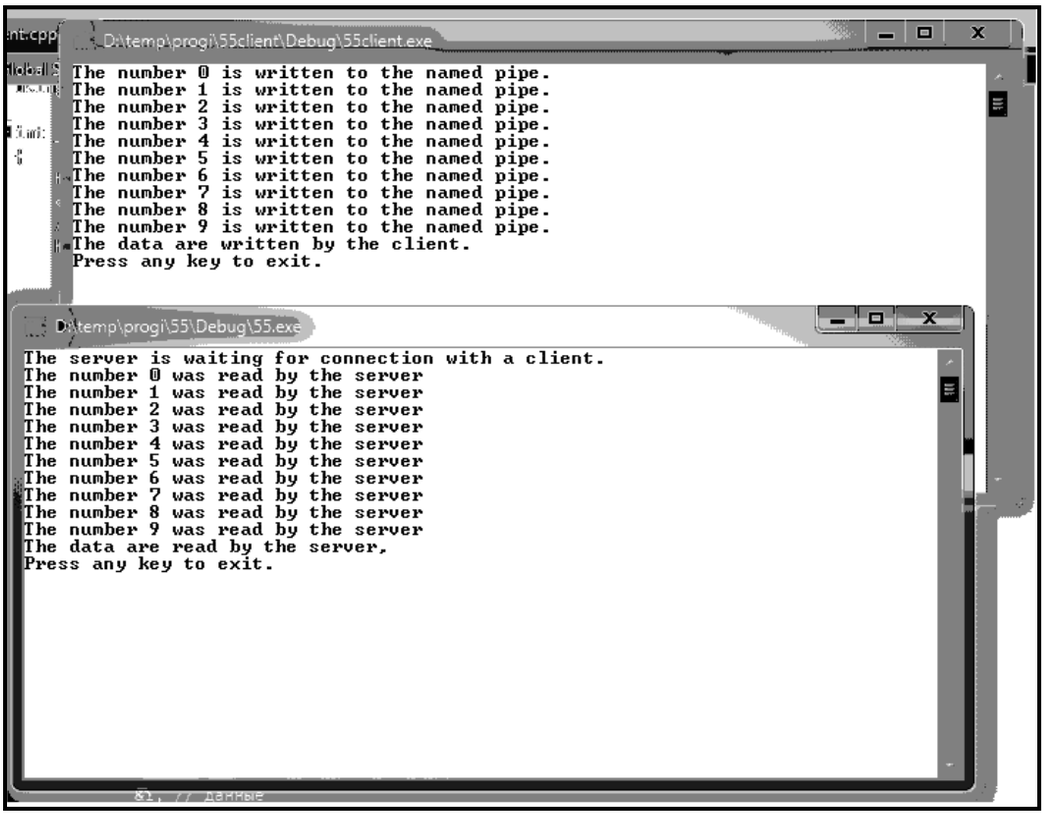

Рис. 8.3. **Ілюстрація роботи програми до прикладу 8.2**

Практичне заняття 9. Програмування на C++/CLI

Мова C++ є потужним інструментом розробки програм, яка справила величезний вплив на розвиток обчислювальної науки. Керовані (managed) розширення від Microsoft додали в мову C++ цілий новий світ – світ платформи .NET. Для того щоб повністю використовувати можливості Visual C++.NET, необхідно розуміти, як він працює з .NET Framework. Розгляд цих питань настільки складний та обширний, що виходить за рамки цього видання. Детально ці питання розглянуті в роботах [31, 33]. Тим не менше, деякі приклади будуть розглянуті.

Доречно, перед тим, як приступити до розбору та виконання цих прикладів, усвідомити, що середовище .NET Framework складається із двох елементів: загальномовного виконуючого середовища (Common Language Runtime – CLR), у якому виконуються програми, і набору бібліотек, що називаються бібліотеками класів .NET Framework. Бібліотека класів .NET Framework забезпечує функціональну підтримку, що необхідна розробленому коду програми при виконанні під керуванням

CLR, незалежно від застосовуваної мови програмування. Крім того, потрібно дати собі відповідь на такі запитання: поняття складки (Assembly); відображення C++ на специфікацію CLS; типи даних C++ і CLR; директива #using та оператор using; стандартне введення-виведення; керовані та некеровані типи даних; типова безпека; управління прибиранням сміття тощо.

Завдання 9.1. Написати програму на керованому C++, яка виводить на екран таблицю сумісних значень температур за Цельсієм та Фаренгейтом. Значення найменшої та найбільшої температури, а також кроку ввести з клавіатури.

```
#include "stdafx.h"
#define CNT_LINE 20  // Рекомендована кількість рядків у таблиці

using namespace System;

// Клас з методами для введення  значень з консолі //////////////////////
value class InputWrapper // клас "прибиральника сміття" InputWrapper
{
public:

//  Введення  цілого числа
int getInt(String ^pprompt) // pprompt - підказка на екрані
{
 Console::Write(pprompt);          // Виведення на екран підказки
 String ^pbuf = Console::ReadLine(); // Зчитування рядка з клавіатури
 return Convert::ToInt32(pbuf);      // Перетворення рядок-ціле число
}

//  Введення дійсного числа
double getDouble(String ^pprompt) // pprompt - підказка на екрані
{
 Console::Write(pprompt);                // Виведення на екран підказки
 String ^pbuf = Console::ReadLine(); // Зчитування рядка з клавіатури
 return Convert::ToDouble(pbuf);   // Перетворення рядок-дійсне число
```

```
}
//  Введення  десяткового числа
Decimal getDecimal(String ^pprompt) // pprompt - підказка на екрані
{
 Console::Write(pprompt);                // Виведення на екран підказки
 String ^pbuf = Console::ReadLine(); // Зчитування рядка з клавіатури
 return Convert::ToDecimal(pbuf); // Перетворення рядок-десяткове число
}
//  Введення рядка
String ^getString(String ^pprompt) // pprompt - підказка на екрані
{
 Console::Write(pprompt);                  // Виведення на екран підказки
 String ^pbuf = Console::ReadLine();      // Зчитування рядка з клавіатури
 return pbuf;
};

}; // End of InputWrapper


// Головна програма
int main(array<System::String ^> ^args)
{
  InputWrapper ^piw = gcnew InputWrapper;      // об'єкт класу InputWrapper
  // Мінімальна температура
  double minTemp = piw->getDouble("Введіть найнижчу температуру за
Цельсієм? ") ;
  double maxTemp = piw->getDouble("Введіть найбільшу температуру за
Цельсієм? ") ;
  double step = (maxTemp-minTemp)/CNT_LINE;
  Console::WriteLine ("Рекомендований крок= " +step.ToString()+" \n"); //
  while (true){
    String ^ans=piw->getString("Ви згодні? (y/n) ");
        if (ans=="y") break;
        if (ans!="n") {Console::WriteLine ("Ви помилилися. Повторіть!\n");
continue; }
    step = piw->getDouble("Введіть значення кроку? ") ;
  }
```

```
 Console::WriteLine ("================================");
 Console::WriteLine ("|T (Цельсія) | T (Фаренгейта) |");
 Console::WriteLine ("================================");
 for (double t = minTemp; t <= maxTemp; t+=step){
  double fahr = t * 9 / 5 + 32;                    // Цельсій – Фаренгейт
  Console::Write("| {0,6:F2}     |", t);        // Виведення (за Цельсієм)
  Console::WriteLine (" {0,9:F3}       |", fahr);   // Виведення (за Фаренгейтом)
 }
Console::WriteLine ("================================");
Console::ReadLine();
return 0;
}
```

Результати роботи програми представлені на рис. 9.1.

```
Введіть найнижчу температуру за за Цельсієм? 0
Введіть найбільшу температуру за за Цельсієм? 140
Рекомендований крок= 7

Ви згодні? (y/n) t
Ви помилилися. Повторіть!

Ви згодні? (y/n) n
Введіть значення кроку? 6
Ви згодні? (y/n) y
================================
¦T (Цельсія) ¦ T (Фаренгейта) ¦
================================
¦   0,00     ¦    32,000       ¦
¦   6,00     ¦    42,800       ¦
¦  12,00     ¦    53,600       ¦
¦  18,00     ¦    64,400       ¦
¦  24,00     ¦    75,200       ¦
¦  30,00     ¦    86,000       ¦
¦  36,00     ¦    96,800       ¦
¦  42,00     ¦   107,600       ¦
¦  48,00     ¦   118,400       ¦
¦  54,00     ¦   129,200       ¦
¦  60,00     ¦   140,000       ¦
¦  66,00     ¦   150,800       ¦
¦  72,00     ¦   161,600       ¦
¦  78,00     ¦   172,400       ¦
¦  84,00     ¦   183,200       ¦
¦  90,00     ¦   194,000       ¦
¦  96,00     ¦   204,800       ¦
¦ 102,00     ¦   215,600       ¦
¦ 108,00     ¦   226,400       ¦
¦ 114,00     ¦   237,200       ¦
¦ 120,00     ¦   248,000       ¦
¦ 126,00     ¦   258,800       ¦
¦ 132,00     ¦   269,600       ¦
¦ 138,00     ¦   280,400       ¦
================================
```

Рис. 9.1. **Ілюстрація роботи програми до прикладу 9.1**

Завдання 9.2. Написати програму перетворення квадратної матриці таким чином, щоб елементи нової матриці дорівнювали сумі елементів початкової матриці, які оточують поточний елемент. Підрахувати кількість помилок у разі виходу індекса масиву за допустимий діапазон.

```
#include "stdafx.h"
#define COL      4   // кількість стовпчиків
#define ROW      3   // кількість рядків
using namespace System;
void main ()
{
// керовані масиви з 4x3 елементів
array<int,2>^ A = {{5,0,1,7},{1,2,1,4},{3,0,0,2}};  // ініціалізований масив
array<int,2>^ B = gcnew array<int,2> (ROW, COL);    // неініціалізований
масив
int cnt_err=0;   // кількість помилок при перетворенні масиву
int i, i1, j, j1;   // змінні циклу (рядки та стовпчики)

// Перебираємо елементи початкового масиву A
for (i=0; i<ROW; i++)
{
       for (j=0; j<COL; j++) {
         B[i,j]=0; // обнуляємо елемент, що формується
           for (i1=i-1; i1<=i+1; i1++)
                 for (j1=j-1; j1<=j+1; j1++){
                     try
                     {
                       B[i,j]+=A[i1,j1];  // накопичуємо суму оточуючих
елементів
                     }
          // у випадку помилки ( індекс масиву!!!)
          catch (IndexOutOfRangeException^ piore)
               {
                 // повідомлення про виняткову ситуацію
                  Console::WriteLine("Error i= {0,2}  j={1,2} ",i1,j1);
```

```
            cnt_err++;
          }
              } // for j1
        } // for j
} // for i

// Виводимо початкову матрицю
Console::WriteLine("\nПочаткова матриця:");
for (i=0; i<ROW; i++) {
    for (j=0; j<COL; j++) {
        Console::Write("{0,3}",A[i,j]);
 }
 Console::WriteLine();
}
// Виводимо трансформовану матрицю
Console::WriteLine("\nТрансформована матриця:");
for (i=0; i<ROW; i++) {
 for (j=0; j<COL; j++) {
        Console::Write("{0,3}",B[i,j]);
 }
 Console::WriteLine();
}
 // Виводимо кількість помилок
 Console::WriteLine("Кількість помилок= {0}",cnt_err);
 Console::ReadLine();
}
```

Результати роботи програми представлені на рис. 9.2.

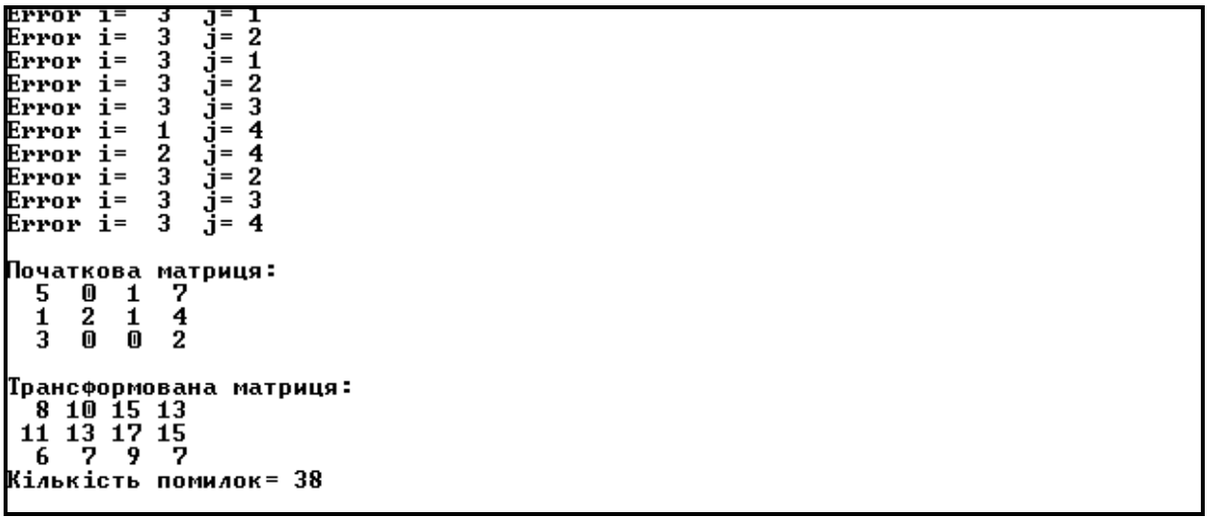

Рис. 9.2. **Ілюстрація роботи програми до прикладу 9.2**

Контрольні завдання для самостійного вирішення

Завдання 10.1. В одновимірному масиві, що складається з n цілих елементів, обчислити: номер елементу, який найменше відрізняється від середнього значення елементів масиву; добуток елементів масиву, розташованих між першим і другим нульовими елементами. Перетворити масив так, щоб спочатку йшов мінімальний елемент, за ним – максимальний. Далі – другий найменший (тобто той, який більше ніж найменший, але менший ніж всі інші) та другий найбільший і т. д. Замість класичного доступу до елементів масиву і виконання операцій над елементами використовувати покажчики. Обробку масиву виконати у функції. Функцію в головній програмі викликати через покажчик. Пам'ять під масив виділити динамічно.

Завдання 10.2. Написати програму, яка: читає текст із файла та виводить його; визначає кількість символів у щонайдовшому слові; після цього при натисненні клавіші <Space> по черзі виділяє кожне слово тексту, що містить максимальну кількість символів.

Завдання 10.3. Змінити програму до завдання 10.2 таким чином, щоб у програмі використовувалися макровизначення та була умовна компіляція. Проект повинен бути багатофайловим.

Завдання 10.4. Написати програму, яка обчислює дані про об'єм

реалізації продукції в натуральному і вартісному виразах. Структуру даних треба розробити самостіно й обґрунтувати вибір. Передбачити збереження даних у файл та їх зворотнє завантаження. Додатково на консолі намалювати стовпчикову діаграму, яка характеризує загальний об'єм реалізації продукції залежно від її типу.

Завдання 10.5. За допомогою бібліотеки STL написати програму, що дозволяє вести каталог та здійснювати пошук інформації про вміст та призначення програмних засобів, що зберігаються на CD – або DVD–дисках. Каталог повинен зберігатися на зовнішніх носіях.

Завдання 10.6. Написати простий Windows-додаток, що при запуску створює вікно, яке повинно активізуватися й відображатися в заданих розмірах і позиції; бути дочірнім зі спадкуванням контексту батьківського вікна; завантажувати іконку у вигляді довільного головного убору; мати курсор у вигляді стрілки зі знаком питання; мати темно-зелений колір фону; розташовуватися посередині екрана та мати висоту в 200 пікселів, а ширину в половину екрана.

Завдання 10.7. Написати Windows-додаток, який дозволяє малювати на екрані графік довільної функції, яка вибирається зі списку. Діапазон зміни аргументу функції вводити у діалозі. Додатково передбачити зміну коефіцієнтів у аналітичному виразі фінкції. Як розширення функціонального призначення програми розробити інтерфейс, який дозволяє малювати графік функції або графічно відображати поведінку процесу, значення якого зберігаються у файлі. Передбачити можливість введення подібних даних за допомогою окремого діалогового вікна.

Рекомендована література

1. Арсак Р. Программирование игр и головоломок / Арсак Р. – М., Мир, 1994. – 224 с.

2. Ахо А. Построение и анализ вычислительных алгоритмов / Ахо А., Хопкрофт Дж., Ульман Дж. – М. : Мир, 1979. – 536 с.

3. Бентли Дж. Жемчужины программирования / Бентли Дж. – СПб. :

Питер, 2002. – 272 с.

4. Брудно А. Л. Московские олимпиады по программированию / Брудно А. Л., Каплан Л. И. ; под. ред. акад. Б. Н. Наумова. – М. : Наука, 1990. – 208 с.

5. Виленкин Н. Я. Популярная комбинаторика / Виленкин Н. Я. – М. : Наука, 1975. – 208 с.

6. Вирт Н. Алгоритмы и структуры данных / Вирт Н. ; пер. с англ. – М. : Мир, 1989. – 360 с.

7. Глушаков С. В. Язык программирования С++ / Глушаков С. В., Коваль А. В., Смирнов С. В. – Харьков : Фолио, 2001. – 500 с.

8. Грегори К. Использование Visual C++ 6 / Грегори К. ; пер. с англ. – М. ; СПб. ; К. : Издательский дом "Вильямс", 2000. – 864 с.

9. Дейтел Х. Как программировать на С++ / Дейтел Х., Дейтел П. ; пер. с англ. – М. : ЗАО "Издательство БИНОМ", 1998. – 1024 с.

10. Джамса К. Учимся программировать на языке С++ / Джамса К. ; пер. с англ. – М. : Мир, 1997. – 320 с.

11. Жарков В. Visual C++ 2008 в учебе, науке и технике / Жарков В. – М. : Жарков Пресс, 2009. – 814 с.

12. Задачи по программированию / Абрамов С. А., Гнездилова Г. Г., Капустина Е. Н., Селюн М. И. – М. : Наука, 1988. – 224 с.

13. Зелковиц М. / Принципы разработки программного обеспечения / Зелковиц М., Шоу А., Гэннон Дж. ; пер. с англ. – М. : Мир, 1982. – 386 с.

14. Керниган Б. Язык программирования Си. Задачи по языку Си / Керниган Б., Ритчи Д., Фьюер А. – М. : Финансы и статистика, 1985, – 279 с.

15. Кнут Д. Искусство программирования для ЭВМ, Т.1 – 3 / Кнут Д. – М. : Мир, 1976 – 1977. – Т.1 – 736 с. ; Т.2 – 724 с. ; Т.3 – 844 с.

16. Кристофидес Н. Теория графов / Кристофидес Н. – М. : – Мир, 1979. – 215 с.

17. Крячков А. В. Программирование на С и С++ / Крячков А. В., Сухинина И. В., Томшин В. К. – М. : Горячая линия-Телеком, 2000. – 344 с.

18. Культин Н. Microsoft Visual C++ в задачах и примерах / Культин Н. – СПб. : BHV, 2010. – 274 с.

19. Марченко А. Л. С++. Бархатный путь / Марченко А. Л. – М. : Горячая линия-Телеком, 1999. – 400 с.

20. Мейерс С. Эффективное использование С++ / Мейерс С. ; пер. с англ. – М. : ДМК, 2000, – 236 с.

21. Методичні рекомендації до виконання практичних завдань з навчальної дисципліни "Основи програмування та алгоритмічні мови" / укл. Тарасов О. В., Федорченко В. М. Лосєв М. Ю. – Харків : ХНЕУ, 2010. – 90 с.

22. Остерн М. Г. Обобщенное программирование и STL: Использование и наращивание стандартной библиотеки шаблонов C++ / Остерн М. Г. ; пер. с англ. – СПб. : Невский Диалект, 2004. – 544 с.

23. Павловская Т. А. C/C++. Программирование на языке высокого уровня / Павловская Т. А. – СПб. : Питер, 2003. – 461 с.

24. Пахомов Б. C/C++ и MS Visual C++ 2010 для начинающих / Пахомов Б. – СПб. : БХВ-Петербург, 2010. – 624 с.

25. Подбельский В. В. Программирование на языке Си / Подбельский В. В., Фомин С. С. – М. : Финансы и статистика, 2004. – 600 с.

26. Подбельский В. В. Язык C++ : учебное пособие / Подбельский В. В. – М. : Финансы и статистика, 1995. – 560 с.

27. Страуструп Б. Дизайн и эволюция языка C++ / Страуструп Б. ; пер. с англ. – М. : ДМК, 2000. – 444 с.

28. Страуструп Б. Язык программирования C++ / Страуструп Б. ; пер. с англ. – СПб. ; М. : БИНОМ, 1999. – 991 с.

29. Фокс Дж. Программное обеспечение и его разработка / Фокс Дж.; пер. с англ. – М. : Мир, 1985. – 368 с.

30. Фридман А. Л. Основы объектно-ориентированного программирования на языке C++. Учебный курс / Фридман А. Л. – М. : Радио и связь, 1999. – 205 с.

31. Хогенсон Г. C++/CLI. Язык Visual C++ для среды .NET / Хогенсон Г. ; пер. с англ.– М. : – Вильямс, 2007. – 464 с.

32. Холл М. Введение в комбинаторику / Холл М. – М. : Мир, 1970. – 424 с.

33. Хортон А. Visual C++ 2005: базовый курс / Хортон А. ; пер. с англ. – М. : Вильямс, 2007. – 1152 с.

34. Шилдт Г. Самоучитель C++ / Шилдт Г. ; пер. с англ. – СПб. : BHV-Санкт-Петербург, 1998. – 620 с.

35. Щупак Ю. А. Win32 API. Эффективная разработка приложений / Щупак Ю. А. – СПб. : Питер, 2007. – 572 с.

36. Эллис М. Справочное руководство по языку C++ с комментариями / Эллис М., Страуструп Б. ; пер. с англ. – М. : Мир, 1992. – 445 с.

Додатки

Додаток А

Функції роботи з рядками у С++

Існують два варіанти роботи з текстовою інформацією. Перший – це скористатися С-рядками – масивами символів, що завершуються нульовим символом. Для їх використання необхідно включити файл string.h або string. Для функцій з префіксом str є аналогічні функції з префіксом wcs для рядків з символами типу wchar_t. Функції пошуку повертають покажчик на позицію знайденого елемента або 0, якщо нічого не знайдено. Для аргументів, правильні такі правила: копіювання, додавання (куди, звідки), пошук (де, що). Функції не обробляють коректно нульові аргументи. Перелік основних функцій роботи з рядками у мові С/С++ наведено в табл. А.1

Таблиця А.1.

Перелік базових функцій роботи з рядками у С/С++

Функція	Пояснення
1	2
memset (void*, int, size_t)	встановлює n перших байтів у вказане значення
strcat(char*, const char*)	додає один рядок до іншого
strchr(const char*, int)	шукає перше входження зазначеного символа в рядку
strcmp(const char*, const char*)	порівнює два рядки (0 - рівні рядки, <0 – перший рядок менше ніж другий, >0 – перший рядок більше за другого)
strcpy(char*, const char*)	копіює один рядок в інший
strcspn(const char*, const char*)	шукає перше входження одного з символів одного рядка в інший (повертається позиція)
strlen(const char*)	повертає довжину рядка (без нульового символа кінця рядка)
strncat(char*, const char*, size_t)	додає n символів одного рядка до іншого
strncmp(const char*, const char*, size_t)	порівнює n перших символів
strncpy(char*, const char*, size_t)	копіювати n перших символів одного рядка в інший

Закінчення табл. А.1

1	2
strpbrk(const char*, const char*)	шукає перше входження одного з символів одного рядка в інший
strrchr(const char*, int)	пошук символа з кінця
strspn(const char*, const char*)	пошук першого символа що не входить у вказаний рядок (повертається номер першого символа)
strstr(const char*, const char*)	пошук першого входження підрядка
strtok(char*, const char*)	повертає наступний токен (елемент розбору)

Другий варіант – це скористатися готовим класом, що реалізує рядок, наприклад string з STL. Перелік основних функцій (методів) роботи з рядками string наведено в табл. А.2.

Таблиця А.2.

Перелік методів роботи з рядками string

Функція	**Пояснення**
1	2
append	додає рядок або символи до рядка
assign	присвоює рядку значення рядків символів або інших рядків C++
at	повертає символ, що стоїть у деякій позиції
begin	повертає ітератор на початок рядка
c_str	повертає рядок у вигляді немодифікованого масиву символів (як у C)
capacity	повертає кількість символів, які можуть поміститися в рядок
clear	видаляє всі символи з рядка
compare	порівнює два рядки
copy	копіює символи з рядка в масив
empty	повертає true, якщо в рядку немає символів
end	повертає ітератор, встановлений після останнього символа рядка
erase	видаляє символи з рядка
find	шукає символи в рядку
find_first_not_of	знаходить перший символ, відмінний від…
find_first_of	знаходить перший символ, схожий з …
find_last_not_of	знаходить останній символ, відмінний від …

Закінчення табл. А.2

1	2
find_last_of	знаходить останній символ, схожий з …
getline	читає з потоку введення в рядок
insert	вставляє символи в рядок
push_back	додає символ у кінець рядка
rbegin	повертає зворотний ітератор на кінець рядка
rend	повертає зворотний ітератор на початок рядка
replace	замінює символи в рядку
rfind	знаходить останнє входження підрядка
size	повертає кількість символів у рядку
substr	повертає певний підрядок

Додаток Б

Функції для роботи з файлами

Роботу з файлами у програмах на C++ частіше за все організують або за допомогою стандартних функцій мови С, або використовуючи потокові класи C++ . Перелік базових функцій роботи з файлами у мові С наведений у табл. Б.1.

Таблиця Б.1

Перелік базових функцій роботи з файлами у мові С/С++

Функція	Пояснення
fopen	функція для відкриття файла
fwrite	функція для запису інформації у файл
fputs	функція для запису рядка у файл
fclose	функція для закриття файла
fread	функція для зчитування даних з файла
feof	функція, яка повертає true, при досягненні кінця заданого файла, і false – у протилежному випадку
fgetc	читання символа з файла
fgets	читання рядка з файла
fseek	переміщення покажчика файла в потрібне місце
ftell	повертає зсув файлового покажчика
rewind	зміщення покажчика в початок файла

Бібліотека потокових класів C++ побудована на основі двох базових класів: **ios** і **streambuf**. Клас **streambuf** програміст зазвичай не використовує. Цей клас потрібен для інших класів бібліотеки введення-виведення. Клас **ios** містить засоби для форматованого введення-виведення і перевірки помилок. Похідні від нього класи **ifstream, ofstream, fstream** служать для роботи з файлами. Потокові класи, їхні методи і дані стають доступними в програмі, якщо в неї включений потрібний заголовний файл:

1) **iostream.h –** для **ios, ostream, istream;**
2) **strstream.h –** для **strstream, istrstream, ostrstream;**
3) **fstream.h – для fstream, ifstream, ofstream.**

Для введення даних з потоку використовуються об'єкти класу **istream**, для виведення у потік - об'єкти класу **ostream**.

У класі **istream** визначені такі функції:

istream &get(char *buffer, int size, char delimiter = '\n'); – ця функція витягує символи з istream і копіює їх у буфер. Операція припиняється при досягненні кінця файла, або коли скопійовані size символів, або при виявленні вказаного роздільника. Сам роздільник не копіюється і залишається в streambuf. Послідовність прочитаних символів завжди завершується нульовим символом;

istream &read(char *buffer, int size); – не підтримує роздільників, і зчитані в буфер символи не завершуються нульовим символом;

istream &getline(char *buffer, int size, char delimiter =\n'); – роздільник витягується з потоку, але в буфер не заноситься. Це основна функція для вилучення рядків з потоку. Зчитані символи завершуються нульовим символом;

istream &get(streambuf &s, char delimiter = '\n'); – копіює дані з istream в streambuf до тих пір, поки не виявить кінець файла або символ-роздільник, який не вилучають із istream. У s нульовий символ не записується;

istream get (char & C); – читає символ з istream в C. У разі помилки C бере значення 0XFF;

int get (void); – витягує з istream черговий символ. При виявленні кінця файла повертає EOF;

int peek (void); – повертає черговий символ з istream, не витягуючи його з istream;

int gcount (void); – повертає кількість символів, прочитаних під час останньої операції неформатованого введення;

istream & ignore (int count = 1, int target = EOF); – витягує символ з istream, поки не відбудеться наступне: функція не отримає count символів; не буде виявлений символ target; не буде досягнуто кінця файла.

У класі **ostream** визначені такі функції:

ostream & put (char C); – розміщує у ostream символ C;

ostream & write (const char * buffer, int size); – записує в ostream вміст буфера. Символи копіюються до тих пір, поки не виникне помилка

або не буде скопійовано size символів. Буфер записується без формату

вання. Обробка нульових символів нічим не відрізняється від обробки інших. Дана функція здійснює передачу необроблених даних (бінарних або текстових) в ostream;

ostream & flush (void); – скидає буфер streambuf.

Для прямого доступу використовуються такі функції установки позиції читання-запису. При читанні:

istream & seekg (long p); – встановлює покажчик потоку get (не плутати з функцією) зі зміщенням p від початку потоку;

istream & seekg (long p, seek_dir point); – вказується початкова точка переміщення **enum seek_dir {beg, curr, end}.** Позитивне значення p переміщає покажчик get вперед (до кінця потоку), від'ємне значення p – назад (до початку потоку);

long tellg (void); – повертає поточне положення покажчика get.

Під час запису:

ostream & seekp (long p); – переміщає покажчик put в streambuf на позицію p від початку буфера streambuf;

ostream & seekp (long p, seek_dir point); – вказує точку відліку;

long tellp (void); – повертає поточне положення покажчика put.

Додаток В

Типи даних у Win32 API

У Windows визначена велика кількість типів даних, які використовуються для специфікації типів параметрів та типів результатів функцій у Windows API. У табл. В.1 наведені деякі з цих типів, які часто зустрічаються у Windows-програмах.

Табл. В.1 містить такі типи: символ (Charaster), ціле число (Integer), булеве значення (Boolean), покажчик (Pointer) і дескриптор (Handle). Символ, ціле число і булеві типи загальні для більшості компіляторів C/C++. Більшість імен типу покажчика починається з префікса P або LP. Win32-based-додаток використовує дескриптор, щоб звертатися до ресурсу, який був завантажений у пам'ять.

Таблиця В.1

Типи даних у Win32 API

Тип	Визначення
1	2
ATOM	Атом (посилання до символьного рядка в таблиці атома)
BOOL	Булева змінна (повинна бути істинною чи хибною
BOOLEAN	Булева змінна (повинна бути істинною чи хибно.
BYTE	Байт (8 bits)
CALLBACK	Угода про виклики для функцій зворотного виклик
CHAR	8-розрядний символ (ANSI) Window
CONST	Змінна, чиє значення має залишитися константою протягом виконання
DWORD	32-розрядне ціле число без знака
FLOAT	Змінна з плаваючою комою
HACCEL	Дескриптор таблиці акселератора
HANDLE	Дескриптор об'єкта
HBITMAP	Дескриптор растра
HBRUSH	Дескриптор кисті
HCURSOR	Дескриптор курсора
HDC	Дескриптор контексту пристрою
HFONT	Дескриптор шрифту
HINSTANCE	Дескриптор зразка
HKEY	Дескриптор ключа реєстру
HMENU	Дескриптор меню
HPALETTE	Дескриптор палітри
HPEN	Дескриптор пера
HWND	Дескриптор вікна
INT	Ціле число зі знаком
LONG	32-розрядне ціле число зі знаком

Продовження табл. В.1

1	2
LPARAM	32-розрядний параметр повідомлення
LPBOOL	Покажчик на BOOL
LPBYTE	Покажчик на БАЙТ
LPCSTR	Покажчик на постійний рядок з нульовим символом у кінці 8-розрядних символів (ANSI) Windows
LPCTSTR	LPCWSTR, якщо UNICODE визначено, інакше LPCSTR
LPCVOID	Покажчик на константу будь-якого типу
LPCWSTR	Покажчик на постійний рядок з нульовим символом у кінці 16-розрядних символів Unicode
LPDWORD	Покажчик на DWORD
LPHANDLE	Покажчик на ДЕСКРИПТОР
LPINT	Покажчик на INT
LPLONG	Покажчик на LONG
LPSTR	Покажчик на рядок з нульовим символом у кінці 8-розрядних символів (ANSI) Windows
LPSTREAM	Покажчик на потік (даних)
LPTSTR	LPWSTR, якщо UNICODE визначено, інакше LPSTR
LPVOID	Покажчик на будь-який тип
LPWORD	Покажчик на СЛОВО
LPWSTR	Покажчик на рядок з нульовим символом у кінці 16-розрядних символів Unicode
LRESULT	Результат обробки повідомлення
LUID	Локально унікальний ідентифікатор
PBOOL	Покажчик на BOOL
PBOOLEAN	Покажчик на BOOL
PBYTE	Покажчик на БАЙТ
PCHAR	Покажчик на CHAR
PCSTR	Покажчик на постійний рядок з нульовим символом у кінці 8-розрядних символів (ANSI) Windows
PHANDLE	Покажчик на ДЕСКРИПТОР
PINT	Покажчик на INT
PLCID	Покажчик на LCID
PLONG	Покажчик на LONG
PSHORT	Покажчик на SHORT
PSTR	Покажчик на рядок з нульовим символом у кінці 8-розрядних символів (ANSI) Windows
PTBYTE	Покажчик на TBYTE
PTCHAR	Покажчик на TCHAR
PVOID	Покажчик на будь-який тип
SHORT	Коротке ціле число
TBYTE	WCHAR, якщо UNICODE визначено, інакше CHAR
TCHAR	WCHAR, якщо UNICODE визначено, інакше CHAR
UCHAR	CHAR без знака

Закінчення табл. В.1

1	2
UINT	INT без знака
ULONG	LONG без знака
UNSIGNED	Атрибут без знака
USHORT	SHORT без знака
VOID	Будь-який тип
WCHAR	16-розрядний символ Unicode
WINAPI	Угода про виклики для Win32 API
WNDPROC	Покажчик на визначену додатком процедуру вікна
WORD	16-розрядне ціле число без знака
WPARAM	32-розрядний параметр повідомлення

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

Методичні рекомендації до практичних завдань
з модуля "Принципи розробки Windows-додатків"
навчальної дисципліни

"ОСНОВИ ПРОГРАМУВАННЯ ТА АЛГОРИТМІЧНІ МОВИ"

для студентів напряму підготовки
"Комп'ютерні науки"
всіх форм навчання

Укладачі: **Федорченко** Володимир Миколайович
Тарасов Олександр Васильович
Гриньов Денис Валерійович

Відповідальний за випуск **Пономаренко В. С.**

Редактор **Семенова І. М.**

Коректор **Даценко Л. О.**

План 2011 р. Поз. № 201.
Підп. до друку Формат 60 х 90 1/16. Папір MultiCopy. Друк Riso.
Ум.-друк. арк. 9,75. Обл.-вид. арк. 12,19. Тираж прим. Зам. №
Видавець і виготівник — видавництво ХНЕУ, 61001, м. Харків, пр. Леніна, 9а
Свідоцтво про внесення до Державного реєстру суб'єктів видавничої справи ***Дк № 481 від 13.06.2001 р***.