МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Методичні рекомендації
до виконання лабораторних робіт
з навчальної дисципліни

"ОСНОВИ ПРОГРАМУВАННЯ ТА АЛГОРИТМІЧНІ МОВИ"

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

Частина 2

Харків. Вид. ХНЕУ, 2009

Затверджено на засіданні кафедри інформаційних систем.
Протокол №7 від 06.02.2009 р.

М54 Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "Основи програмування та алгоритмічні мови" для студентів напряму підготовки "Комп'ютерні науки" всіх форм навчання. Ч. 2 / Укл. М. Ю. Лосєв, Ю. Е. Парфьонов, В. М. Федорченко, О. В. Щербаков. – Харків: Вид. ХНЕУ, 2009. – 180 с. (Укр. мов.)

Подано методичні рекомендації до виконання лабораторних робіт з даної навчальної дисципліни; наведено лекційний матеріал і загальні положення, які передують опису лабораторних завдань.

Рекомендовано для студентів напряму підготовки "Комп'ютерні науки".

Загальні положення

Методичні рекомендації призначені для виконання лабораторних робіт з другої частини навчальної дисципліни "Основи програмування та алгоритмічні мови".

Перед виконанням кожної роботи необхідно вивчити відповідний лекційний матеріал і звернути особливу увагу на загальні положення, які передують опису лабораторних завдань.

Наведені приклади програм слід розглядати лише як один із можливих варіантів вирішення задачі.

Методичні рекомендації містять опис 8 лабораторних робіт. Кожен розділ, який відповідає окремій лабораторній роботі, складається з таких підрозділів:

мета роботи й вимоги до теоретичної та практичної підготовки, що необхідна для виконання лабораторної роботи;

рекомендації щодо підготовки до виконання лабораторної роботи, основні теоретичні відомості, необхідні для її виконання;

суть роботи – загальна постановка завдання до лабораторної роботи (необов'язково);

індивідуальні варіанти завдань;

контрольні запитання.

При проведенні всіх лабораторних робіт використовується єдина конфігурація програмно-апаратних засобів: персональна ЕОМ типу IBM-PC з процесором не нижче Pentium III, операційна система Windows XP або Windows Vista, середовище візуальної розробки програм Microsoft Visual Studio .NET.

Під час проведення лабораторних робіт студент повинен продемонструвати:

творчий, індивідуальний підхід до розробки проектів (програмної коди);

грамотне використання існуючого програмного забезпечення;

навики програмування на мовах високого рівня C/C++.

Студент повинен уміти перетворити свою програму в програмний продукт, використовувати якісний аналіз програми, виконувати оцінку отриманих результатів. Велике значення має зручний інтерфейс з користувачем і поясненнями до програми.

Варіант завдання до лабораторної роботи вибирається відповідно до номера студента в журналі групи.

***Типовий порядок виконання роботи й методичні рекомендації до її виконання*:**

уважно ознайомитися з методичними рекомендаціями до конкретної лабораторної роботи (теоретичними відомостями, прикладами, формулюванням завдань);

створити заготівлю консольного застосування, скористатися для цього майстром створення додатків Microsoft Visual Studio;

заповнити отриману заготівлю консольного застосування конкретним змістом відповідно до запропонованого завдання (див. приклади);

усунути всі помилки, що виникли на етапі компіляції початкового тексту програми;

виконання програми здійснити в покроковому режимі;

вивести у вікно попереднього перегляду значення всіх проміжних змінних;

знайти свою папку проекту і ознайомитися з її вмістом (за допомогою Блокнота відкрити файл ReadMe.txt і перекласти текст, який у ньому записаний);

підсумковий запуск додатка виконати за допомогою виконуваного модуля;

відповісти на контрольні запитання;

за допомогою динамічної довідки з'ясувати призначення основних службових слів у програмі;

виконати експериментальну частину роботи згідно з отриманим завданням;

оформити звіт і здати викладачеві.

Звіт з будь-якої лабораторної роботи повинен містити:

1. Титульний лист:
   - назва дисципліни;
   - тема лабораторної роботи;
   - дата виконання роботи;
   - П.І.Б. студента, курс, номер групи;
   - П.І.Б., посада викладача.
2. Лист змісту (нумерований перелік назв пунктів роботи із зазначенням номерів сторінок).

3. Опис виконаних завдань:

Умова завдання (завдання).

Опис архітектури програми – специфікація програмних вимог (склад, структура модулів, зв'язки між ними, алгоритми):

формулювання завдання;

специфікація даних;

математична модель обробки даних;

програмний інтерфейс;

план тестування.

Початковий код програми з коментарями для бібліотек (призначення кожної бібліотеки), що підключаються, оголошень, інструкцій, що управляють, і функцій (призначення кожної функції та інструкції, опис параметрів і значення, що повертається).

Приклади результатів роботи програми на тестових початкових даних.

4. Висновки за роботою з урахуванням усіх виконаних завдань:

аналіз отриманих результатів за кожним пунктом завдання;

аналіз результатів тестування програм;

ступінь відповідності розроблених програм постановці завдання;

інша інформація.

Викладач може вносити корективи до оформлення звіту.

Вимоги до оформлення звіту

Поля сторінки: ліве – 2,5 см, праве – 1,5 см, верхнє й нижнє – 2 см; шрифт Times New Roman (висота – 14 пт), міжрядковий інтервал – множник 1,1.

Номери сторінок повинні знаходитися у правому верхньому куті, титульний лист не нумерується.

Листи звіту мають бути з´єднані скріпкою або іншим загальноприйнятим способом.

Лабораторна робота №1
Підготовка і рішення на ПК завдань обробки масивів з використанням покажчиків

Мета лабораторної роботи – освоїти основні прийоми використання покажчиків у програмах на мові С++.

Перед виконанням лабораторної роботи студент повинен знати:

поняття покажчика і основні операції з покажчиками;

взаємозв'язок між покажчиками і массивами;

основи динамічного розподілу пам'яті.

Після виконання лабораторної роботи студент повинен уміти використовувати покажчики при розробці програм на мові С++.

Теоретичний матеріал

Покажчики

Коли компілятор обробляє оператора визначення змінної, наприклад, int і =10, він виділяє пам'ять відповідно до типу (int) і ініціалізує її вказаним значенням (10). Усі звернення в програмі до змінної за її іменем (і) замінюються компілятором на адресу області пам'яті, в якій зберігається значення змінної. Програміст може визначити власні змінні для зберігання адрес областей пам'яті. Такі змінні називаються покажчиками. Отже, *покажчики призначені для зберігання адрес областей пам'яті.* У С++ розрізняють три види покажчиків – покажчики на об'єкт, на функцію і на void, що відрізняються властивостями і набором допустимих операцій. Покажчик не є самостійним типом, він завжди пов'язаний з яким-небудь іншим конкретним типом.

Покажчик на функцію містить адресу в сегменті коду, за яким розташовується виконуваний код функції, тобто адреса, за якою передається управління при виклику функції. Покажчики на функції використовуються для непрямого виклику функції (не через її ім'я, а через звернення до змінної, що зберігає її адресу), а також для передачі імені функції в іншу функцію як параметр. Покажчик функції має тип покажчик функції, що повертає значення заданого типу і має аргументи заданого типу:

тип (*ім'я) (список_типів_аргументів);

Наприклад, оголошення:

int (*fun) (double, double);

задає покажчик з ім'ям fun на функцію, що повертає значення типу int і що має два аргументи типу double.

Покажчик на об'єкт містить адресу області пам'яті, в якій зберігаються дані певного типу (основного або складеного). Просте оголошення покажчика на об'єкт (надалі званого просто покажчиком) має вигляд:

тип *ім’я;

де тип може бути будь-яким, окрім посилання і бітового поля, причому тип може бути до цього моменту тільки оголошений, але ще не визначений (отже, у структурі, наприклад, може бути присутнім покажчик на структуру того ж типу).

Зірочка відноситься безпосередньо до імені, тому для того щоб оголосити декілька покажчиків, потрібно ставити її перед ім'ям кожного з них. Наприклад, в операторі

int *a, b *c;

описуються два покажчики на ціле з іменами a, c, а також ціла змінна b.

Розмір покажчика залежить від моделі пам'яті. Можна визначити покажчик на покажчик.

Покажчик *на void* застосовується в тих випадках, коли конкретний тип об'єкта, адресу якого потрібно зберігати, не визначений (наприклад, якщо в одній і тій же змінній у різні моменти часу потрібно зберігати адреси об'єктів різних типів).

Покажчику на void можна привласнити значення покажчика будь-якого типу, а також порівнювати його з будь-якими покажчиками, але перед виконанням яких-небудь дій з областю пам'яті, на яку він посилається, потрібно перетворити його до конкретного типу явним чином.

Покажчик може бути константою або змінною, а також указувати на константу або змінну. Розглянемо приклади:

```
int i:                          // ціла змінна
const int ci = 1;               // ціла константа
int * pi:                       // покажчик на цілу змінну
const int * pci;                // покажчик на цілу константу
int * const cp = &i;            // покажчик-константа на цілу змінну
const int * const cpc =&ci;     // покажчик-константа на цілу константу
```

Як видно з прикладів, модифікатор const, що знаходиться між ім'ям покажчика і зірочкою, відноситься до самого покажчика і забороняє його

зміну, а const зліва від зірочки задає постійність значення, на яке він указує. Для ініціалізації покажчиків використана операція отримання адреси &. Величини типу покажчик підкоряються загальним правилам визначення області дії, видимості і часу життя.

Ініціалізація покажчиків

Покажчики найчастіше використовують при роботі з динамічною пам'яттю, яка називається деякими естетами купою (переклад з англійської мови слова heap). Це вільна пам'ять, в якій можна під час виконання програми виділяти місце відповідно до потреб. Доступ до виділених ділянок динамічної пам'яті, званих динамічними змінними, проводиться тільки через покажчики. Час життя динамічних змінних – від точки створення до кінця програми або до явного звільнення пам'яті. У C++ використовується два способи роботи з динамічною пам'яттю. Перший використовує сімейство функцій malloc і дістався у спадок від C, другий використовує операції new і delete.

При визначенні покажчика треба прагнути виконати його ініціалізацію, тобто привласнення початкового значення. Ненавмисне використання неініціалізованих покажчиків – поширене джерело помилок у програмах. Ініціалізація записується після імені покажчика або в круглих дужках, або після знака рівності.

Існують наступні способи ініціалізації покажчика:

1. Привласнення покажчику адреси існуючого об'єкта:

за допомогою операції отримання адреси:

```
int a = 5;              // ціла змінна
int* p = &a;            // у покажчик записується адреса a
int* p(&a);             // те ж саме іншим способом;
```

за допомогою значення іншого покажчика, що ініціалізував:

```
int* r = p;
```

за допомогою імені масиву або функції, які трактуються як адреса:

```
int b[10];                  // масив
int* t = b;                 // привласнення адреси початку масиву
void f(int a )  { /* ... */ }   // визначення функції
void (*pf)(int);            // покажчик на функцію
pf = f;                     // привласнення адреси функції.
```

2. Привласнення покажчику адреси області пам'яті в явному вигляді:

```
char* vp = (char *)0xB8000000;
```

Тут 0xB8000000 – шістнадцятирічна константа, (char *) – операція приведення типу: константа перетвориться до типу "покажчик на char".

3. Привласнення порожнього значення:

```
int* suxx = NULL;
int* rulez = 0;
```

У першому рядку використовується константа NULL, визначена в деяких заголовних файлах C як покажчик, рівний нулю. Рекомендується використовувати просто 0, оскільки це значення типу int буде правильно перетворено стандартними способами відповідно до контексту. Оскільки гарантується, що об'єктів з нульовою адресою немає, порожній покажчик можна використовувати для перевірки, посилається покажчик на конкретний об'єкт чи ні.

4. Виділення ділянки динамічної пам'яті і привласнення її адреси покажчику:

за допомогою операції new:

```
int* n = new int;                    // 1
int* m = new int (10);               // 2
int* q = new int [10];               // 3
```

за допомогою функції malloc:

```
int* u = (int *)malloc(sizeof(int)); // 4
```

В операторі 1 операцію new виконує виділення достатнього для розміщення величини типу int ділянки динамічної пам'яті і записує адресу початку цієї ділянки в змінну n. Пам'ять під саму змінну n (розміру, достатнього для розміщення покажчика) виділяється на етапі компіляції.

В операторі 2, окрім описаних вище дій, проводиться ініціалізація виділеної динамічної пам'яті значенням 10.

В операторі 3 операція new виконує виділення пам'яті під 10 величин типу int (масиву з 10 елементів) і записує адресу початку цієї ділянки в змінну q, яка може трактуватися як ім'я масиву. Через ім'я можна звертатися до будь-якого елемента масиву. Якщо пам'ять виділити не вдалося, за стандартом повинне породжуватися виключення bad_alloc. Старі версії компіляторів можуть повертати 0.

В операторі 4 робиться те ж саме, що і в операторі 1, але за допомогою функції виділення пам'яті malloc, успадкованої з бібліотеки З. У функцію передається один параметр – кількість пам'яті, що виділяється, у байтах. Конструкція (int*) використовується для приведення типу покажчика, що повертається функцією, до необхідного типу. Якщо пам'ять виділити не вдалося, функція повертає 0.

Операцію new використовувати більше, чим функцію malloc, особливо при роботі з об'єктами.

Звільнення пам'яті, виділеної за допомогою операції new, повинно виконуватися за допомогою delete, а пам'яті, виділеною функцією malloc, – за допомогою функції free. При цьому змінна-покажчик зберігається і може ініціалізуватися повторно. Наведені вище динамічні змінні знищуються таким чином:

delete n; delete m; delete [] q; free (u);

Якщо пам'ять виділялася з допомогою new[], для звільнення пам'яті необхідно застосовувати delete[]. Розмірність масиву при цьому не указується. Якщо квадратних дужок немає, то ніякого повідомлення про помилку не видається, але помічений як вільний буде тільки перший елемент масиву, а інші стануть недоступними для подальших операцій. Такі елементи пам'яті називаються сміттям.

За допомогою комбінацій зірочок, круглих і квадратних дужок можна описувати складені типи і покажчики на складені типи, наприклад, в операторі

int *(*p[10])();

оголошується масив з 10 покажчиків на функції без параметрів, що повертають покажчики на int.

За замовчуванням квадратні і круглі дужки мають однаковий пріоритет, більший, ніж зірочка, і розглядаються зліва направо. Для зміни порядку розгляду використовуються круглі дужки.

При інтерпретації складних описів необхідно дотримуватися правила із середини назовні:

1) якщо праворуч від імені є квадратні дужки, то це масив, якщо дужки круглі – функція;

2) якщо зліва є зірочка, це покажчик на проінтерпретовану раніше конструкцію;

3) якщо справа зустрічається закриваюча кругла дужка, необхідно застосувати наведено вище правило усередині дужок, а потім переходити назовні;

4) в останню чергу інтерпретується специфікатор типу.

Операції з покажчиками

З покажчиками можна виконувати наступні операції: розадресація, або непряме звернення до об'єкта (*), привласнення, складання з константою, віднімання, інкремент (++), декремент (--), порівняння, приведення

типів. При роботі з покажчиками часто використовується операція отримання адреси (&).

Операція розадресації, або розіменування, призначена для доступу до величини, адреса якої зберігається в покажчику. Цю операцію можна використовувати як для отримання, так і для зміни значення величини (якщо вона не оголошена як константа):

```
    char a;                    // змінна типу char
char * p = new char;           /* виділення пам'яті під покажчик і під
                                  динамічну змінну типу char */
    *p = 'A'; a = *p;          // привласнення значення обом змінним
```

Як видно з прикладу, конструкцію *ім’я_покажчика можна використовувати в лівій частині оператора привласнення, оскільки вона є L-значенням, тобто визначає адресу області пам'яті. Для простоти цю конструкцію можна вважати ім'ям змінної, на яку посилається покажчик. З нею допустимі всі дії, визначені для величин відповідного типу (якщо покажчик ініціалізував). На одну і ту ж область пам'яті може посилатися декілька покажчиків різного типу. Застосована ним операція розадресації дасть різні результати. Наприклад, програма

```
#include <stdio.h>
 int main()
 {
 unsigned long int A = 0Xcc77ffaa;
 unsigned short int* pint = (unsigned short int*) &A;
 unsigned char* pchar = (unsigned char *) &A;
 printf("  | %x | Xx | Xx |", A *pint, *pchar);
 return 0;
   }
```

виведе на екран рядок:

```
 | cc77ffaa | ffaa | aa |
```

Значення покажчиків pint і pchar однакові, але розадресації **pchar** дає в результаті один молодший байт за цією адресою, а pint – два молодші байти. У наведеному вище прикладі при ініціалізації покажчиків були використані операції приведення типів. Синтаксис операції явного приведення типу простий: перед ім'ям змінної в дужках указується тип, до якого її потрібно перетворити. При цьому не гарантується збереження інформації, тому в загальному випадку явних перетворень типу слід уникати.

При змішуванні у виразі покажчиків різних типів явне перетворення типів потрібне для всіх покажчиків, окрім void*. Покажчик може неявно перетворюватися в значення типу bool (наприклад, у виразі умовного оператора), при цьому ненульовий покажчик перетвориться в true, а нульовий – false. Привласнення без явного приведення типів допускається в двох випадках:

покажчикам типу void*;

якщо тип покажчиків справа і зліва від операції привласнення один і той же.

Таким чином, неявне перетворення виконується тільки до типу void*. Значення 0 неявно перетвориться до покажчика на будь-який тип. Привласнення покажчиків на об'єкти покажчикам на функції (і навпаки) неприпустимо. Заборонено привласнювати значення покажчикам-константам, утім, як і константам будь-якого типу (привласнювати значення покажчикам на константу і змінним, на які посилається покажчик-константа, дозволяється).

Арифметичні операції з покажчиками (складання з константою, віднімання, інкремент і декремент) автоматично враховують розмір типу величин, що адресуються покажчиками. Ці операції застосовні тільки до покажчиків одного типу і мають сенс в основному при роботі із структурами даних, послідовно розміщеними в пам'яті, наприклад, з масивами.

Інкремент переміщає покажчик до наступного елемента масиву, *декремент* – до попереднього. Фактично значення покажчика змінюється на величину sizeof(тип). Якщо покажчик на певний тип збільшується або зменшується на константу, його значення змінюється на величину цієї константи, помножену на розмір об'єкта даного типу, наприклад:

```
short * p = new short [5]:
p++;                    // значення p збільшується на 2
long * q = new long [5];
q++;                    // значення q збільшується на 4
```

Різниця двох покажчиків - це різниця їх значень, що ділиться на розмір типу в байтах (у застосуванні до масивів різниця покажчиків, наприклад, на третій і шостий елементи рівна 3). Підсумовування двох покажчиків не допускається.

При записі виразів з покажчиками слід звертати увагу на пріоритети операцій. Як приклад розглянемо послідовність дій, задану в операторі

```
*p++ = 10;
```

Операції розадресації і інкремента мають однаковий пріоритет і виконуються справа наліво, але, оскільки інкремент постфіксний, він виконується після виконання операції привласнення. Таким чином, спочатку за адресою, записаною в покажчику p, буде записане значення 10, а потім покажчик буде збільшений на кількість байтів відповідно до його типу. Те ж саме можна записати докладніше:

```
*p = 10; p++;
```

Вираз (*p)++, навпаки, інкрементує значення, на яке посилається покажчик.

Унарна операція отримання адреси & застосовна до величин, що мають ім'я і розміщені в оперативній пам'яті. Таким чином, не можна одержати адресу скалярного виразу, неіменованої константи або регістрової змінної. Приклади операції наводилися вище.

Ідентифікатор масиву є константним покажчиком на його нульовий елемент. Наприклад, для масиву з попереднього лістингу ім'я b – це те ж саме, що &b[0], а до і-го елемента масиву можна звернутися, використовуючи вираз *(b+i). Можна описати покажчик, привласнити йому адресу початку масиву і працювати з масивом через покажчик. Наступний фрагмент програми копіює всі елементи масиву a в масив b:

```
Int a[100], b[100];
int *pa = a;
int *pb = b:
for (int i = 0; i<100; i++)
*pb++ = *pa++;   // або pb[i]= pa[i];
```

Динамічні масиви створюють за допомогою операції new, при цьому необхідно вказати тип і розмірність, наприклад:

```
int n = 100;
float *p = new float [n];
```

У цьому рядку створюється змінна-покажчик на float, у динамічній пам'яті відводиться безперервна область, достатня для розміщення 100 елементів речовинного типу, і адреса її початку записується в покажчик p. Динамічні масиви не можна при створенні ініціалізувати і вони не обнуляються.

Перевага динамічних масивів полягає в тому, що розмірність може бути змінною, тобто об'єм пам'яті, що виділяється під масив, визначається на етапі виконання програми. Доступ до елементів динамічного масиву здійснюється точно так, як і до статичних, наприклад, до елемента номер 5 наведеного вище масиву можна звернутися як p[5] або *(p+5).

Альтернативний спосіб створення динамічного масиву – використання функції malloc бібліотеки C:

```
int n = 100:
 float *q = (float *) malloc(n * sizeof(float));
```

Операція перетворення типу, записана перед зверненням до функції malloc, потрібна тому, що функція повертає значення покажчика типу void*, а ініціалізувався покажчик на float.

Пам'ять, зарезервована під динамічний масив за допомогою new [], повинна звільнятися операцією delete [], а пам'ять, виділена функцією malloc, – за допомогою функції free, наприклад:

```
delete [] p; free (q);
```

При невідповідності способів виділення і звільнення пам'яті результат не визначений. Розмірність масиву в операції delete не указується, але квадратні дужки обов'язкові.

Багатовимірні масиви задаються вказівкою кожного вимірювання в квадратних дужках, наприклад, оператор

```
int matr [5][6];
```

задає опис двовимірного масиву з 5 рядків і 6 стовпців. У пам'яті такий масив розташовується послідовно. Багатовимірні масиви розміщуються так, що при переході до наступного елемента найшвидше змінюється останній індекс. Для *доступу* до елемента багатовимірного масиву указуються всі його індекси, наприклад, matr[i][j], або більш екзотичним способом: *(matr[i]+j) або *(*(matr+i)+j). Це можливо, оскільки matr[i] є адресою початку i-го рядка масиву.

Для створення динамічного багатовимірного масиву необхідно вказати в операції new усі його розмірності (найлівіша розмірність може бути змінною) наприклад:

```
int nstr = 5;
int ** m = (int **) new int [nstr][10];
```

Більш універсальний і безпечний спосіб виділення пам'яті під двовимірний масив, коли обидві його розмірності задаються на етапі виконання програми наведений нижче:

```
int nstr, nstb;
 cout << " Введіть кількість рядків і стовпців :";
cin >> nstr , nstb;
int **a = new int *[nstr];         // 1
```

```
for(int i = 0;  i<nstr;  i++)        // 2
a[i] = new int [nstb];               // 3
```

В операторі 1 оголошується змінна типу покажчик на покажчик на int і виділяється пам'ять під масив покажчиків на рядки масиву (кількість рядків nstr). В операторі 2 організовується цикл для виділення пам'яті під кожну строку масиву. В операторі 3 кожному елементу масиву покажчиків на рядки привласнюється адреса початку ділянки пам'яті, виділеного під рядок двовимірного масиву. Кожен рядок складається з nstb елементів типу int.

Звільнення пам'яті з-під масиву з будь-якою кількістю вимірювань виконується за допомогою операції delete []. Покажчик на константу видалити не можна.

Завдання до лабораторної роботи

Скласти програму, що виконує з одновимірним масивом дії, відповідно до варіанта завдання. Замість класичного доступу до елементів масиву (наприклад, **MyArray[i]**) і виконання операцій над елементами використовувати покажчики. Обработку масиву виконати у функції. Функцію в головній програмі викликати через покажчик. Пам'ять під масив виділити динамічно.

Скласти програму, що виконує з одновимірним масивом наступні дії.

Варіант 1

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

суму негативних елементів масиву;

множення елементів масиву, розташованих між максимальним і мінімальним елементами.

Упорядкувати елементи масиву за збільшенням.

Варіант 2

В одновимірному масиві, що складається з n речових елементів, обчислити:

суму позитивних елементів масиву;

множення елементів масиву, розташованих між максимальним за модулем і мінімальним за модулем елементами.

Упорядкувати елементи масиву за убуванням.

Варіант 3

В одновимірному масиві, що складається з цілих елементів, обчислити:

множення елементів масиву з парними номерами;

суму елементів масиву, розташованих між першим і останнім нульовими елементами.

Перетворити масив так, щоб спочатку розташовувалися всі позитивні елементи, а потім – всі негативні (елементи, які дорівнюють 0, вважати позитивними).

Варіант 4

В одновимірному масиві, що складається з n речових елементів, обчислити:

суму елементів масиву з непарними номерами;

суму елементів масиву, розташованих між першим і останнім негативними елементами.

Стискувати масив, видаливши з нього всі елементи, модуль яких не перевищує 1. Елементи, що звільнилися в кінці масиву, заповнити нулями.

Варіант 5

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

максимальний елемент масиву;

суму елементів масиву, розташованих до останнього позитивного елемента.

Стискувати масив, видаливши з нього всі елементи, модуль яких знаходиться в інтервалі [a, b]. Елементи, що звільнилися в кінці масиву, заповнити нулями.

Варіант 6

В одновимірному масиві, що складається з n речових елементів, обчислити:

мінімальний елемент масиву;

суму елементів масиву, розташованих між першим і останнім позитивними елементами.

Перетворити масив так, щоб спочатку розташовувалися всі елементи, які дорівнюють нулю, а потім – усі останні.

Варіант 7

В одновимірному масиві, що складається з n цілих елементів, обчислити:

номер максимального елемента масиву;

множення елементів масиву, розташованих між першим і другим нульовими елементами.

Перетворити масив так, щоб у першій його половині розташовувалися елементи, що стояли в непарних позиціях, а в другій половині – елементи, що стояли в парних позиціях.

Варіант 8

В одновимірному масиві, що складається з n речових елементів, обчислити:

номер мінімального елемента масиву;

суму елементів масиву, розташованих між першим і другим негативними елементами.

Перетворити масив так, щоб спочатку розташовувалися всі елементи, модуль яких не перевищує 1, а потім – усі останні.

Варіант 9

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

максимальний за модулем елемент масиву;

суму елементів масиву, розташованих між першим і другим позитивними елементами.

Перетворити масив так, щоб елементи, які дорівнюють нулю, розташовувалися після всіх останніх.

Варіант 10

В одновимірному масиві, що складається з n цілих елементів, обчислити:

мінімальний за модулем елемент масиву;

суму модулів елементів масиву, розташованих після першого елемента, який дорівнює нулю.

Перетворити масив так, щоб у першій його половині розташовувалися елементи, що стояли в парних позиціях, а в другій половині — елементи, що стояли в непарних позиціях.

Варіант 11

В одновимірному масиві, що складається з n речових елементів, обчислити:

номер мінімального за модулем елемента масиву;

суму модулів елементів масиву, розташованих після першого негативного елемента.

Стискувати масив, видаливши з нього всі елементи, величина яких знаходиться в інтервалі [a, b]. Елементи, що звільнилися в кінці масиву, заповнити нулями.

Варіант 12

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

номер максимального за модулем елемента масиву;

суму елементів масиву, розташованих після першого позитивного елемента.

Перетворити масив так, щоб спочатку розташовувалися усі елементи, ціла частина яких лежить в інтервалі [a, b], а потім – всі останні.

Варіант 13

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

кількість елементів масиву, розташованих у діапазоні від A до B;

суму елементів масиву, розташованих після максимального елемента.

Упорядкувати елементи масиву за убуванням модулів елементів.

Варіант 14

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

кількість елементів масиву, які дорівнюють 0;

суму елементів масиву, розташованих після мінімального елемента.

Упорядкувати елементи масиву за збільшенням модулів елементів.

Варіант 15

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

кількість елементів масиву, які більше C;

множення елементів масиву, розташованих після максимального за модулем елемента.

Перетворити масив так, щоб спочатку розташовувалися всі негативні елементи, а потім – усі позитивні (елементи, які дорівнюють 0, вважати позитивними).

Варіант 16

В одновимірному масиві, що складається з n дійсних елементів, обчислити:

кількість негативних елементів масиву;

суму модулів елементів масиву, розташованих після мінімального за модулем елемента.

Замінити всі негативні елементи масиву їх квадратами і упорядкувати елементи масиву за збільшенням.

Варіант 17

В одновимірному масиві, що складається з n цілих елементів, обчислити:

кількість позитивних елементів масиву;

суму елементів масиву, розташованих після останнього елемента, який дорівнює нулю.

Перетворити масив так, щоб спочатку розташовувалися всі елементи, ціла частина яких не перевищує 1, а потім – усі останні.

Варіант 18

В одновимірному масиві, що складається з n речових елементів, обчислити:

кількість елементів масиву, які менше C;

суму цілих частин елементів масиву, розташованих після останнього негативного елемента.

Перетворити масив так, щоб спочатку розташовувалися всі елементи, що відрізняються від максимального не більше ніж на 20%, а потім – усі останні.

Варіант 19

В одновимірному масиві, що складається з n речових елементів, обчислити:

множення негативних елементів масиву;

суму позитивних елементів масиву, розташованих до максимального елемента.

Змінити порядок проходження елементів у масиві на зворотний.

Варіант 20

В одновимірному масиві, що складається з n речових елементів, обчислити:

множення позитивних елементів масиву;

суму елементів масиву, розташованих до мінімального елемента.

Упорядкувати за збільшенням окремо елементи, що стоять на парних місцях, і елементи, що стоять на непарних місцях.

Контрольні запитання

1. Дайте визначення динамічному масиву.
2. Які операції можливі з покажчиками?
3. Укажіть приклади завдання розмірності динамічного масиву.
4. Як можна ініціалізувати елементи массиву за допомогою покажчиків?
5. Яке інформаційне навантаження несе ім'я масиву?
6. Яка операція використовується для визначення адреси довільного елемента масиву?

Лабораторна робота №2
Підготовка і рішення на ПК завдань з використанням рядків і файлів

Мета лабораторної роботи – знайомство з можливостями введення-виведення даних, освоєння методики обробки інформації з використанням рядків, а також препроцесорної обробки даних, освоєння методики обробки інформації з використанням багатофайлових програм.

Перед виконанням лабораторної роботи студент повинен знати:

визначення і правила опису рядків з використанням засобів препроцесорної обробки;

основні функції для роботи з рядками і файлами;

Після виконання лабораторної роботи студент повинен уміти:

виконувати основні операції з рядками: створення, копіювання, конкатенацію, пошук підрядка в рядку і т. д.;

розробляти програми для вирішення завдань з використанням багатофайлових програм.

Теоретичний матеріал

Рядки

Тип char. Значеннями типу char є цілі числа із знаком (signed char) або без знака (unsigned char), які поміщаються в один байт. Від інших цілих типів його відрізняє наявність символічних констант вигляду:

'A' – для символів, що зображаються, '\ooo' і '\xhhh' – для всіх символів без виключення;

де ooo – 8-річні, а hhh – 16-річні цифри.

Декілька символів мають власні імена:

\n – новий рядок;

\t – горизонтальна табуляція;

\v – вертикальна табуляція;

\b – повернення назад;

\r – повернення каретки;

\a – дзвінок (attention);

\ – зворотна коса межа;

\' – одинарна лапка;

\" – подвійна лапка.

Зауваження. Значення типу char, що виводяться у вихідний потік cout, виглядають як символи, а не як числа, тільки завдяки визначенню класу cout.

Рядки символів як масиви

Рядок має тип "масив з символів". Рядок завершується нульовим символом. Наприклад, рядок QWERTY має тип char [7], порожній рядок " має тип char[1].

Рядкова константа – це послідовність символів, поміщена в подвійні лапки. У числі символів рядка можуть знаходитися будь-які символьні константи, наприклад:

Дзвінок в кінці повідомлення \007\n.

Сусідні рядкові константи транслятором "склеюються". Наприклад: АБВ ДЕ означає те ж, що АБВГДЕ.

Рядкові константи можна використовувати для ініціалізації символьних масивів. Наприклад, так можна визначити масив **s** з 7 символів і ініціалізувати його:

char s[] = АБВГДЕ; .

Завдання. Заданий рядок. Скопіювати його в символьний масив. Для контролю вивести у стандартний вихідний потік рядок і масив.

Рішення.

```
#include <iostream>
int main()  {
      char s1[ ]=1234567890, s2[11];
      for (int i = 0; s1[i]; i++) s2[i] = s1[i];
            s2[i] = 0;
      cout << s1 << ' = ' << s2 << '\n';
return 0;
}
```

Програму можна зробити трохи коротше, переписавши оператора циклу:

```
for (int i = 0; s2[i]=s1[i]; i++);
```

Якщо пригадати про покажчики на символи, можна написати і так:

```
for (char *p1 = s1, *p2 = s2; *p2++=*p1++;);
```

Зауваження. Кома в C++ є не тільки роздільником, але і оператором послідовного виконання. Значенням цієї операції є значення найправішого операнда.

Рядкові бібліотечні функції

Функції для роботи з рядками описані в заголовному файлі string.h. Деякі з них:

```
char *stpcpy(char *dest, const char *src);
```

Копіює символи рядка, поки не скопіює нульовий символ. Повертає величину dest + strlen(src). Пам'ять для dest повинна бути наперед зарезервована.

```
char *strcat(char *dest, const char *src);
```

Приєднує другий рядок до першого. Повертає покажчик на початок нарощеного рядка.

```
char *strchr(const char *s, int з);
```

Сканує рядок s у пошуці першого входження заданого символу з. Нульовий символ можна шукати разом з іншими. Повертає покажчик на знайдений символ або 0, якщо символу немає.

```
int strcmp(const char *s1, const char*s2);
```

Порівнює 2 рядки. Повертає ціле менше нуля, якщо s1 < s2, рівне нулю – якщо s1 == s2, і більше нуля – якщо s1 > s2.

```
char *strcpy(char *dest, const char *src);
```

Копіює другий аргумент у перший. Повертає покажчик на копію. Пам'ять для dest повинна бути наперед зарезервована.

```
char *strdup(const char *s);
```

Копіює рядок у новостворюваний функцією malloc() область пам'яті. Повертає покажчик на створену копію або 0 при невдачі. Програміст відповідає за звільнення пам'яті.

Приклад.

```
char *dup_str = strdup (string);
...
free (dup_str);
size_t strlen(const char *s);
```

Підраховує розмір рядка. Повертає кількість символів рядка без нульового символу. Тип size_t визначений у файлі string.h і інших заголовних файлах як ціле без знака:

typedef unsigned size_t;

char *strpbrk (const char *s1, const char *s2);

Сканує перший рядок у пошуці першого входження будь-якого символу з другого рядка. Повертає покажчик на знайдений символ або 0 при невдачі.

char *strrchr (const char *s, int a);

Те ж, що strchr, але знаходить останнє входження символу a в рядок s.

char *strset(char *s, int ch);

Пише символ ch замість всіх символів рядка. Повертає s.

char *strstr(const char *s1, const char *s2);

Знаходить перше входження підрядка s1 у рядок s1. Повертає покажчик на місце першого входження або 0, якщо такого немає.

char *strtok(char *s1, const char *s2);

Сканує перший рядок у пошуці першої ділянки, що не містить символів з s2. Перший виклик функції повертає покажчик на початок першої ділянки і записує 0 в s1 відразу після кінця ділянки. Подальші виклики з NULL як 1-й аргумент обробляють рядок далі, поки що є такі ділянки. Якщо їх немає, повертається 0. Рядок s2 може змінюватися від виклику до виклику.

Функцію застосовують для виділення слів з пропозиції s1. У рядку s2 знаходяться символи-роздільники.

Зауваження. Для кожної функції існує варіант з дальніми покажчиками. Його ім'я, як правило, має префікс **"_f".**

Різновиди введення і виведення

Засоби введення і виведення формально не входять у стандарт мов C і C++, але фактично стандартізовані й містяться в системних бібліотеках функцій.

У них можна виділити наступні групи:

1) консольні – орієнтовані на введення з клавіатури і виведення на дисплей. Описані в заголовному файлі conio.h;

2) файлові – призначені для роботи з файлами. Описані в io.h;

3) потокові – аналогічні файловим, але надають більший сервіс програмісту. Описані в stdio.h;

4) засоби ДОС – введення і виведення функціями операційної системи. Описані в dos.h;

5) об'єктні – об'єктно-орієнтоване введення/виведення, тільки в C++. Описані в iostream.h, fstream.h, iomanip.h.

Відкриття і закриття потоку

Схема роботи з потоком така ж, як і з файлом: відкрити потік, виконати читання і/або запис, закрити потік.

Відкриває потік функція

FILE* fopen(
const char *filename,// ім'я файла, що асоціюється з потоком
const char *mode // рядок режимів роботи з потоком
) – повертає покажчик на потік, який ідентифікує його в подальших операціях.

У рядку режимів можуть знаходитися наступні символи:

r – відкрити тільки для читання;

w – створити для запису. Існуючий файл буде перекритий новим;

a – відкрити для дозапису, або створити для запису, якщо файл не існує;

+ – операції виконуватимуться з уже існуючим файлом;

t – текстовий режим (обробка символів CR-LF);

b – двійковий режим (ніякої обробки).

За відсутності в рядку b або t режим визначається глобальній змінній _fmode, визначеній у заголовному файлі fcntl.h.

FILE – це структура для потоку, що управляє, оголошена в stdio.h. Вона не призначена для прямого використання.

Потік є програмною надбудовою над файлом, що надає програмісту додатковий сервіс. Крім сумісного відкриття потоку і файла (fopen), можна відкрити потік і асоціювати його з уже відкритим файлом (fdopen), відкрити файл і асоціювати його з уже відкритим потоком (freopen).

Закриває потік і вивантажує буфери функція

int *fclose(FILE *stream)* – повертає 0 при успіху і EOF – при помилці.

EOF – константа, визначена в stdio.h.

int *fcloseall(void)* – закриває всі відкриті потоки, окрім стандартних: stdin, stdout, stdprn, stderr і stdaux.

Введення і виведення символів

Читання символу з потоку виконується функцією

int *fgetc(*FILE **stream)* – повертає код символу. При помилці повертає EOF.

Запис символу в потік виконується функцією

int *fputc(*int *з,* FILE **stream*) – повертає код символу з. При помилці повертає EOF.

Читання символу зі стандартного потоку stdin виконується функцією int *fgetchar(void);.*

Запис символу в стандартний потік stdout виконується функцією int *fputchar(int з);.*

Завдання. Скопіювати файл xxx.bin у файл yyy.bin.

Рішення.

```
#include <stdio.h>
void main()  {       FILE *in = fopen ("noname00.cpp","rt");
       FILE *out = fopen ("noname00.000","wt");
      if (!in) return;
    while (!feof(in))
           fputc(fgetc(in), out);
fcloseall();    }
```

Введення і виведення рядків

Читання рядка з потоку виконується функцією

char *fgets(

char *s, // покажчик на буфер, що приймає рядок

int n, // гранична кількість читаних символів (звичайно розмір буфера)

FILE *stream)

Вона повертає покажчик на буфер або NULL при помилці.

Читання припиняється, коли досягнутий кінець рядка або прочитано n-1 символів з файла. Рядок у буфері замикається нульовим символом. Покажчик файла переміщається за символи CR-LF.

Запис рядка в потік виконується функцією

char **fputs(*

char **s, // покажчик на рядок*

FILE **stream)*

Вона повертає покажчик на останній записаний символ або EOF при помилці.

Термінальний символ рядка не копіюється. Символи CR-LF додаються у файл.

Завдання. Скопіювати текстовий файл xxx.txt у файл yyy.txt за рядками.

Рішення.

```
#include <stdio.h>
void main()   {    const int n = 100;
   char buf [n];
   FILE *in = fopen (noname00. cpp,r);
   FILE *out = fopen (noname00. 000,w);
   if (!in) return;
   while (!feof(in)) {            fgets(buf, n, in);
          fputs(buf, out);
  }
   fcloseall();   }
```

Введення і виведення записів

Читання записів з потоку виконується функцією

```
size_t fread(
void *ptr,     // покажчик на буфер у пам'яті, що приймає записи
size_t size, // розмір запису в байтах
size_t n,     // кількість читаних записів
FILE *stream )
```

При успіху функція повертає n, при невдачі – кількість прочитаних записів, можливо нульова.

Загальна кількість читаних байтів дорівнює n *size.

Виведення записів у потік виконується функцією

```
size_t fwrite(
const void *ptr,
size_t  size
size_t  n
FILE* stream )
```

Вона додає вказану кількість записів у файл.

Сенс параметрів і значення, що повертається, той же, що у функції fread.

Управління покажчиком файла

Читання і запис виконуються в тому місці файла, де знаходиться покажчик файла. Встановити покажчик можна функцією

```
int fseek(FILE *stream,
long offset,                  // зсув покажчика
int whence  )                  // відлік зсуву
```

Вона повертає 0 при успіху.

При помилці, викликаною неможливістю відкрити файл або пристрій, повертає ненульове значення. Інші помилки не діагностуються.

Для вказівки точки відліку зсуву використовують константи:

SEEK_SET = 0 – відлік від початку файла;

SEEK_CUR = 1 – відлік від кінця файла;

SEEK_END = 2 – відлік від поточної позиції покажчика.

Функція

long ftell(FILE *stream) – повертає поточну позицію покажчика.

При помилці повертає 1 і встановлює глобальну змінну errno в ненульове значення.

Зауваження. Ті ж дії виконуються функціями fsetpos і fgetpos.

Стан потоку

Макрос, який перевіряє досягнення кінця файла потоку:

int feof (FILE *stream) – повертає не 0, якщо досягнутий кінець файла, і 0 – навпаки.

Макрос, що тестує індикатор помилки потоку:

int ferror(FILE *stream) – повертає не 0, якщо виявлена помилка запису або читання.

Одного разу встановлений індикатор помилки зберігається до виконання функцій clearerr, rewind або закриття потоку. Індикатор "кінець файла" встановлюється заново кожною операцією читання.

void clearerr(FILE *stream) – обнуляє індикатори помилки і кінця файла.

void rewind(FILE *stream) – робить те ж, що clearerr, а також встановлює покажчик у початок файла.

При виникненні помилки глобальна змінна errno (визначена у файлах errno.h, stddef.h, stdlib.h) одержує ненульовий номер помилки.

Форматоване виведення

Розглянуті вище функції виводять інформацію в потік без або майже без перетворення. Функція fprintf перетворить дані, що виводяться, у послідовність символів, керуючись рядком формату.

int fprintf (FILE *stream,

const char *format // рядок формату

[, argument, ...]) // значення, що виводяться

При успіху повертає кількість виведених байт, при невдачі – EOF. Квадратні дужки говорять про необов'язковість аргументу.

Рядок формату містить прості символи і специфікації формату. Прості символи копіюються у вихідний потік без зміни, специфікації застосовуються для форматування решти аргументів функції. Якщо аргументів менше, ніж специфікацій, наслідки непередбачувані. Якщо аргументів більше, ніж специфікацій, зайві аргументи ігноруються.

Загальний вигляд специфікації формату наступний:

%[прапори] [ширина] [.точність] [розмір] тип;

прапори – ознаки вирівнювання, використання знаків, десяткової крапки, кінцевих нулів, 8-річних і 16-річних префіксів;

ширина – мінімальне число друкованих символів з урахуванням пропусків і нулів;

точність – максимальне число друкованих символів (для цілих – мінімальне число цифр);

розмір – визначає розмір аргументу;

тип –- символ специфікації типу – обов'язковий елемент формату.

Форматоване введення

Для форматованого введення з потоку застосовують функцію

int fscanf (FILE *stream,

const char *format

[, address, ...])

Ця функція повертає число полів введення тих, що відформатували і розміщених у пам'яті. При невдачі повертає EOF.

Функція fscanf розглядає вхідний потік як послідовність полів введення. Поле введення закінчується:

першим символом пропуску (але не включає його);

першим символом, який не може бути перетворений за специфікацією формату, зіставленій цьому полю;

(n+1)-м символом, якщо специфікація включає ширину поля в n символів.

Функція проглядає послідовність полів введення, форматує їх і розміщує за адресами – аргументами fscanf. Число адрес, специфікацій формату і полів введення повинно бути узгоджено.

Рядок формату складається з символів (' ',\t,\n), що невідображаються, символів (всі інші, окрім '%'), що відображаються, і специфікаторів формату.

Якщо fscanf зустрічає символ, що не відображається, в рядку формату, вона прочитуватиме, але не зберігатиме всі символи вхідного потоку, що не відображаються, аж до першого символу, що відображається.

Якщо fscanf зустрічає символ, що відображається, в рядку формату, вона прочитає, але не збереже відповідний символ вхідного потоку.

Специфікація формату вказує fscanf на читання, перетворення і розміщення в пам'яті одного вхідного поля.

Загальний вигляд специфікації формату:

%[*] [ширина] [модіф. розміру] [модіф. типу арг.] символ типу;

* – відміняє привласнення поля введення;

ширина – максимальне число прочитуваних символів;

модифікатор розміру – N – near, F – far;

модифікатор типу аргументу – змінює тип адресного аргументу;

символ типу – символ специфікації типу – обов'язковий елемент формату.

У таблиці 2.1 наведені інші функції введення формату із зазначенням заголовного файла і вхідного потоку.

Таблиця 2.1

Функції введення із зазначенням заголовного файла і вхідного потоку

cprintf	CONIO.H	Консоль
fprintf	STDIO.H	Потік
printf	STDIO.H	stdout
sprintf	STDIO.H	Рядок

У табл. 2.2. подані інші функції виведення формату із зазначенням заголовного файла і вхідного потоку.

Таблиця 2.2

Функції виведення із зазначенням заголовного файла і вхідного потоку

cscanf	CONIO.H	Консоль
fscanf	STDIO.H	Потік
scanf	STDIO.H	stdin
sscanf	STDIO.H	Рядок

Завдання до лабораторної роботи

Виконати вправи лабораторної роботи, оформивши програмний продукт у вигляді багатофайлового проекту.

Усі необхідні дані для функцій повинні передаватися їм як параметри. Використання глобальних змінних у функціях не допускається.

За допомогою текстового редактора створити файл, що містить текст, довжина якого не перевищує 1000 символів (довжина рядка не повинна перевищувати 70 символів). Ім'я файла повинне мати розширення DAT.

Варіант 1

Написати програму, яка:

виводить текст на екран дисплея;

при натисненні довільної клавіші по черзі виділяє кожне речення тексту;

визначає кількість речень у тексті.

Варіант 2

Написати програму, яка:

виводить текст на екран дисплея;

при натисненні довільної клавіші по черзі виділяє кожне слово тексту;

визначає кількість слів у тексті.

Варіант 3

Написати програму, яка:

виводить текст на екран дисплея;

при натисненні довільної клавіші по черзі виділяє кожне слово тексту, що закінчується на голосну букву;

визначає кількість слів у тексті, що закінчуються на голосну букву.

Варіант 4

Написати програму, яка:

виводить текст на екран дисплея;

при натисненні довільної клавіші по черзі виділяє кожне речення тексту в послідовності 2, 3, 1.

Варіант 5

Написати програму, яка:

виводить текст на екран дисплея;

при натисненні довільної клавіші по черзі виділяє кожне із слів тексту, у яких перший і останній символи співпадають;

визначає кількість слів тексту, у яких перший і останній символи співпадають.

Варіант 6

Написати програму, яка:

виводить текст на екран дисплея;

при натисненні довільної клавіші по черзі виділяє кожне слово тексту, що починається на голосну букву;

визначає кількість слів у тексті, що починаються на голосну букву.

Варіант 7

Написати програму, яка:

виводить текст на екран дисплея;

визначає кількість символів у найдовшому слові;

при натисненні довільної клавіші по черзі виділяє кожне слово тексту, що містить максимальну кількість символів.

Варіант 8

Написати програму, яка:

виводить текст на екран дисплея;

визначає кількість символів у найкоротшому слові;

при натисненні довільної клавіші по черзі виділяє кожне слово тексту, що містить мінімальну кількість символів.

Варіант 9

Написати програму, яка:

виводить текст на екран дисплея;

визначає в кожному реченні тексту кількість символів, відмінних від букв і пробілу;

при натисненні довільної клавіші по черзі виділяє кожне речення тексту, а у виділеній пропозиції – по черзі всі символи, відмінні від букв і пробілу.

Варіант 10

Написати програму, яка:

виводить текст на екран дисплея;

визначає кількість речень тексту і кількість слів у кожному реченні;

при натисненні довільної клавіші по черзі виділяє кожне речення тексту, а у виділеному реченні – по черзі всі слова.

Варіант 11

Написати програму, яка:

виводить текст на екран дисплея;

визначає кількість букв *а* в останньому слові тексту;

при натисненні довільної клавіші по черзі виділяє останнє слово, а у виділеному слові – по черзі всі букви *а*.

Варіант 12

Написати програму, яка:

виводить текст на екран дисплея;

визначає найдовшу послідовність цифр у тексті (вважати, що будь-яка кількість пробілів між двома цифрами не перериває послідовності цифр);

при натисненні довільної клавіші по черзі виділяє кожну послідовність цифр, що містить максимальну кількість символів.

Варіант 13

Написати програму, яка:

виводить текст на екран дисплея;

визначає порядковий номер заданого слова у кожному реченні тексту;

при натисненні довільної клавіші по черзі виділяє кожне речення тексту, а у виділеному реченні – задане слово.

Варіант 14

Написати програму, яка:

виводить текст на екран дисплея;

при натисненні довільної клавіші по черзі виділяє у тексті задане слово (задане слово вводиться з клавіатури);

виводить текст на екран дисплея ще раз, викидаючи з нього задане слово і видаляючи зайві пробіли.

Варіант 15

Написати програму, яка:

виводить текст на екран дисплея;

при натисненні довільної клавіші по черзі виділяє в тексті задані слова, які потрібно поміняти місцями (задані слова вводяться з клавіатури);

виводить текст на екран дисплея ще раз, міняючи в ньому місцями задані слова і видаляючи зайві пробіли.

Варіант 16

Написати програму, яка:

виводить текст на екран дисплея;

при натисненні довільної клавіші по черзі виділяє в тексті задане слово (задане слово вводиться з клавіатури);

виводить текст на екран дисплея ще раз, беручи задане слово в лапки, і по черзі виділяє задане слово разом з лапками.

Варіант 17

Написати програму, яка:

виводить текст на екран дисплея;

виводить текст на екран дисплея ще раз, вставляючи в кожне речення останнім задане слово (задане слово вводиться з клавіатури);

при натисненні довільної клавіші по черзі виділяє в тексті вставлене слово.

Варіант 18

Написати програму, яка:

виводить текст на екран дисплея;

при натисненні довільної клавіші по черзі виділяє в тексті зайві пробіли між словами;

виводить текст на екран дисплея ще раз, прибираючи зайві пробіли між словами і починаючи кожне речення з нового рядка.

Варіант 19

Написати програму, яка:

виводить текст на екран дисплея;

при натисненні довільної клавіші по черзі виділяє в тексті задане слово (задане слово вводиться з клавіатури);

виводить текст на екран дисплея ще раз, замінюючи в заданому слові маленькі букви великими.

Варіант 20

Написати програму, яка:

виводить текст на екран дисплея;

визначає найбільшу кількість пробілів у тексті, що йдуть підряд;

при натисненні довільної клавіші по черзі виділяє кожну з послідовностей пробілів максимальної довжини.

Контрольні запитання

1. На які групи можна розділити бібліотечні функції введення-виведення?

2. Що таке потік?

3. Чи можна відкрити потік, не відкриваючи файла?

4. Як відкрити потік у двійковому режимі?

5. Що повертає функція fopen?

6. Яка функція виводить символ у стандартний вивідний потік?

7. Які функції читають і записують рядок у потік?

8. Як встановити покажчик на кінець потоку, відкритого для читання?

9. Як перевірити, чи досяг покажчик потоку кінця файла?

10. Як перевірити, чи немає помилки при роботі з потоком?

11. Чи можна скинути індикатор помилки, не закриваючи потоку?

12. Яка функція виконує форматоване виведення в потік?

13. Чим відрізняється функція printf від функції fprintf?

14. Що таке поле введення для функції fscanf?

15. Як влаштований рядок формату функції fscanf?

Лабораторна робота №3. Підготовка і рішення на ПК завдань з використанням рядків і макросів

Мета лабораторної роботи – знайомство з можливостями препроцесорної обробки даних, освоєння методики обробки інформації з використанням багатофайлових програм.

Перед виконанням лабораторної роботи студент повинен знати:

визначення і використання засобів препроцесорної обробки;

основні засоби роботи з макросами.

Після виконання лабораторної роботи студент повинен уміти розробляти програми для вирішення завдань з використанням багатофайлових проектів.

Теоретичний матеріал

Стадії і команди препроцесорної обробки

В інтегроване середовище підготовки програм на C++ в компілятор мови як обов'язковий компонент входить препроцесор. Призначення препроцесора – обробка початкового тексту програми до її компіляції.

Препроцесорна обробка відповідно до вимог стандарту мови C++ включає декілька стадій, що виконуються послідовно. Конкретна реалі-

зація транслятора може об'єднувати декілька стадій, але результат повинен бути таким, неначебто вони виконувалися послідовно:

усі системно залежні позначення (наприклад, системно залежний індикатор кінця рядка) перекодують у стандартні коди;

кожна пара з символів \' і "кінець рядка" забирається, і тим самим наступний рядок початкового файла приєднується до рядка, в якому знаходилася ця пара символів;

у тексті розпізнаються директиви препроцесора, а кожен коментар замінюється одним символом порожнього проміжку;

виконуються директиви препроцесора і проводяться макропідстановки;

ESC-послідовності в символьних константах і символьних рядках, наприклад '\ п', замінюються на їх еквіваленти (на відповідні числові коди);

суміжні символьні рядки конкатенуються, тобто з'єднуються в один рядок.

Знайомство з переліченими завданнями препроцесорної обробки пояснює деякі угоди синтаксису мови. Наприклад, стає зрозумілим сенс тверджень: кожен символьний рядок може бути перенесений у файлі на наступний рядок, якщо використовувати символ \' або " два символьні рядки, записані поряд, сприймається як один рядок.

Розглянемо детально стадію обробки директив препроцесора. При її виконанні можливі наступні дії:

заміна ідентифікаторів (позначень) наперед підготовленими послідовностями символів;

включення в програму текстів із вказаних файлів;

виключення з програми окремих частин її тексту (умовна компіляція);

макропідстановка, тобто заміна позначення текстом, що параметризується, формованим препроцесором з урахуванням конкретних параметрів (аргументів).

Для управління препроцесором, тобто для завдання потрібних дій, використовуються команди (директиви) препроцесора, кожна з яких поміщається на окремому рядку і починається з символу #. Визначені наступні препроцесорні директиви: #define, #include, #undef, #if, #ifdef, #ifndef, #else, #endif, #elif, #line. #error, #pragma #.

Директива #define має декілька модифікацій. Вони передбачають визначення макросів або препроцесорних ідентифікаторів, кожному з яких

ставиться у відповідність деяка символьна послідовність. У подальшому тексті програми препроцесорні ідентифікатори замінюються на наперед заплановані послідовності символів.

Директива #include дозволяє включати в текст програми текст з вибраного файла.

Директива #undef відміняє дію команди #define, яка визначила до цього ім'я препроцесорного ідентифікатора.

Директива #if і її модифікації #ifdef, #ifndef спільно з директивами #else, #endif, #elif дозволяють організувати умовну обробку тексту програми. Умовність полягає в тому, що компілюється не весь текст, а тільки ті його частини, які так чи інакше виділені за допомогою перелічених директив.

Директива #line дозволяє управляти нумерацією рядків у файлі з програмою. Ім'я файла і початковий номер рядка указуються безпосередньо в директиві #line.

Директива #error дозволяє задати текст діагностичного повідомлення, яке виводиться при виникненні помилок.

Директива #pragma викликає дії, залежні від реалізації.

Директива # нічого не викликає, оскільки є порожньою директивою, тобто не дає ніякого ефекту і завжди ігнорується.

Включення текстів з файлів

Для включення тексту з файла використовується команда #include, що має дві форми запису:

```
#include  <имя_файла>  // ім'я в кутових дужках
#include "имя_файла"     // ім'я в лапках.
```

Якщо "имя_файла" в кутових дужках, то препроцесор розшукує файл у стандартних системних каталогах. Якщо "имя_файла" поміщене в лапки, то спочатку препроцесор проглядає поточний каталог користувача і тільки тоді звертається до проглядання стандартних системних каталогів.

Починаючи працювати з мовою С++, користувач відразу ж стикається з необхідністю використання в програмах засобів введення-виведення. Для цього на початку тексту програми поміщають директиву:

```
#include <iostream.h>
```

Виконуючи цю директиву, препроцесор включає в програму засоби зв'язку з бібліотекою введення-виведення. Пошук файла iostream.h ведеться у стандартних системних каталогах.

За прийнятою угодою суфікс .h приписується тим файлам, які потрібно поміщати в заголовку програми, тобто до виконуваних операторів.

Окрім такого деякою мірою стандартного файла, яким є iostream.h, у заголовок програми можуть бути включені будь-які інші файли (стандартні або підготовлені спеціально).

Заголовні файли виявляються вельми ефективним засобом при модульній розробці великих програм, коли зв'язок між модулями, що розміщуються в різних файлах, реалізується не тільки за допомогою параметрів, але і через зовнішні об'єкти, глобальні для декількох або всіх модулів. Описи таких зовнішніх об'єктів (змінних, масивів, структур і т. п.) поміщаються в одному файлі, який за допомогою директив #include включається у всі модулі, де необхідні зовнішні об'єкти. У той же файл можна включити і директиву підключення бібліотеки функцій введення-виведення.

У заголовному файлі прийнято розміщувати:

визначення типів, що задаються користувачем, констант, шаблонів;

оголошення (прототипи) функцій;

оголошення зовнішніх глобальних змінних (з модифікатором extern);

простори імен.

Заголовний файл може бути, наприклад, таким:

```
#include <iostream.h> // включення засобів обміну
extern int ii, jj, 11;        // цілі зовнішні змінні
extern float AA, BB;     // речовинні зовнішні змінні
```

У практиці програмування на C++ звичайна і в деякому розумінні зворотна ситуація. Якщо в програмі використовується декілька функцій, то іноді зручно текст кожної з них зберігати в окремому файлі. При підготовці програми користувач включає в неї тексти використовуваних функцій за допомогою команд #include.

Умовна компіляція

Умовна компіляція забезпечується в мові C++ набором команд, які, по суті, управляють не компіляцією, а препроцесорною обробкою:

```
#if константний вираз
#ifdef ідентифікатор
#ifndef ідентифікатор
#else
#endif
#elif
```

Перші три команди виконують перевірку умов, дві наступні – дозволяють визначити діапазон дії умови, що перевіряється. Команду #elif розглянемо дещо пізніше. Загальна структура застосування директив умовної компіляції така:

```
#if ... текст 1
#else текст_2
#endif
```

Конструкція #else текст_2 не обов'язкова. Текст_1 включається в компільований текст тільки при істинності умови, що перевіряється. Якщо умова помилкова, то за наявності директиви #else на компіляцію передається текст_2. Якщо директива #else відсутня, то весь текст від #if до #endif за помилкової умови опускається. Відмінність між формами команд #if полягає в наступному.

У першій з перелічених директив #if перевіряється значення константного цілочисельного виразу. Якщо воно відмінне від нуля, то вважається, що умова, яка перевіряється, істинна. Наприклад, в результаті виконання директив:

```
#if 5+4
текст_1
#endif
```

текст_1 завжди буде включений у компільовану програму.

У директиві #ifdef перевіряється, чи визначений за допомогою команди #define до теперішнього моменту ідентифікатор, поміщений після #ifdef. Якщо ідентифікатор визначений як препроцесорний, то текст_1 використовується компілятором.

У директиві #ifndef перевіряється зворотна умова – істинною вважається невизначеність ідентифікатора, тобто той випадок, коли ідентифікатор не був використаний у команді #define або його визначення було відмінено командою #undef.

Умовну компіляцію зручно застосовувати при відладці програм для включення або виключення контрольного друку. Наприклад:

```
#define DE 1
#ifdef DE
cout << " Налагоджувальний друк ";
 #endif
```

Такого друку, що з'являється в програмі залежно від визначеності ідентифікатора DE, може бути декілька і, прибравши директиву #define DE 1, відразу ж відключаємо весь налагоджувальний друк.

Файли, призначені для препроцесорного включення в модулі програми, звичайно забезпечують захистом від повторного включення. Таке повторне включення може відбутися, якщо декілька модулів, у кожному з яких заплановано препроцесорне включення одного і того ж файла, об'єднуються в загальний текст програми. Наприклад, такими засобами захисту забезпечені всі заголовні файли (подібні iostream.h) стандартної бібліотеки. Схема захисту від повторного включення може бути такою:

```
// Файл з іменам filename
#ifndef _FILE_NAME
// Текст файла filename, що включається
#define _FILE_NAME 1
endif
```

Тут _FILE_NAME – зарезервований для файла filename препроцесорний ідентифікатор, який не повинен зустрічатися в інших текстах програми.

Для організації розгалужень під час обробки препроцесором початкового тексту програми введена директива:

#elif константний вираз.

Структура початкового тексту із застосуванням цієї директиви така:

```
#if
текст_для_ if
#elif вираз_1
текст_1
#elif вираз_2
текст_2
#else
текст_для_випадку_else
#endif
```

Препроцесор перевіряє спочатку умову в директиві #if, якщо вона помилкова (рівна 0) – обчислює вираз_1, якщо вираз_1 дорівнює 0 – обчислюється вираз_2 і т. д. Якщо всі вирази помилкові, то в компільований текст включається текст_для_випадку_e1se. Інакше, тобто при

появі хоч би одного дійсного виразу (у #if або в #elif) починає оброблятися текст, розташований безпосередньо за цією директивою, а вся решта директив не розглядається. Таким чином, препроцесор обробляє завжди тільки одну з ділянок тексту, виділених командами умовної компіляції.

Макропідстановки засобами препроцесора

Макрос, за визначенням, є засіб заміни однієї послідовності символів іншою. Для виконання замін повинні бути задані відповідні макроозначення. Просте макроозначення ми вже ввели, розглядаючи директиву

#define ідентифікатор рядок заміщення

Така директива зручна, проте вона має істотний недолік – рядок заміщення фіксований. Великими можливостями володіє наступне макроозначення з параметрами

#define ім'я (список_параметрів) рядок__заміщення

Тут ім'я – ім'я макросу (ідентифікатор), список_параметрів – список розділених комами ідентифікаторів. Між іменем макросу і списком параметрів не повинно бути пробілів.

Класичний приклад макроозначення:

#define max(a,b) (a < b ? b : a)

дозволяє формувати в програмі вираз, який визначає максимальне з двох значень аргументів. При такому визначенні входження в программу max(X,Y) замінюється виразом (X < Y ? Y : X), а використання max(Z,4) приведе до формування виразу (Z < 4 ? 4 : Z).

У першому випадку при дійсному значенні X < Y повертається значення Y, навпаки – значення X. У другому прикладі Z порівнюється з константою 4 і вибирається більше із значень.

Не менш часто використовується визначення

#define ABS(X) (X < 0 ? - (X) : X).

За його допомогою в програму можна вставляти вираз для визначення абсолютних значень змінних. Конструкція ABS(E - Z) замінюється виразом (E - Z < 0 ? -(E - Z) : E-Z), у результаті обчислення якого визначається абсолютне значення виразу E – Z.

Порівнюючи макроси з функціями, найчастіше відзначають, що на відміну від функції, визначення якої завжди присутнє в одному екземплярі, коди, що формуються макросом, вставляються в програму стільки разів, скільки разів використовується макрос. У цьому відношенні макроси подібні вбудовуваним (Online) функціям, але на відміну від вбудовуваних

функцій підстановка для макросу виконується завжди. Звернемо увагу на ще одну відмінність: функція визначена для даних того типу, який вказаний у специфікації її параметрів і повертає значення тільки одного конкретного типу. Макрос придатний для обробки параметрів будь-якого типу, допустимих у виразах, що формуються при обробці рядка заміщення. Тип набуваючого значення залежить тільки від типів параметрів і від самих виразів. Таким чином, макрос може замінювати декілька функцій. Наприклад, наведені макроси max () і ABS() правильно працюють для параметрів з цілими або плаваючими типами, а результат залежить тільки від типів параметрів. Механізм перевантаження і шаблони функцій дозволяють вирішувати ті ж завдання, що і макроси. Саме тому на відміну від мови С в програмах на С++ макрозасоби використовуються рідше.

Завдання до лабораторної роботи

Виконати вправи лабораторної роботи, оформивши програмний продукт у вигляді багатофайлового проекту.

Усі необхідні дані для функцій повинні передаватися їм як параметри. Використання глобальних змінних у функціях забороняється.

Варіанти для виконання завдання знаходяться в лабораторній роботі №2.

Контрольні запитання

1. Дайте визначення макросу.
2. Для чого використовується умовна компіляція?
3. Як можно захиститись від повторного включення модулів у заголовних файлах?
4. Які дії можно виконувати за допомогою директив препроцесора?
5. Що розміщується в заголовних файлах?

Лабораторна робота №4. Підготовка і рішення на ПК завдань обробки масивів структур

Мета лабораторної роботи – освоїти основні прийоми застосування структур для вирішення типових економічних завдань.

Після виконання лабораторної роботи студент повинен знати:

основні правила роботи зі структурними типами даних;

особливості застосування маніпуляторів потоку setw і setprecision.

Після виконання лабораторної роботи студент повинен уміти:

обґрунтовувати доцільність застосування структурних типів даних при рішенні типових економічних завдань;

описувати нові типи і створювати відповідні екземпляри структур;

здійснювати доступ до елементів даним (полям) структури;

створювати табличні документи у вигляді послідовного масиву записів, програмно оформлених у вигляді відповідних масивів структур.

Теоретичний матеріал

Структури – це складені типи даних, побудовані з використанням інших типів. Вони є об'єднаним загальним ім'ям набором даних різних типів. Саме тим, що в структурах можуть зберігатися дані різних типів, вони і відрізняються від масивів, які зберігають дані одного типу.

Окремі дані структури називаються *елементами,* або *полями*. Елементи однієї і тієї ж структури повинні мати унікальні імена, але дві різні структури можуть містити не конфліктуючі елементи з однаковими іменами.

Відповідно до синтаксису мови опис структури починається із службового слова struct, услід за яким поміщається вибране користувачем ім'я типу. Елементи, що входять у структуру, поміщаються у фігурні дужки, услід за якими ставиться крапка з комою, елементи структури можуть бути як базових, так і похідних типів.

Опис структури не резервує ніякого простору в пам'яті, він тільки створює новий тип даних, який може використовуватися для визначення змінних. У структурі обов'язково повинен бути вказаний хоч би один компонент.

Припустимо, що необхідно створити якийсь тип для опису студента університету. Цей тип повинен містити ім'я студента, його адресу, вік і успішність. Нижче наведено опис структури, що задовольняє цим вимогам:

```
struct  Student
  {
    char Name[20];          //  Ім'я
```

```
  char Address[30];       //  Адреса
  int  Age;               //  Вік
 double Rating;           //  Успішність
};
```

Ключове слово struct указує на те, що код визначає формат структури. Ідентифікатор Student – назва, або тег, для цього формату. Таким чином, тепер можна створювати змінні типу Student так само, як змінні будь-якого базового типу, наприклад int або char. Між фігурними дужками знаходиться список типів заданих даних. Кожен елемент списку – це оператор визначення. Тут можна використовувати будь-який з типів даних C++, включаючи масиви і інші структури. У даному прикладі використовуються два масиви типу char, зручні для збереження рядків з атрибутами "Ім'я" і "Адреса", а також int і double – для зберігання відповідних числових значень.

Тепер, коли структура описана, її можна використовувати. Спочатку потрібно створити (визначити) екземпляр структури. Це виглядає таким чином:

```
Student  BestStudent;
```

Для доступу до елементів структури використовуються операції доступу до елементів: операція крапка (.) і операція стрілка (->).

Операція стрілка забезпечує доступ до елемента структури через покажчик на об'єкт (у даній роботі цей тип доступу не використовується).

Операція крапка звертається до елемента структури на ім'я об'єкта (екземпляра структури) або за посиланням на об'єкт. Наприклад:

```
strcpy(BestStudent.Name, "ІВАНОВА");
strcpy(BestStudent.Address, "ПР. ЛЕНІНА, 9-А");
BestStudent.Age = 17;
BestStudent.Rating = 10.57;
```

У даному фрагменті програми відбувається роздільна ініціалізація елементів-даних екземпляра BestStudent структури Student. Причому, для запису у відповідні поля рядків символів, що позначають ім'я (прізвище) і адресу студента, використовується функція strcpy( ).

Оператор

```
strcpy(BestStudent.Name, "ІВАНОВА");
```

забезпечує виклик цієї функції, звернення до поля Name екземпляра BestStudent і запис в це поле рядка "ІВАНОВА".

Оператор

```
BestStudent.Age = 17;
```

ініціалізував елемент, даного з ім'ям Age (вік) значенням 17.

При необхідності можлива одночасна ініціалізація відразу всіх елементів-даних знов створеного екземпляра структури. Наприклад:

```
BestStudent = {"ІВАНОВА", "ПР. ЛЕНІНА 9-А", 17, 10.57};
```

Нижче наведений приклад програми, яка створює, ініціалізує і виводить на друк вміст структури, використовуваної для опису розглянутих вище реквізитів студента.

```
// Програма 1
// Елементарна обробка полів  структур
#include <iostream>
#include <string.h>          //  Для роботи з функцією strcpy();
        struct Student
 {
           char Name[20];          // Ім'я
           char Address[30];       // Адреса
          int  Age;                // Вік
          double Rating;           // Успішність
   } BestStudent;

int main()
{
        BestStudent = {"ІВАНОВА", "ПР. ЛЕНІНА 9-А", 17, 10.57};
// Виведення на друк вмісту полів екземпляра структури
       cout << "      СТУДЕНТ Ф_ТА ЭІ \n";
       cout << "\n ИМЯ:\t" << BestStudent.Name
        << "\n Адрес:\t" << BestStudent.Address
        << "\n Возраст:\t"  << BestStudent.Age
        << "\n Успеваемость:\t" << BestStudent.Rating << endl;
return 0;
}
```

Масиви структур

Методика створення масивів структур така сама, як і при створенні масивів. Наприклад, щоб створити масив зі 100 екземплярів BestStudent структури Student, необхідний наступний запис:

```
Student  BestStudent[100];
```

У результаті буде створений стоелементний масив BestStudent типу Student. Отже, кожен елемент масиву, такий, як BestStudent[0] або BestStudent[99] – це об'єкт типу Student, і доступ до його полів можна організувати вже розглянутим вище способом:

```
cin >> BestStudent[0].Name;        // ініціалізація поля Name
cout << BestStudent [99].Rating; //виведення на екран вмісту поля Rating
```

Слід мати на увазі, що сам BestStudent – це масив, а не структура, так що конструкції типу BestStudent.Name є помилковими.

Розглянемо наступний приклад. Потрібно розробити програму, що послідовно формує в процесі діалогу з користувачем документ такого змісту (табл. 4.1).

Таблиця 4.1

Відомості про вартість виданих деталей

n/n	ДЕТАЛЬ	ВАРТІСТЬ	ВИДАНО	ВИТРАТА
n	a[8]	t	k	c = t * k
Разом:		s1	s2	s3

Діалог повинен здійснюватися за наступним сценарієм:

ФОРМУВАННЯ ДОКУМЕНТА
ДОКУМЕНТ ПОВИНЕН МІСТИТИ 3 **РЯДКИ-ЗАПИСИ:**
ФОРМУВАННЯ 1 ЗАПИСУ
ВВЕСТИ НАЗВУ ДЕТАЛІ (НЕ БІЛЬШЕ 8 СИМВОЛІВ): СТАТОР
ВКАЖІТЬ ВАРТІСТЬ ДЕТАЛІ: 332.5
НЕОБХІДНА КІЛЬКІСТЬ (ЦІЛЕ ЧИСЛО): 5
ФОРМУВАННЯ 2 ЗАПИСУ
ВВЕСТИ НАЗВУ ДЕТАЛІ (НЕ БІЛЬШЕ 8 СИМВОЛІВ): КОРПУС
і т. д.

При цьому інформація про кожну деталь повинна зберігатися в одному блоці (записи), що дозволяє індивідуальну обробку, а кількість записів у документі – визначатися в програмі у вигляді відповідної константи.

З постановки завдання виходить, що запис містить декілька різнотипних полів: порядковий номер деталі (n/n), для його зберігання доцільно використовувати тип int; найменування деталі (ДЕТАЛЬ) – тут може використовуватися масив, наприклад з 8-ми символів; кількість виданих деталей (ВИДАНО) – даному полю відповідає тип int; а полю вартості деталі (ВИТРАТА) – тип double. Очевидно, що застосування масиву в даному випадку не вважається можливим. Дійсно, масив може містити декілька елементів, проте кожен елемент повинен бути одного типу, що суперечить умові завдання. Виходом із ситуації, що створилася, буде застосування структури – більш універсальної форми даних, чим масив.

Нижче наведений один з можливих варіантів програми рішення поставленого завдання на основі застосування масиву структур.

```
// Програма 2
// Рішення економічного завдання з використанням  масивів структур

#include <iostream>
#include <iomanip> // Для використання маніпуляторів потоку setw()
                   // і setprecision()
const int  n=3;    // Розмірність оброблюваного масиву структур
   struct zap      // Робочий варіант оголошення структури zap
   {
       char a[8];
       double t;
       int к;
       double c;
    } ;
int main()
{
  double s1,s2,s3;  // Розрахункові суми
    zap zapis[n];    // Окреме визначення масиву zapis[n] екземплярів
                     // структур  типу zap
//  Введення початкових даних:
        cout << "ФОРМУВАННЯ ДОКУМЕНТА: \n";
        cout << "ДОКУМЕНТ ПОВИНЕН МІСТИТИ " << n
        << " РЯДКИ - ЗАПИСИ: \n";
```

```
for( int i=0; i < n; i++)
     {
 cout << "\nФОРМУВАННЯ  " << i + 1 <<  "  ЗАПИСУ";
 cout << "\nВВЕСТИ НАЗВУ ДЕТАЛІ (НЕ БІЛЬШЕ 8 СИМВОЛІВ):   ";
cin >>  zapis[i].a; // Доступ до елемента a екземпляра структури
                    // виконується операцією крапка (.)
out << "\nВКАЖІТЬ ВАРТІСТЬ ДЕТАЛІ  : ";
cin >> zapis[i].t; //Доступ до елемента t екземпляра структури zapis[i]
 cout << "\nНЕОБХІДНА КІЛЬКІСТЬ  (ЦІЛЕ ЧИСЛО): \n";
 cin >>  zapis[i].k; //  Доступ до елемента екземпляра структури
}
 //  Виконання розрахунків:
      s1=0;s2=0;s3=0;
     for( i=0; i < n; i++)
       {
          zapis[i].c = zapis[i].t * zapis[i].k;
               s1 += zapis[i].t;
              s2 += zapis[i].k;
              s3 += zapis[i].c;
       }
 //  Побудова "шапки" таблиці
 clrscr();
 cout << "\n   ВІДОМОСТІ ПРО ВАРТІСТЬ ВИДАНИХ ДЕТАЛЕЙ    ";
 cout << "\n|-------------------------------------------------------------------------|";
 cout << "\n|  n/n  |  ДЕТАЛЬ   |   ВАРТІСТЬ  |  ВИДАНО  | ВИТРАТА |";
 cout << "\n|-------------------------------------------------------------------------|";
 // Заповнення таблиці даними:
   for(i=0;i<n;i++)
    cout << "\n|" << setw(7)<< i+1 << "|" << setw(8)<< (zapis[i]).a << "|"
 //                    ------
 //                    |->  маніпулятор потоку, вказує, що наступна
 //                   вихідна  величина (i+1) буде надрукована з шириною
 //                    поля, рівної 7 символам
    << setw(13)<< setprecision(2)<< (zapis[i]).t << "|"
 //                         --------------
```

```
//                    |->  маніпулятор потоку вказує, що наступні
//                     вихідні  будуть надруковані з двома цифрами після
//                     десяткової точки
    << setw(8)<< setprecision(2)<< (zapis[i]).k << "|"
   << setw(8)<< setprecision(2)<< (zapis[i]).c << "|";
  cout << "\n|-------------------------------------------------------------------------| ";
  cout << "\n|  РАЗОМ:        |" << setw(13)<< setprecision(2)<< s1
  << "|" << setw(8)<< s2<< "|" <<setw(8) << setprecision(2)<< s3 << "|";
   cout << "\n|-------------------------------------------------------------------------|";
    return 0;
}
```

Для форматування виведення інформації на монітор у програмі використовуються маніпулятори потоку setw і setprecision. Їх призначення наведене у відповідних коментарях програми 1.

Завдання до лабораторної роботи

Розробити програму для обробки відомості (дивіться варіанти завдань). Програма повинна забезпечувати:

введення початкових даних відомості з консолі в масив, що складається з декількох екземплярів структури;

обробку і виведення даних відповідно до варіанта завдання;

виведення на консоль будь-якого із записів відомості за критерієм збігу значення одного з полів запису з даними запиту, що вводиться з консолі;

виведення на консоль повідомлення за відсутності інформації, відповідної до критерію відбору.

Варіант 1

Відомість нарахування зарплати співробітникам підприємства:

№ з/п	Прізвище	Зарплата, грн	Утримано, грн	Видано, грн
1. 2	F	Z	P	S = Z - P
	Разом	S1	S2	S3

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для співробітників, фактична зарплата яких не перевищує значення, введеного з консолі;

записи відомостей повинні бути відсортовані за збільшенням фактичної зарплати.

Варіант 2

Відомість витрати палива на автобазах міста:

№ з/п	Автобаза	Витрачено палива (кг)	Кількість автомашин	Середня витрата (кг)
1. 2	A	T	K	C = T \ K
	Разом	S1	S2	S3

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для автобаз, витрата палива на яких більше, а кількість автомашин менше відповідних значень, введених з консолі;

записи відомостей повинні бути відсортовані за убуванням витрати палива.

Варіант 3

Відомість використання машинного часу на обчислювальному центрі:

№ з/п	Кафедра	Використання машиного часу (год.)		Відхилення від плану	
		за планом	фактично	у годинах	у %
1	K	P	F	$O_1 = P - F$	$O_2 = O_1 \cdot 100 / P$
2					
	Разом	$\sum$	$\sum$		

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для кафедр, фактичне використання машинного часу якими перевищує заплановане;

записи відомостей повинні бути відсортовані за збільшенням фактичного використання машинного часу.

Варіант 4

Відомість споживання електроенергії на заводах міста:

№ з/п	Завод	Потреба в електроенергії, кВт/ч		Відхилення від плану	
		за планом	фактично	у кВт/год.	у %
1	Z	P	F	$O_1 = P - F$	$O_2 = O_1 \cdot 100 / P$
2					
…					
	Разом	$\sum$	$\sum$		

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для заводів, де перевитрата електроенергії (у %) більше значення, введеного з консолі;

записи відомостей повинні бути відсортовані за убуванням відхилення витрат електроенергії (у %).

Варіант 5

Відомість руху матеріалів на складах підприємства за звітний період:

№ з/п	Склад	Рух матеріалів за період, грн			Залишок на кінець періоду
		залишок на початок періоду	отримано	видано	
1	C	Oc	P	V	R = Oc + P - V
2					
…					
	Разом	$\sum$	$\sum$	$\sum$	$\sum$

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для всіх складів, де залишки матеріалів на кінець періоду більше значення, введеного з консолі;

записи відомостей повинні бути відсортовані за збільшенням залишків матеріалів на кінець періоду.

Варіант 6

Відомість прибутків підприємства за звітний період за видами продукції:

№ з/п	Продукція	Кількість (шт.)	Оптова ціна (грн)	Собівартість (грн)	Прибуток (грн)
1. 2	A	T	K	C = T \ K	
	Разом	S1	S2	S3	

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для збиткових видів продукції;

записи відомостей повинні бути відсортовані за убуванням прибутку.

Варіант 7

Відомість об'єму постачань продукції в натуральному і вартісному виразах:

№ з/п	Продукція	Шифр	Об’єм постачань	Оптова ціна (грн)	Об’єм (грн)
1. 2	P	H	V	Z	O = V·Z
	Разом		S1	S2	S3

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для видів продукції, об'єм постачання яких у вартісному виразі знаходиться в інтервалі, межі якого вводяться з консолі;

записи відомостей повинні бути відсортовані за збільшенням об'єму постачання у вартісному виразі.

Варіант 8

Відомість відвідування занять студентами:

№ з/п	Прізвище	Пропущено годин		Відхилення від плану	
		за планом	фактично	у годинах	у %
1	F	V	O	$P_1 = V - O$	$P_2 = P_1 \cdot 100 / V$
2					
…					
	Разом	$\sum$	$\sum$		

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості про студентів, для яких відсоток пропуску занять з неповажних причин не менше значення того, що вводиться з консолі;

записи відомостей повинні бути відсортовані за убуванням відсотка пропуску занять з неповажних причин.

Варіант 9

Відомість розрахунку середньої вартості перевезення авіапасажирів:

№ з/п	Тип літака	Рейс	Витрати на рейс (грн)	Кількість пасажирів	Середня вартість (грн)
1. 2	T	R	Z	K	S= Z / K
	Разом		S1	S2	S3

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для рейсів, на яких середня вартість перевезення більше значення того, що вводиться з консолі;

записи відомостей повинні бути відсортовані за збільшенням середньої вартості перевезення.

Варіант 10

Відомість обліку часу роботи верстатів підприємства:

№ з/п	Тип станка	Час роботи (год.)		Відхилення від плану	
		за планом	фактично	у годинах	у %
1.	Z	P	F	$O_1 = P - F$	$O_2 = O_1 \cdot 100 / P$
	Разом	$\sum$	$\sum$		

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості по верстатах, які простоювали (у %) більше значення того, що вводиться з клавіатури;

записи відомостей повинні бути відсортовані за збільшенням відхилення від плану в %.

Варіант 11

Відомість випуску деталей робітниками цеху:

№ з/п	Прізвище	Кількість деталей (шт.)		Брак	
		виготовлено	прийнято	шт.	у %
1	Z	P	F	$O_1 = P - F$	$O_2 = O_1 \cdot 100 / P$
2					
	Разом	∑	∑	∑	

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості по робітниках, для яких відсоток випуску бракованих деталей не менше або не більше значення того, що вводиться з клавіатури;

записи відомостей повинні бути відсортовані за збільшенням відсотка браку.

Варіант 12

Відомість наявності і руху основних фондів підприємства:

№ з/п	Фонд	Наявність на початок року (шт.)	Надійшло (шт.)	Вибуло (шт.)	Наявність на кінець року (шт.)
1. 2	F	N1	P	V	N2 = N1 + P - V
	Разом		S1	S2	S3

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості по фондах, для яких відсоток вибуття більше значення того, що вводиться з клавіатури;

записи відомостей повинні бути відсортовані за збільшенням вибуття фондів.

Варіант 13

Відомість нарахування зарплати співробітникам підприємства:

№ з/п	Прізвище	Нараховано (грн)		Виплати	
		Оклад	Аванс	Премія, %	До виплати
1.	Z	P	F	Pp	P +P · Pp / 100 - F
2					
	Разом	∑	∑	∑	

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості по робітниках, для яких премія не менше або не більше значення того, що вводиться з клавіатури;

записи відомостей повинні бути відсортовані за збільшенням окладу.

Варіант 14

Відомість продажу запчастин в автомагазині за звітний період:

№ з/п	Запчастина	Вид автомобіля	Вартість запчастини	Кількість проданих запчастин	Сума продажу
1. 2	A	T	Д1	Д2	C = Д1 · Д2

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для запчастин, які мають максимальну і мінімальну суму продажу або не меньше значення того, що вводиться з клавіатури;

записи відомостей повинні бути відсортовані за збільшенням вартості запчастини.

Варіант 15

Відомість термінів ремонту обладнання підприємства за звітний період:

№ з/п	Обладнання	Вид ремонту	Дата прийняття на ремонт	Дата виконання ремонту	Термін виконання ремонту
1. 2. 3	A	T	Д1	Д2	C = Д1 - Д2

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для терміну ремонту не більше значення того, що вводиться з клавіатури;

записи відомостей повинні бути відсортовані за убуванням терміну ремонту.

Варіант 16

Відомість поставок товарів клієнтам підприємства за звітний період:

№ з/п	Товар	Клієнт	Вартість за одиницю	Кількість	Дата поставки	Вартість поставки
1. 2. 3	А	Т	С	К	Д	С1 =С · К
	Разом					∑

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для дати поставки не меньше значення того, що вводиться з клавіатури;

записи відомостей повинні бути відсортовані за збільшенням вартості поставки.

Варіант 17

Відомість оплати ремонту обладнання підприємства за звітний період:

№ з/п	Обладнання	Вид ремонту	Вартість комплектуючих	Вартість виконання робіт	Дата оплати ремонту
1. 2. 3	А	Т	С1	С2	Д
	Разом		∑	∑	

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для терміну оплати ремонту не більше або не менше значення того, що вводиться з клавіатури;

записи відомостей повинні бути відсортовані за зменшенням дати оплати ремонту.

Варіант 18

Відомість проходження технічного огляду автотранспорту за звітний період:

№ з/п	Автомобіль	Власник	Дата технічного огляду	Дата проходження технічного огляду	Штраф за несвоєчасний технічний огляд
1. 2. 3. 4	А	Т	Д1	Д2	С = (Д1 - Д2) ·10

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для дати проходження технічного огляду не більше або не меньше значення того, що вводиться з клавіатури;

записи відомостей повинні бути відсортовані за збільшенням штрафу.

Варіант 19

Відомість заявок авіабілетів підприємства за звітний період:

№ з/п	Пункт призна-чення	№ рейсу	Прізвище пасажира	Дата відправлення	Вартість квітка	Додана вартість
1. 2. 3. 4	A	T	B	Д	C	C1 = C·0,2
	Разом				$\sum$	

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для дати відправлення не більше та не меньше значення того, що вводиться з клавіатури;

записи відомостей повинні бути відсортовані за збільшенням дати відправлення.

Варіант 20

Відомість телефонних переговорів абонента за звітний період:

№ з/п	Абонент	№ телефону	Тривалість переговорів (хв.)	Дата переговорів	Тариф, грн\хв.	Вартість переговорів
1. 2. 3. 4	A	T	B	Д	C	C1 = C·B
	Разом					$\sum$

Вимоги до програми:

виведення на консоль сформованої відомості і виписки з відомості для дати не більше та не меньше значення того, що вводиться з клавіатури;

записи відомостей повинні бути відсортовані за зменшенням дати переговорів.

Контрольні запитання

1. Структура як тип даних.
2. Сформулювати відмінності структури від масиву.
3. Як передати структуру у функцію?
4. Навести приклад передачі структури у функцію за значенням.
5. Навести приклад передачі структури у функцію за допомогою покажчика.
6. Навести приклад опису масиву структур.

Лабораторна робота №5.
Підготовка і рішення на ПК завдань обробки масивів структур з використанням контейнерів

Мета лабораторної роботи – придбання практичних навичок роботи з динамічними структурами даних.

Перед виконанням лабораторної роботи студент повинен знати основи застосування елементів бібліотеки стандартних шаблонів (STL).

Після виконання лабораторної роботи студент повинен уміти розробляти типові програми з використанням контейнерів даних мовою С++.

Короткі теоретичні відомості

STL – це бібліотека стандартних шаблонів. Вона містить, наприклад, способи організації даних, що часто зустрічаються, так називані контейнери: динамічні масиви, двонаправлені списки, стеки 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. Ви, наприклад, можете використати той самий алгоритм сортування як для динамічного масиву, так і для стека.

STL складається із трьох основних частин: контейнери, алгоритми й ітератори.

Частина перша. Вона містить різноманітні контейнери. Частина контейнерів зрозуміла, це динамічні масиви, списки, черги та ін. Інша частина більше просунута, якщо можна так виразитися. Сюди, наприклад, відносяться так називані асоціативні контейнери. Основна їхня відмінна риса – це те, що значення, які зберігаються в них, шукаються по ключах. При цьому ключ може бути самим різним. Аналогія такого контейнера з життя – це телефонна книга. Там номера телефонів шукаються на прізвище власника або назви фірми.

У кожному контейнері, крім властне даних, є методи для роботи із цими даними (для додавання, пошуку, видалення та ін.).

Друга частина бібліотеки STL – це алгоритми. Алгоритми не є частиною контейнерів, а утворюють окрему підсистему. Але при цьому майже будь-який алгоритм може застосовуватися до майже будь-якого контейнера. Тобто, викликаючи метод для деякого алгоритму, ми викликаємо цей метод сам по собі, а не для екземпляра деякого класу. Контейнер же, до якого застосовується алгоритм, передається як параметр.

І, нарешті, третя частина STL – це різноманітні ітератори. У першому наближенні ітератор – це деякий покажчик, що може оббігати всі елементи контейнера. Ітератори відіграють такі ж ролі, що й індекс у елемента масиву. Через індекс масиву ми можемо одержати деякий елемент масиву, і через ітератор ми можемо одержати деякий елемент контейнера. Ітератори бувають різних типів: для руху тільки вперед, для руху в обидва боки та ін.

Приклад контейнера vector:

Vector є не чим іншим, як динамічним одновимірним масивом, тобто ви можете додавати в нього елементи, видаляти їх і т. д.

Приклад використання цього шаблону:

```
#include <iostream>
// Додаємо потрібний простір імен.
#include <vector>
using namespace std;
void main()
{
   // Повідомляємо вектор із цілих.
   vector <int> k;
   // Додаємо елементи в кінець вектора.
```

```
    k.push_back(22);
    k.push_back(11);
    k.push_back(4);
    // Показуємо всі елементи вектора.
    for (int i = 0; i<k.size(); i++)
    {
       cout<<k[i]<<"\n";
    }
    cout<<"***\n";
    // Видаляємо елемент із кінця вектора.
    k.pop_back();
    // Показуємо всі елементи вектора.
    for (i = 0; i<k.size(); i++)
    {
       cout<<k[i]<<"\n";
    }
    cout<<"***\n";
    // Видаляємо всі елементи вектора
    k.clear();
    // Перевіряємо, що вектор порожній.
    if(k.empty)
    {
       cout<<"Vector is empty\n";
    }
}
```

Використані методи й змінні шаблона vector (push_back, pop_back, clear й empty) досить ясні з коментарів. Зверніть ще увагу, що для доступу до окремих елементів вектора, ми використаємо оператор [] як і для елементів масиву. Також зверніть увагу, що ми повинні підключити потрібний простір імен (vector).

Виведіть цей код, як і слід було сподіватися, спочатку 22, 11 й 4, потім, після видалення останнього елемента вектора, 22 й 11, і потім, після видалення всіх елементів вектора, напис "Vector is empty".

А от ще які класи-контейнери є в STL:

list – двонаправлений список. Доступ до елементів можливий тільки послідовно із двох сторін списку;

stack – стік;

map – асоціативний список. Зберігає дані у вигляді пари Джерело-Значення (з кожним ключем пов'язане тільки одне значення);

multimap – асоціативний список. Зберігає дані у вигляді пари Джерело-Значення (з кожним ключем пов'язано кілька значень);

set – безліч. Усі його елементи унікальні;

multiset – безліч. У ньому можуть бути співпадаючі елементи;

queue – черга;

deque – двостороння черга;

bitset – набір бітів (кожний з яких відповідає за відсутність / наявність чого-небудь).

Для використання цих шаблонів ви повинні написати на початку програми потрібний include. Як правило, include пишеться такий же, як ім'я шаблона (так, наприклад, для використання шаблона list треба написати #include <list>). Два винятки: для використання multimap треба в include додати map і для multiset – set.

Отже, контейнер – шаблон класу, що слугує для зберігання даних. Простими словами: контейнер потрібний нам як "обгортка" навколо наших даних, щоб зберігати їх, наприклад, у списку. Такий список можна організувати із чисел float, з рядків string, з об'єктів створеного нами класу (структури) і т. д.

Приклад ітератора:

У першому наближенні ітератор – це покажчик на деякий елемент у контейнері. І, як й у випадку з покажчиками, добратися до елемента контейнера можна через безіменний ітератор.

Приклад коду:

```
#include <iostream>
#include <vector>
using namespace std;
void main()
{
   // Повідомляємо вектор із цілих.
   vector <int> k;
   // Додаємо елементи в кінець вектора.
   k.push_back(22);
   k.push_back(11);
   k.push_back(4);
```

```
    // Повідомляємо ітератор.
    vector <int>::iterator p;
    // Установлюємо ітератор у початок
    // і пересуваємо його в циклі
    // на наступну позицію.
    for (p = k.begin(); p < k.end(); p++)
    {
       // Виводимо вміст елементів вектора
       // через безіменний ітератор.
       cout<<*p<<"\n";
    }
}
```

Вихід: 22, 11 й 4.

Ітератор – спеціальний тип, що слугує аналогом С-го покажчика, використовується для проходу за списком, масиву, пошуку елемента в асоціативному списку і т. д. Це дійсний аналог покажчика, до нього можна застосовувати операції ++ й -- , "зірочку" або -> для одержання значення. Операція ++ зрушує ітератор до наступного елемента динамічного масиву, списку, черги й т. п. Можна порівнювати два ітератори. Замість NULL, ознакою виходу за межі припустимих значень служить спеціальний ітератор, одержаний з контейнера за допомогою функції end().

Приклад алгоритму:

Розглянемо застосування алгоритмів на прикладі сортування елементів вектора.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

void main() {
    // Повідомляємо вектор із цілих.
    vector <int> k;
    // Додаємо елементи в кінець вектора.
    k.push_back(22);
    k.push_back(11);
    k.push_back(4);
```

```
    k.push_back(100);
    vector <int>::iterator p;
    // Виведення невідсортованого вектора.
    for (p = k.begin(); p<k.end(); p++)
    {
       cout<<*p<<"\n";
    }
    // Сортування вектора.
    sort(k.begin(), k.end());
    // Виведення відсортованого вектора.
    cout<<"sorted:\n";
    for (p = k.begin(); p<k.end(); p++)
    {
       cout<<*p<<"\n";
    }
}
```

Як ви бачите, головна частина в нашій програмі – це рядок

```
    ...
    // Сортування вектора.
    sort(k.begin(), k.end());
    ...
```

При виклику функції sort ми вказуємо, від якого й до якого елементів ми робимо сортування. У нашому випадку ми сортуємо наш вектор від початку й до кінця.

Результатом виконання програми буде висновок чисел 4, 11, 22, 100 (саме в такому, відсортованому порядку).

Зверніть увагу, що функція sort не належить нашому вектору. Тобто ми не пишемо щось начебто

```
    ...
    k.sort(...);
    ...
```

Тобто наш метод сортування (як й інші алгоритми) утворить окрему підсистему в бібліотеці STL (поряд, наприклад, з контейнерами – тим же вектором, наприклад).

Алгоритм – шаблон функції, що виконує конкретне завдання (пошук, сортування й т. д.) над будь-якими контейнерами. Функція-алгоритм приймає як параметри ітератори або покажчики.

Приклад зберігання об'єктів у списку / динамічному масиві:

```
#include < list>
#include < vector>
using namespace std;
 struct  MyStruct
 {
    int      A;
    float   B;
    char   C[50];
 }
 void main()
 {
    vector < MyStruct>  arr;

    MyStruct   obj;
    arr.push_back(obj);
    arr[0].A = 123;
    list  < MyStruct>  thelist;
    thelist.push_back(obj);
    thelist.push_back(obj);
    thelist.push_back(obj);
    for(list< MyStruct>::iterator   i = thelist.begin(); i != thelist.end(); i++)
    {
          // можна звернутися до структури так:
          (*i).A = 123;
          // а можна одержати посилання й працювати з нею:
          MyStruct   &thestruct  =  *i;
          thestruct.B = -0.234;
          strcpy(thestruct.C, "Some text");
    }
 }
```

Приклад організації списка списків, масива масивів і т. п.

У будь-якому контейнері можна зберігати вкладений контейнер. Для простоти запису можна використати typedef:

```
#include < list>
#include < vector>
 using namespace std;
```

```
// відключаємо попередження через занадто довгі імена типів
#pragma warning(disable: 4786)
   void main()
{
    // список дійсних
    typedef   list< float>   ListOfFloat;
   // список списків дійсних
    typedef   list< ListOfFloat>  ListOfListOfFloat;
    // як можна працювати зі списком списків:
    ListOfListOfFloat  thelist;
    // додаємо 2 елементи в список списків:
    thelist.resize(2);
    // а можна й так:
    ListOfFloat  thelistinside;
    thelistinside.push_back(-0.34f);
    thelistinside.push_back(3.45f);
    thelist.push_back(thelistinside);
    // або так:
    thelist.push_back(ListOfFloat());
    // проходимо в циклі по списках – зовнішньому й вкладеному
    for(ListOfListOfFloat::iterator  i1  =  thelist.begin();  i1  !=  thelist.end();
i1++)
    {
        // для зручності одержуємо посилання на вкладений список
        ListOfFloat  &thelistinside = *i1;
        // проходимо по вкладеному списку
        for(ListOfFloat::iterator  i2      =      thelistinside.begin(); i2      !=
thelistinside.end(); i2++)
        {
            float  value = *i2;
        }
        }
        }
```

Пояснення:

- створення списку й додавання елементів за аналогією з vector;
- функції resize(), push_front(), clear(), begin() та ін. є в кожному контейнері (в vector у тому числі);

- copy(A, B, C) – алгоритм, що копіює всі елементи з контейнера або масиву, на який указує ітератор або покажчик C, у контейнер або масив, на який указують літератор B або покажчики A й B, причому A указує на перший елемент, B указує на елемент, що перебуває після останнього елемента масиву або контейнера. Для будь-якого контейнера B можна одержати за допомогою функції end(), для масиву – можна додати розмір масиву до покажчика на перший елемент (наприклад, values + 3);

- ітератор ("псевдопокажчик") для кожного контейнера, створюваного нами, має свій відмінний тип. Для того, щоб оголосити ітератор i з контейнера

list< float>,

використовується вираз;

list< float> ::iterator i;

- як уже говорилося, для одержання значення, на яке вказує ітератор, використовується зірочка: *i. Для переміщення за списком ми просто зрушуємо ітератор операцією ++;

- для того, щоб видалити елемент із "голови" або "хвоста", використовується пара функцій pop_front(), pop_back(). Для видалення довільного елемента використовується функція erase(i), де i – це ітератор, що вказує на елемент;

- для вставки елемента в середину списку або масиву використовується функція insert(i, v), де v – значення, що вставляє це, i – ітератор, що вказує на елемент, перед яким потрібно вставити значення.

Завдання до лабораторної роботи

Можна вибрати (за дозволом викладача) інший тип контейнера.

На 9 балів: замість даного завдання виконати завдання №1 ЛР "Обробка одновимірних масивів" використовуючи контейнер ВЕКТОР.

Варіант 1

Скласти програму, що містить динамічну інформацію про наявність автобусів в автобусному парку.

Відомості про кожен автобус включають:

номер автобуса;

прізвище й ініціали водія;

номер маршруту.

Програма повинна забезпечувати:

початкове формування даних про всі автобуси в парку у вигляді списку;

при виїзді кожного автобуса з парку вводиться номер автобуса, і програма видаляє дані про цей автобус зі списку автобусів, що перебувають у парку, і записує ці дані в список автобусів, що перебувають на маршруті;

при в'їзді кожного автобуса в парк уводиться номер автобуса, і програма видаляє дані про цей автобус зі списку автобусів, що перебувають на маршруті, і записує ці дані в список автобусів, що перебувають у парку;

за запитом видаються відомості про автобуси, що перебувають у парку, або про автобуси, що перебувають на маршруті.

Варіант 2

Скласти програму, що містить поточну інформацію про книги в бібліотеці.

Відомості про книги включають:

номер УДК;

прізвище й ініціали автора;

назву;

рік видання;

кількість екземплярів даної книги в бібліотеці.

Програма повинна забезпечувати:

початкове формування даних про всі книги в бібліотеці у вигляді вектора;

додавання даних про книги, що знову надходять у бібліотеку;

видалення даних про книги, що списують;

за запитом видаються відомості про наявність книг у бібліотеці, упорядковані за роками видання.

Варіант 3

Скласти програму, що містить поточну інформацію про заявки на авіаквитки.

Кожна заявка включає:

пункт призначення;

номер рейсу;

прізвище й ініціали пасажира;

бажану дату вильоту.

Програма повинна забезпечувати:

зберігання всіх заявок у вигляді списку;

додавання заявок у список;

видалення заявок;

виведення заявок за заданим номером рейсу й датою вильоту;

виведення всіх заявок.

Варіант 4

Скласти програму, що містить поточну інформацію про заявки на авіаквитки.

Кожна заявка включає:

пункт призначення;

номер рейсу;

прізвище й ініціали пасажира;

бажану дату вильоту.

Програма повинна забезпечувати:

зберігання всіх заявок у вигляді вектора;

додавання й видалення заявок;

за заданим номером рейсу й датою вильоту виведення заявок з їхнім наступним видаленням;

виведення всіх заявок.

Варіант 5

Скласти програму, що містить поточну інформацію про книги в бібліотеці.

Відомості про книги включають:

номер УДК;

прізвище й ініціали автора;

назву;

рік видання;

кількість екземплярів даної книги в бібліотеці.

Програма повинна забезпечувати:

початкове формування даних про всі книги в бібліотеці у вигляді списку;

при видачі кожної книги на руки вводиться номер УДК, і програма зменшує значення кількості книг на одиницю або видає повідомлення про те, що необхідної книги в бібліотеці немає або необхідна книга перебуває на руках;

при поверненні кожної книги вводиться номер УДК і програма збільшує значення кількості книг на одиницю;

за запитом видаються відомості про наявність книг у бібліотеці.

Варіант 6

Скласти програму, що містить динамічну інформацію про наявність автобусів в автобусному парку.

Відомості про кожен автобус включають:

номер автобуса;

прізвище й ініціали водія;

номер маршруту;

ознаку того, де перебуває автобус – на маршруті або в парку.

Програма повинна забезпечувати:

початкове формування даних про всі автобуси у вигляді вектора;

при виїзді кожного автобуса з парку вводиться номер автобуса і програма встановлює значення ознаки "автобус на маршруті";

при в'їзді кожного автобуса в парк уводиться номер автобуса і програма встановлює значення ознаки "автобус у парку";

за запитом видаються відомості про автобуси, що перебувають у парку, або про автобуси, що перебувають на маршруті.

Варіант 7

У файловій системі каталог файлів організований у вигляді лінійного списку. Для кожного файла в каталозі містяться наступні відомості:

ім'я файла;

дата створення;

кількість звертань до файла.

Написати програму, що забезпечує:

початкове формування каталогу файлів;

виведення каталогу файлів;

видалення файлів, дата створення яких менше заданої;

вибірку файла з найбільшою кількістю обігів.

Програма повинна забезпечувати діалог за допомогою меню й контроль помилок при введенні.

Варіант 8

Предметний покажчик організований у вигляді вектора. Кожен компонент покажчика містить слово й номери сторінок, на яких це слово зустрічається. Кількість номерів сторінок, що відносяться до одного слова, лежить у діапазоні від одного до десяти.

Написати програму, що забезпечує:

початкове формування предметного покажчика;

виведення предметного покажчика;

виведення номерів сторінок для заданого слова.

Програма повинна забезпечувати діалог за допомогою меню й контроль помилок при введенні.

Варіант 9

Текст допомоги для деякої програми організований у вигляді списку.

Кожен компонент тексту допомоги містить термін (слово) і текст, що містить пояснення до цього терміна. Кількість рядків тексту, що відносяться до одного терміна, становить від однієї до п'яти.

Написати програму, що забезпечує:

початкове формування тексту допомоги;

виведення тексту допомоги;

виведення тексту, що пояснює, для заданого терміна.

Програма повинна забезпечувати діалог за допомогою меню й контроль помилок при введенні.

Варіант 10

Картотека в бюро обміну квартир організована у вигляді вектора. Відомості про кожну квартиру включають:

кількість кімнат;

поверх;

площа;

адреса.

Написати програму, що забезпечує:

початкове формування картотеки;

уведення заявки на обмін;

пошук у картотеці підходящого варіанта: при рівності кількості кімнат поверху й розходженні площ у межах 10% відповідна картка виводиться й видаляється зі списку, у протилежному разі заявка, що надійшла, включається у вектор;

виведення усього вектора.

Програма повинна забезпечувати діалог за допомогою меню й контроль помилок при введенні.

Варіант 11

Англо-російський словник побудований у вигляді списку (MAP).

Кожен компонент містить англійське слово, російське слово, що відповідає йому, й лічильник кількості звертань до даного компонента.

Спочатку список формується в порядку англійського алфавіту. У процесі експлуатації словника при кожному звертанні до компонента до лічильника обігів додається одиниця.

Написати програму, що:

забезпечує початкове введення словника з конкретними значеннями лічильників обігів;

формує нове подання словника у вигляді списку за наступним алгоритом: а) у старому словнику шукається компонент із найбільшим значенням лічильника обігів; б) знайдений компонент заноситься в новий словник і видаляється зі старого; в) перехід до п.а) до вичерпання вихідного словника;

робить виведення вихідного й нового словників.

Програма повинна забезпечувати діалог за допомогою меню й контроль помилок при введенні.

Варіант 12

Анкета для опитування населення містить дві групи питань. Перша група містить відомості про респондента:

вік;

стать;

освіта (початкова, середня, вища).

Друга група містить власне запитання анкети, відповіддю на які може бути або ТАК, або НІ.

Написати програму, що:

забезпечує початкове ведення анкет і формує з них вектор;

на основі аналізу анкет видає відповіді на наступні запитання: а) скільки чоловіків старше 40 років, що мають вищу освіту, відповіли ТАК на запитання анкети; б) скільки жінок молодше 30 років, що мають середню освіту, відповіли НІ на запитання анкети; в) скільки чоловіків молодше 25 років, що мають початкову освіту, відповіли ТАК на запитання анкети;

робить виведення всіх анкет і відповідей на запитання.

Програма повинна забезпечувати діалог за допомогою меню й контроль помилок при введенні.

Варіант 13

Написати програму, що містить поточну інформацію про книги в бібліотеці.

Відомості про книги включають:

номер УДК;

прізвище й ініціали автора;

назву;

рік видання;

кількість екземплярів даної книги в бібліотеці.

Програма повинна забезпечувати:

початкове формування даних про всі книги в бібліотеці у вигляді списку;

додавання даних про книги, що знову надходять у бібліотеку;

видалення даних про книги, що списують;

за запитом видаються відомості про наявність книг у бібліотеці, упорядковані за роками видання.

Варіант 14

На міжміській телефонній станції картотека абонентів, що містить відомості про телефони і їхніх власників, організована у вигляді вектора.

Написати програму, що:

забезпечує початкове формування картотеки у вигляді вектора;

робить виведення всієї картотеки;

вводить номер телефону й час розмови;

виводить повідомлення на оплату телефонної розмови.

Програма повинна забезпечувати діалог за допомогою меню й контроль помилок при введенні.

Варіант 15

На міжміській телефонній станції картотека абонентів, що містить відомості про телефони і їхніх власників, організована у вигляді списку.

Написати програму, що:

забезпечує початкове формування картотеки у вигляді списку;

робить виведення всієї картотеки;

уводить номер телефону й час розмови;

виводить повідомлення на оплату телефонної розмови.

Програма повинна забезпечувати діалог за допомогою меню й контроль помилок при введенні.

Варіант 16

Автоматизована інформаційна система на залізничному вокзалі містить відомості про відправлення поїздів далекого прямування.

Для кожного поїзда вказується:

номер поїзда;

станція призначення;

час відправлення.

Дані в інформаційній системі організовані у вигляді вектора. Написати програму, що:

забезпечує введення даних в інформаційну систему й формування вектора;

робить виведення елементів вектора;

уводить номер поїзда й виводить всі дані про цей поїзд;

уводить назву станції призначення й виводить дані про всі поїзди, що відправляються до цієї станції.

Програма повинна забезпечувати діалог за допомогою меню й контроль помилок при введенні.

Варіант 17

Автоматизована інформаційна система на залізничному вокзалі містить відомості про відправлення поїздів далекого прямування.

Для кожного поїзда вказується:

номер поїзда;

станція призначення;

час відправлення.

Дані в інформаційній системі організовані у вигляді списку. Написати програму, що:

забезпечує введення даних в інформаційну систему й формування списку;

робить виведення усього списку;

уводить номер поїзда й виводить всі дані про цей поїзд;

уводить назву станції призначення й виводить дані про всі поїзди, що відправляються до цієї станції.

Програма повинна забезпечувати діалог за допомогою меню й контроль помилок при введенні.

Варіант 18

Скласти програму, яка містить поточну інформацію про заявки на закупівлю будматеріалів.

Кожна заявка включає:

пункт призначення;

номер рейсу;

прізвище й ініціали покупця;

бажану дату завезення.

Програма повинна забезпечувати:

зберігання всіх заявок у вигляді списку;

додавання заявок у список;

видалення заявок;

виведення заявок за заданим номером рейсу і датою завезення;

виведення всіх заявок.

Варіант 19

Скласти програму, яка містить поточну інформацію про канцелярські товари на складі.

Відомості про канцелярські товари включають:

номер за порядком;

найменування;

рік випуску;

кількість.

Програма повинна забезпечувати:

початкове формування даних про канцтовари у вигляді вектора;

додавання даних про канцтовари, що знов поступають на склад;

видалення даних про списувані канцтовари;

за запитом видаються відомості про наявність про канцелярських товарів на складі, упорядковані за роками випуску.

Варіант 20

Скласти програму, яка містить поточну інформацію про заявки на туристичні тури.

Кожна заявка включає:

пункт призначення;

номер туру;

прізвище й ініціали замовника;

бажану дату вильоту.

Програма повинна забезпечувати:

зберігання всіх заявок у вигляді списку;

додавання заявок у список;

видалення заявок;

виведення заявок за заданим номером туру і датою вильоту;

виведення всіх заявок.

Контрольні запитання

1. Що таке контейнер?
2. Які стандартні види контейнерів ви знаєте?
3. Опишіть ситуації, коли і які контейнери слід використовувати.
4. Охарактеризуйте основні алгоритми для контейнерів.
5. Що дозволяють виконувати ітератори стосовно контейнерів.
6. Які ітератори ви знаєте?
7. Розкрийте поняття "функціональний об'єкт".
8. Розкрийте поняття "алокатор".
9. Розкрийте поняття "прямий ітератор".
10. Що таке предикат?

Лабораторна робота №6
Дослідження структури Windows-додатка

Мета лабораторної роботи – одержання практичних навичок у побудові базового додатка для Win32, дослідження параметрів віконних процедур.

Перед виконанням лабораторної роботи студент повинен знати: принципи й логіку роботи Windows-додатка; порядок і засоби формування структури класу вікна, реєстрації класу, написання віконної процедури, створення й відображення вікна, циклу обробки повідомлень.

Після виконання лабораторної роботи студент повинен уміти: розробляти базові процедурні Windows-додатки мовою C++.

Короткі теоретичні відомості

Розглянемо ситуацію, коли користувач додатка натискає клавішу, а система виробляє повідомлення про цю подію. Windows забезпечує підтримку клавіатури, що не залежить від типу пристрою (device-independent support). Для кожного типу клавіатури вона встановлює відповідний драйвер, тобто спеціальну програму, що служить посередником між клавіатурою й операційною системою. Клавіатурна підтримка Windows не залежить від мови спілкування із системою. Це досягається використанням спеціальної клавіатурної розкладки (layout), що користувач вибрав у цей момент. Кожній клавіші на рівні апаратури привласнене унікальне значення – ідентифікатор клавіші, що залежить від типу пристрою й називається скан-кодом.

Клавіатурний драйвер інтерпретує скан-код і перетворить його в обумовлений Windows-код віртуальної клавіші (virtual-key code), що не залежить від типу пристрою й ідентифікуючий функціональний зміст клавіші. Після цього перетворення скан-коду драйвер створює повідомлення, у яке включає: скан-код, віртуальний код й іншу супутню інформацію. Потім він поміщає повідомлення в спеціальну чергу системних повідомлень. Windows вибирає повідомлення із цієї черги й посилає в чергу повідомлень відповідного потоку (thread). Зрештою, цикл вибірки повідомлень даного потоку передає його відповідній віконній процедурі для обробки. Модель уведення із клавіатури в системі Windows подана на рис. 6.1.

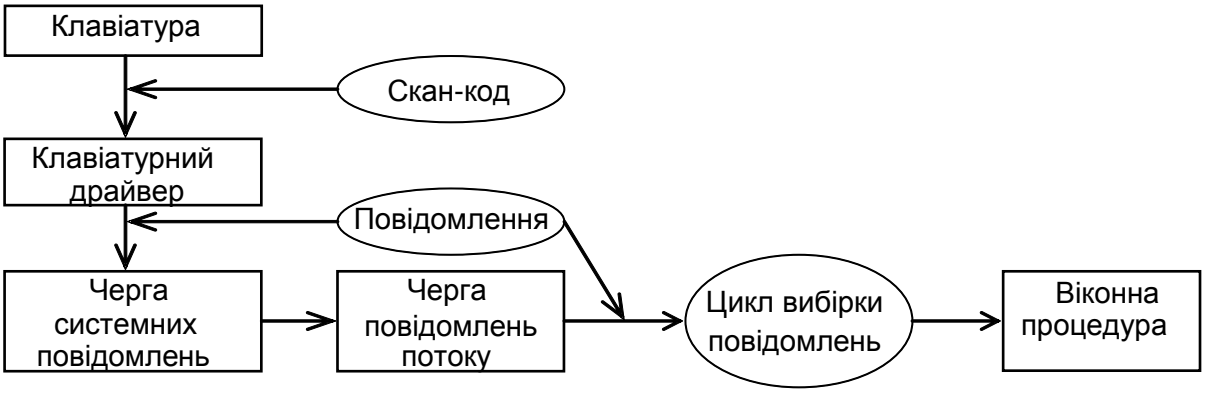

Рис. 6.1. **Модель уведення із клавіатури в системі Windows**

Тут буфер клавіатури служить сполучною ланкою між прикладною програмою й одним із сервисів ОС. Так само формують (або можуть формуватися) свої специфічні дані оброблювачі інших подій. При цьому використовується універсальна структура даних MSG (повідомлення), що описує будь-яку подію. Вона містить супровідну інформацію, достатню для того, щоб повідомленням можна було скористатися. Наприклад, для повідомлення від клавіатури це повинен бути код натиснутої клавіші, для повідомлення від миші – координати її покажчика, для повідомлення WM_SIZE – розміри вікна.

Кожен оброблювач події (драйвер) поміщає сформоване повідомлення в певну динамічну структуру даних у пам'яті. Інші апаратні й програмні оброблювачі точно так само формують свої повідомлення, ставлячи їх у чергу за вже існуючими. Так формується системна черга повідомлень.

Розглянута модель вироблення й проходження повідомлень допомагає зрозуміти структуру, прийняту для всіх Windows-додатків. Останні два блоки в розглянутій схемі (рис. 6.1) визначають особливості будови будь-якого Windows-додатка. Найпростіше з них повинне складатися як мінімум із двох функцій:

- функції WinMain, з якої починається виконання програми і яка "закручує" цикл очікування повідомлень (message pump);
- віконної процедури, що викликає система, направляючи їй відповідні повідомлення.

Кожен додаток у системі, заснованій на повідомленнях, повинен вміти одержувати й обробляти повідомлення зі своєї черги. Основу

такого додатка в системі Windows становить функція WinMain, що містить стандартну послідовність дій. Однак обробляється більшість повідомлень вікном – об'єктом операційної системи Windows.

З погляду користувача, вікно – це прямокутна область екрана, що відповідає якомусь додатку або його частині. Додаток може управляти декількома вікнами, серед яких звичайно виділяють одне головне вікно (Frame Window).

З погляду операційної системи, вікно – це в більшості випадків кінцевий пункт, у якій направляються повідомлення.

З погляду програміста, вікно – це об'єкт, атрибути якого (тип, розмір, положення на екрані, вид курсору, меню, значок, заголовок) повинні бути спочатку сформовані, а потім зареєстровані системою.

Маніпуляція вікном здійснюється за допомогою спеціальної віконної функції, що має цілком певну, устояну структуру.

Функція WinMain() виконується першої в будь-якому додатку. Її ім'я зарезервоване операційною системою. Ім'я віконної процедури довільно й вибирається розроблювачами. Система Windows реєструє це ім'я, пов'язуючи його з додатком.

Таким чином, головною метою функції WinMain() є:

- реєстрація віконного класу;
- створення вікна;
- запуск циклу очікування повідомлень.

Далі розглянемо більш докладно структуру традиційного Windows-додатка, що повинен бути взятий за основу при рішенні свого варіанта завдання.

```
// Визначає точку входу додатка
#include <windows.h>
//=== Глобальні змінні:
HINSTANCE hInst;                    //Дескриптор додатка
LPCTSTR lpszAppName = "MyApp"; //Покажчик на рядок, що
//містить ім'я вікна
LPCTSTR lpszTitle    = "My Programm";        //Текст заголовка вікна
LPCTSTR lpszClassName = "Hello";
//=======Прототипи функцій, що входять у даний модуль
int APIENTRY WinMain( HINSTANCE hInstance,
HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)
```

```
{
  MSG      msg;
  HWND  hWnd;
// Реєстрація віконного класу
WNDCLASSEX wc;
  wc.style  = CS_HREDRAW | CS_VREDRAW; //стиль вікна
  wc.lpfnWndProc   = (WNDPROC)WndProc;       //віконна процедура
  wc.cbClsExtra  = 0;
  wc.cbWndExtra  = 0;
  wc.hInstance = hInstance;              //опис додатка
  wc.hIcon = LoadIcon( hInstance, lpszAppName );//визначення іконки
  wc.hCursor = LoadCursor(NULL, IDC_ARROW); //визначення
                                                    //курсору
  wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);//установка
                                                        //фону
  wc.lpszMenuName  = lpszAppName;             //визначення меню
  wc.lpszClassName = lpszClassName;             //ім’я класу вікна
  wc.cbSize = sizeof(WNDCLASSEX); //розмір структури в байтах
  wc.hIconSm = NULL; //дескриптор піктограми
if ( !RegisterClassEx( &wc ) )
    return( FALSE );
// Запам'ятовування дескриптора (хендла) додатка
  hInst = hInstance;
//======= Створення головного вікна
hWnd = CreateWindow
(lpszClassName, //ім'я класу вікна
lpszTitle, //ім'я додатка
WS_OVERLAPPEDWINDOW, //стилі вікна
CW_USEDEFAULT,  //положення по горизонталі верхнього лівого
//кута
CW_USEDEFAULT, //положення по вертикалі верхнього лівого кута
CW_USEDEFAULT,  //ширина вікна
CW_USEDEFAULT,  //висота вікна
NULL,           //дескриптор батьківського вікна
NULL,           //дескриптор меню вікна
hInstance,      //дескриптор додатка
```

```
        NULL                //покажчик на додаткові параметри
        );
        if ( !hWnd )
            return( FALSE );
          //======= Показ вікна
          ShowWindow( hWnd, nCmdShow );
         //======= Відновлення вікна
          UpdateWindow( hWnd );
        //===== Цикл очікування й обробки повідомлень
          while( GetMessage( &msg, NULL, 0, 0) )
          {
            TranslateMessage( &msg );
            DispatchMessage( &msg );
          }
          return( msg.wParam );
        }
        //  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
        //  ПРИЗНАЧЕННЯ:  Обробка повідомлень головного вікна.
        //  WM_CREATE – повідомлення ініціалізації при створенні вікна
        //  WM_COMMAND – обробка команд меню
        //  WM_DESTROY – посилка повідомлення про завершення й
//вихід
        LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM
wParam, LPARAM lParam )
        {
        switch( uMsg )
          {
            case WM_CREATE: //повідомлення ініціалізації
        break;
            case WM_COMMAND : //повідомлення від меню, гарячих клавіш
//і т. д.
                    //====== Розшифровка вибору в меню:
        switch( LOWORD( wParam ) )
                    {
                      case IDM_TEST :
                            {
                     //====== Сюди міститься код реакції на вибір пункту меню
```

```
                    }
                    break;
                case IDM_EXIT :
                    DestroyWindow( hWnd );
                    break;
            }
            break;
        case WM_DESTROY :      // Повідомлення закриття вікна
            PostQuitMessage(0);
            break;
    //обробка повідомлень, які не оброблені користувачем
        default :
            return( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
    }
    return (FALSE);
}
```

Особливість першої лабораторної роботи полягає у варіюванні параметрами функцій реєстрації й створення вікна, віконної процедури для рішення індивідуального завдання.

До таких параметрів відносяться:

1. Вид створюваного вікна на екрані монітора.
2. Стиль класу вікна.
3. Системні піктограми (іконки).
4. Системні курсори миші.
5. Стиль вікна.

Розглянемо їх докладніше.

1. Вид створюваного вікна описується в останньому параметрі функції WinMain(), а також у другому параметрі функції показу вікна ShowWindow(). Можливі значення задаються, виходячи з табл. 6.1.

Таблиця 6.1

Можливі значення другого параметра функції ShowWindow()

Параметр	Значення	Опис
1	2	3
SW_HIDE	0	Вікно сховане
SW_SHOWNORMAL	1	Вікно показане в його нормальних розмірах
SW_NORMAL	1	

Закінчення табл. 6.1

1	2	3
SW_SHOWMINIMIZED	2	Вікно згорнуте й показане як піктограма
SW_SHOWMAXIMIZED	3	Вікно розгорнуте
SW_MAXIMIZE	3	
SW_SHOWNOACTIVE	4	Вікно відображається в його розмірах і позиції, установлених безпосередньо перед поточними значеннями розмірів і позиції. Активне вікно залишається активним
SW_SHOW	5	Вікно відображається в його поточних розмірах і позиції
SW_MINIMIZE	6	Вікно згорнуте й активізує вікно верхнього рівня в списку системи
SW_SHOWMINNOACTIVE	7	Вікно згорнуте. Активне вікно залишається активним
SW_SHOWNA	8	Вікно показане в його поточному стані. Активне вікно залишається активним
SW_RESTORE	9	Активізувати й відобразити вікно. Якщо вікно згорнуте або розгорнуте, йому будуть повернуті його первинні розміри й позиція
SW_SHOWDEFAULT	10	Застосовується при запуску додатка за замовчуванням

2. Найменування стилів класу вікна починається з ідентифікаторів CS_. Для стилю вікна відведено 16 бітів і тільки один із цих бітів установлений в одиницю. Таким чином, стилі використовуються як бітові прапори, тобто із цими стилями можна робити операції логічного додавання й логічного множення для одержання комбінованих стилів. Перелік прапорів наведений у табл. 6.2.

Таблиця 6.2

Перелік бітових прапорів стилю класу вікна

Прапор	Опис
1	2
CS _VREDRAW	Перемалювати вікно при зміні висоти вікна
CS_HREDRAW	Перемалювати вікно при зміні ширини вікна
CS_KEYCVTWINDOW	
CS_DBLCLKS	Послати повідомлення віконної функції при подвійному щиглику мишею, якщо курсор перебуває в межах вікна

Закінчення табл. 6.2

1	2
CS_OWNDC	Для кожного вікна класу виділяється власний контекст
CS_CLASSDC	Той самий контекст пристрою розділяється всіма вікнами цього класу
CS_PARENTDC	Дочірні вікна успадковують контекст батьківського вікна
CS_NOKEYCVT	
CS_NOCLOSE	Забрати команду "Close" із системного меню
CS_SAVEBITS	Зберігати частину області екрана, закриту вікном, як bitmip, при видаленні відновлювати перекриту область
CS_BYTEALIGNCLIENT	Вирівнює межу робочої області вікна (у горизонтальному напрямку) таким чином, щоб для відображення рядка було потрібно ціле число байтів
CS_BYTEALIGNWINDOW	Теж, але дія охоплює все вікно
CS_GLOBALCLASS	Дозволяється створювати клас, що не залежить від поточного hInstance

3. Програміст може використати власну піктограму, що він сам розробив, а може застосувати одну з визначених (табл. 6.3). У випадку використання власної іконки перший параметр функції LoadIcon() повинен бути рівним хендлу програми (hInstance). При використанні визначеної іконки перший параметр функції дорівнює нулю.

Таблиця 6.3

Список визначених системних піктограм

Іконка	Опис
IDI_APPLICATION	Піктограма додатка, задана за замовчуванням
IDI_ASTERISK	Зірочка (використовується в інформаційних повідомленнях)
IDI_ERROR	Піктограма попередження про виниклу помилку
IDI_EXCLAMATION	Знак оклику (використовується в попереджуючих повідомленнях)
IDI_HAND	Піктограма у формі руки (для організації попереджуючих повідомлень)
IDI_QUESTION	Знак питання (використовується в підказках)
IDI_WINLOGO	Емблема Windows

4. Сказане про іконку можна повністю віднести й до курсору миші, яким будуть користуватися вікна створюваного класу (не плутати з курсором, застосовуваним при редагуванні текстових файлів). Поле WndClassex.hCursor визначає хендл курсору. Всі ідентифікатори визначених курсорів починаються з IDC (табл. 6.4).

Таблиця 6.4

Список визначених ідентифікаторів курсору

Курсор	Опис
IDC_APPSTARTING	Стандартна стрілка й пісочний годинник
IDC_ARROW	Стандартна стрілка
IDC_CROSS	Перехрестя
IDC_HELP	Стрілка й знак питання
IDC_ICON	Порожня піктограма
IDC_IBEAM	Текстовий дуавр
IDC_NO	Перекреслений кружок
IDC_SIZE	Чотирекінцева стрілка
IDC_SIZEALL	Те ж, що й IDC_SIZE
IDC_SIZENS	Двокінцева стрілка, що вказує на північ і південь
IDC_SIZENWSE	Двокінцева стрілка, що вказує на північний захід і південний схід
IDC_SIZENESW	Двокінцева стрілка, що вказує на північний схід і південний захід
IDC_SIZEWE	Двокінцева стрілка, що вказує на захід і схід
IDC_UPARROW	Вертикальна стрілка
IDC_WAIT	Пісочний годинник

5. Параметр стилів вікна визначає індивідуальні характеристики конкретного вікна. Стиль визначає, чи буде вікно мати заголовок, іконку системного меню, кнопки мінімізації, максимізації, характер межі вікна, визначає також взаємини вікон типу предок-нащадок і т. д. Під це поле приділяється 32 біти. Визначено кілька десятків стилів вікон. Їхні ідентифікатори починаються з букв WS_. Як й у випадку зі стилями класу, ці значення використовуються як бітові прапори, тобто комбінуючи їх за допомогою логічних операцій, можна одержати той стиль вікна, що потрібно. Перелік бітових прапорів стилю вікна наведений в табл. 6.5.

Таблиця 6.5

Перелік бітових прапорів стилю вікна

Прапор	Опис
1	2
WS_BORDER	У вікна є тонка обмежуюча рамка
WS_CAPTION	WS_BORDER | WS_DLGFRAME
WS_CHILD	Створюється дочірнє вікно, що має за замовчуванням тільки робочу область, меню вікна цього стилю не мають ніколи
WS_CHILDWINDOW	Те ж, що й WS_CHILD

Закінчення табл. 6.5

1	2
WS_CLIPCHILDREN	При промальовуванні батьківського вікна область, займана дочірніми вікнами, не прорисовується
WS_CLIPSIBLINGS	Дочірнє вікно, що має цей стиль і перекриває інше дочірнє вікно, при промальовуванні перекриває області, що не прорисовуються
WS_DISABLED	Створюється вікно, у якому спочатку заборонене одержання даних, уведених користувачем
WS_DLGFRAME	У вікна є рамка, що звичайно буває у діалогових вікон
WS_GROUP	Вікно є першим вікном групи
WS_HSCROLL	У вікна є горизонтальна лінійка прокручування
WS_ICONIC	Те ж, що й WS_MINIMIZE
WS_MAXIMIZE	Створюється споконвічно максимізоване вікно
WS_MAXIMIZEBOX	У вікна є кнопка максимізації
WS_MINIMIZE	Створюється споконвічно мінімізоване вікно
WS_MINIMIZEBOX	У вікна є кнопка мінімізації
WS_OVERLAPPED	Вікно має заголовок і рамку, що обрамляє
WS_OVERLAPPEDWINDOW	WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_THICKFRAME | WS_MINIMIZEBOX | WS_MAXIMIZEBOX
WS_POPUP	Створюється спливаюче вікно
WS_POPUPWINDOW	WS_SYSMENU | WS_BORDER | WS_POPUP
WS_SIZEBOX	Те ж, що й WS_THICKFRAME
WS_SYSMENU	У вікна є системне меню
WS_TABSTOP	Вікно може одержувати клавіатурний фокус при натисканні користувачем клавіші Tab
WS_THICKFRAME	У вікна є досить товста рамка, що дозволяє йому змінювати розміри. Заголовка у вікна немає
WS_TILED	Те ж, що й WS_OVERLAPPED
WS_TILED_WINDOW	Те ж, що й WS_OVERLAPPEDWINDOW
WS_VISIBLE	Створюється споконвічно відображуване вікно
WS_VSCROLL	У вікна є вертикальна лінійка прокручування

Варто звернути увагу, що далі такий докладний розгляд програм буде відсутній. Це не стосується необхідних функцій і параметрів для виконання завдань. При необхідності варто використати довідкові системи або літературу, наведену наприкінці методичних рекомендацій. Крім того, кожне завдання до лабораторної роботи включає дві частини. Перша частина – загальна й виконується всіма студентами, а друга – індивідуальна, відповідно до призначеного викладачем номера варіанта. У плині всього циклу лабораторних робіт зміна номера індивідуального варіанта не допускається.

Створення процедурно-орієнтованого проекту

Для того щоб створити процедурно-орієнтований додаток Windows за допомогою майстра Visual C++ Application Wizard, необхідно відкрити діалогове вікно New Project (рис. 6.2).

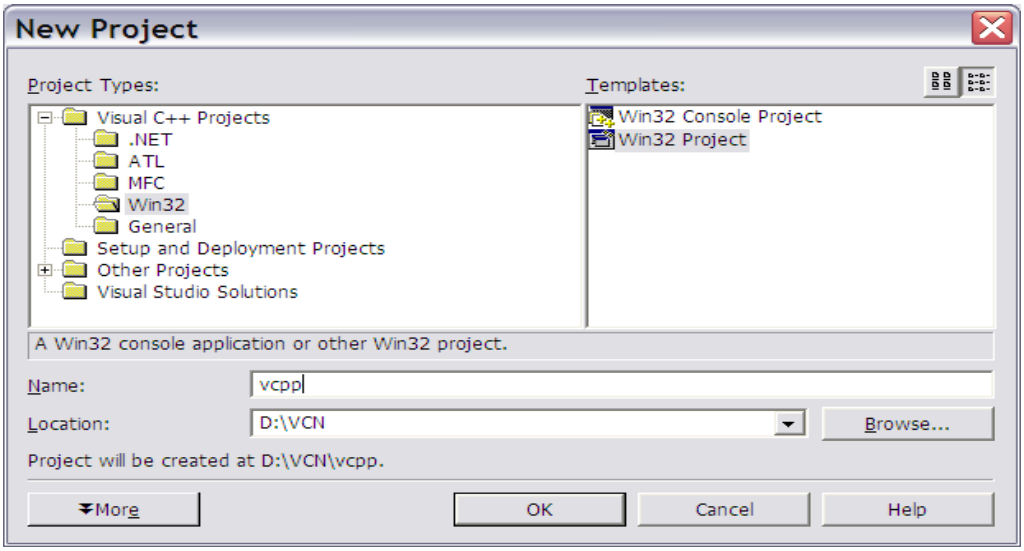

Рис. 6.2. **Діалогове вікно New Project**

Це діалогове вікно можна відкрити, вибравши пункт меню File > New. У вас з'явиться вибір: створити проект (Project) або файл (File). Виберіть опцію Project.

У вікні New Project, що відкрилося, виберіть шаблон Win32 Project, уведіть ім'я нового проекту (vcpp) і задайте місце розташування файлів проекту (D:\VCN). Натисніть кнопку OK, у результаті чого на екрані з'явиться діалогове вікно Win32 Application Wizard (рис. 6.3).

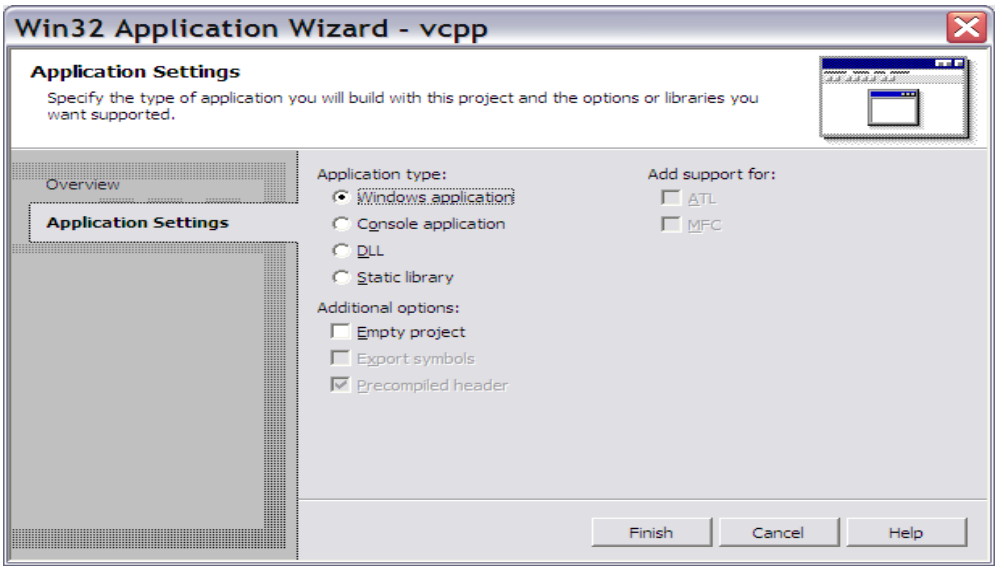

Рис. 6.3. **Діалогове вікно Application Wizard**

При роботі із процедурно-орієнтованим додатком на вкладці Application Settings повинна бути обрана кнопка Windows Application. Натисніть кнопку Finish, і майстер додатків автоматично створить деяке число файлів, у тому числі файлів з базовим вихідним кодом. Вікно проектування Visual C++ з відкритим файлом вихідного коду для проекту vcpp показане на рис. 6.4.

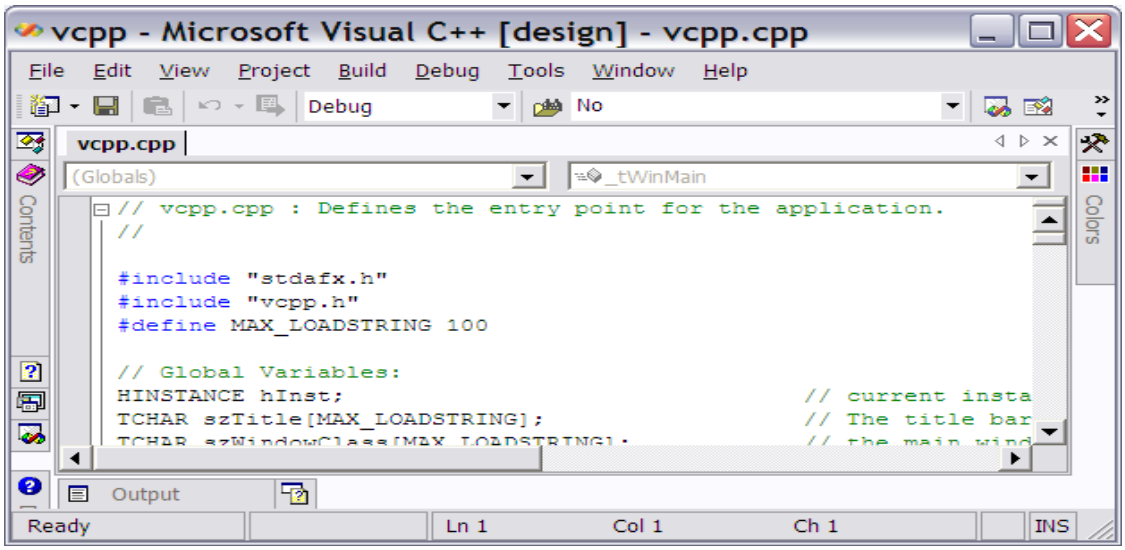

Рис. 6.4. **Файл vcpp.cpp вихідного коду**

На даному етапі можна було б скомпілювати й відредагувати зв'язку проекту, а потім виконати такий. Однак, запустивши свою програму, ви побачите лише вікно (рис. 6.5).

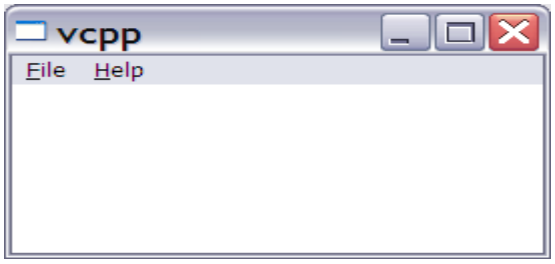

Рис. 6.5. **Вид стандартного вікна нового проекту**

Завдання до лабораторної роботи

Написати програму, яка використає функції API, що створює спливаюче вікно Windows. Вікно повинне містити заголовок з відображенням прізвища студента (російською мовою), а також інформацію про номер навчальної групи. Параметри вікна повинні відповідати завданню.

Варіант 1

Вікно повинне:

бути спочатку схованим;

мати вирівняну межу клієнтської області по горизонталі, щоб для відображення рядка було потрібно ціле число байтів;

мати задану за замовчуванням іконку;

мати курсор у вигляді двокінцевої стрілки з напрямками на північ і південь;

мати товсту рамку, заголовок із кнопками максимізації й мінімізації;

мати коричневі кольори тла;

розташовуватися в довільному місці екрана.

Варіант 2

Вікно повинне:

бути показаним у поточному стані;

мати відсутню команду "Close" у системному меню;

мати завантажену іконку у вигляді діаграми;

мати курсор у формі пісочного годинника;

мати тонку рамку, заголовок із кнопкою мінімізації й системним меню;

мати маслинові кольори тла;

розташовуватися в центрі екрана, займаючи його половину по горизонталі й чверть по вертикалі.

Варіант 3

Вікно повинне:

мати відображатися в розмірах і позиції, установлених безносередньо перед поточними значеннями;

перемальовуватися при зміні висоти вікна;

мати іконку у вигляді знака питання;

мати курсор у вигляді перехрестя;

мати тонку рамку, заголовок із кнопкою максимізації й системного меню;

мати темно-зелені кольори тла;

розташовуватися в координатах (40,80) і займати 180 точок по ширині й 220 – по висоті.

Варіант 4

Вікно повинне:

відображатися при запуску додатка за замовчуванням;

перемальовуватися при зміні ширини вікна;

мати іконку у вигляді логотипу Windows;

мати курсор у формі чотирьохкінцевої стрілки;

мати тільки горизонтальні смуги прокручування й спочатку заборонений стан;

мати темно-сизі кольори тла;

розташовуватися в правому нижньому куті екрана.

Варіант 5

Вікно повинне:

бути згорнуте й показано як піктограма;

бути частиною класу, що не залежить від поточного додатка;

мати іконку у вигляді повідомлення про збій у програмі;

мати курсор у вигляді стандартної стрілки й пісочного годинника;

мати горизонтальну й вертикальні смуги прокручування, без заголовка;

мати темно-сині кольори тла;

розташовуватися в правому нижньому куті.

Варіант 6

Вікно повинне:

відображатися в поточних розмірах і позиції;

посилати повідомлення віконній процедурі при подвійному щиглику миші, якщо курсор перебуває в межах вікна;

мати іконку у вигляді зірочки;

мати курсор у вигляді перекресленого кружка;

мати тонку рамку й одержати клавіатурний фокус при натисканні користувачем клавіші Tab;

мати кольори тла – індиго;

розташовуватися по верхньому лівому краї вікна й мати висоту в 150 пікселей, а ширину в 333 пікселя.

Варіант 7

Вікно повинне:

бути згорнутим й активізуватися на верхньому рівні в списку системи;

бути дочірнім зі спадкуванням контексту батьківського вікна;

завантажувати іконку у вигляді довільного головного убору;

мати курсор у вигляді стрілки зі знаком питання;

мати рамку, що обрамляє, аналогічну для діалогових вікон Windows;

мати темно-червоні кольори тла;

розташовуватися посередині ліворуч і мати висоту в 166 пікселей, а ширину в половину екрана.

Варіант 8

Вікно повинне:

активізуватися й відображатися в заданих розмірах і позиції. Якщо вікно згорнуте або розгорнуте, йому будуть повернуті його первинні розміри й позиція;

зберігати частину області екрана, закриту вікном, як bitmip, при видаленні відновлювати перекриту область;

мати іконку у вигляді знака оклику;

мати курсор у вигляді текстового дуавра;

бути дочірнім і мати тільки робочу область;

мати жовтогарячі кольори тла;

розташовуватися посередині праворуч і мати висоту у чверть екрана, а ширину в 350 пікселей.

Варіант 9

Вікно повинне:

бути згорнутим;

мати вирівняну межу всього вікна по горизонталі, щоб для відображення рядка було потрібно ціле число байтів;

мати іконку у вигляді руки;

мати курсор у вигляді вертикальної стрілки;

мати заголовок і рамку, що обрамляє;

мати коричнево-зелені кольори тла;

розташовуватися по правому нижньому краї вікна й мати висоту й ширину у чверть екрана.

Варіант 10

Вікно повинне:

бути показаним у нормальних розмірах;

мати власний контекст;

мати завантажену іконку у вигляді сонця;

курсор у вигляді двокінцевої стрілки, що вказує на північний захід і південний схід;

мати кнопку максимізації й бути спочатку мінімізованим;

мати ясно-червоні кольори тла;

розташовуватися по правому верхньому краї вікна й мати висоту й ширину в половину екрана.

Варіант 11

Вікно повинне:

бути спочатку розгорнутим у весь екран;

перемальовуватися при зміні ширини вікна;

мати іконку у вигляді повідомлення про збій у програмі;

мати курсор у вигляді двокінцевої стрілки з напрямками на північний схід і південний захід;

мати товсту рамку, заголовок із системним меню, але без кнопок максимізації й мінімізації;

мати пурпурні кольори тла;

розташовуватися посередині й мати розміри 377, 73.

Варіант 12

Вікно повинне:

бути показаним у поточному стані;

перемальовуватися при зміні висоти вікна;

мати іконку у вигляді руки;

мати курсор у формі двокінцевої стрілки, кінці якої вказують на захід і схід;

бути дочірнім і мати тільки робочу область;

мати сірі кольори тла;

розташовуватися зверху над центром екрана, займаючи його половину по горизонталі й третину по вертикалі.

Варіант 13

Вікно повинне:

бути згорнутим й активізуватися на верхньому рівні в списку системи;

зберігати частину області екрана, закриту вікном, як bitmip, при видаленні відновлювати перекриту область;

мати іконку у вигляді зірочки;

мати курсор у формі чотирьохкінцевої стрілки;

мати тонку рамку й горизонтальну смугу прокручування;

мати ясно-жовті кольори тла;

розташовуватися в координатах (120, 210) і займати 323 точки по ширині й 423 – по висоті.

Варіант 14

Вікно повинне:

відображатися при запуску додатка за замовчуванням;

посилати повідомлення віконній процедурі при подвійному щиглику миші, якщо курсор перебуває в межах вікна;

завантажувати іконку у вигляді футбольного м'яча;

мати курсор у формі порожньої піктограми;

мати заголовок і рамку, що обрамляє;

мати тло кольору хакі;

розташовуватися в лівому нижньому куті екрана й мати розміри 300·100 пікселей.

Варіант 15

Вікно повинне:

бути згорнуте й показано як піктограма;

бути дочірнім зі спадкуванням контексту батьківського вікна;

мати іконку у вигляді знака питання;

мати курсор у вигляді стандартної стрілки й пісочного годинника;

бути з тонкою рамкою й одержати клавіатурний фокус при натисканні користувачем клавіші Tab;

мати руді кольори тла;

розташовуватися в правому верхньому куті й мати розміри 200·400.

Варіант 16

Вікно повинне:

відображатися в розмірах і позиції, установлених безпосередньо перед поточними значеннями;

бути частиною класу, що не залежить від поточного додатка;

мати задану за замовчуванням іконку;

мати курсор у вигляді перекресленого кружка;

мати тонку рамку, заголовок із кнопкою мінімізації й бути спочатку максимізованим;

мати зелені кольори тла;

розташовуватися по верхньому лівому краї вікна й мати висоту й ширину в одну третину встановленого дозволу на моніторі.

Варіант 17

Вікно повинне:

активізуватися й відображатися в заданих розмірах і позиції. Якщо вікно згорнуте або розгорнуте, йому будуть повернуті його первинні розміри й позиція;

мати вирівняну межу клієнтської області по горизонталі, щоб для відображення рядка було потрібно ціле число байтів;

мати іконку у вигляді повідомлення про збій у програмі;

мати курсор у вигляді перекресленого кружка;

мати товсту рамку, заголовок із кнопками максимізації й мінімізації;

мати синьо-зелені кольори тла;

розташовуватися посередине праворуч і мати висоту в 257 пікселей, а ширину у чверть екрана.

Варіант 18

Вікно повинне:

бути показаним у поточному стані;

мати вирівняну межу всього вікна по горизонталі, щоб для відображення рядка було потрібно ціле число байтів;

завантажувати іконку у вигляді довільного найменування валюти;

мати курсор у вигляді перехрестя;

бути дочірнім і мати системне меню;

мати сині кольори тла;

розташовуватися праворуч і мати висоту в половину екрана, а ширину в третину екрана.

Варіант 19

Створити два вікна з однаковими властивостями. Кожне вікно повинне:

відображатися в поточних розмірах і позиції;

мати відсутню команду "Close" у системному меню;

мати іконку у вигляді руки;

мати іконку у вигляді знака оклику;

одне з вікон зробити першим й активним вікном групи;

мати сизі кольори тла;

розташовуватися ліворуч (праворуч), займаючи всю висоту екрана й половину його ширини.

Варіант 20

Вікно повинне:

бути згорнутим;

мати власний контекст;

завантажувати іконку у вигляді молодого півмісяця;

мати курсор у вигляді двокінцевої стрілки з напрямком на північ;

мати тільки горизонтальні смуги прокручування й спочатку заборонений стан;

мати червоні кольори тла;

розташовуватися по правому нижньому краї вікна й мати висоту й ширину в третину екрана.

Контрольні запитання

1. Чим відрізняється текст додатка, що створює вікна, від тексту додатка, що не створює вікон?
2. Який алгоритм роботи головної функції додатка?
3. Які дії повинен виконати додаток для створення вікна?
4. У чому особливості при обробці подій, керованих повідомленнями?
5. Опис яких програмних об'єктів необхідно включити в текст додатка для реєстрації класу вікон?
6. Яке призначення формальних параметрів функції вікна?
7. Яке повідомлення одержує функція вікна на етапі створення вікна?
8. Що відбувається з тими повідомленнями Windows, які не обробляються у функції вікна?
9. У чому суть циклу обробки повідомлень?
10. Поясніть порядок виклику функції віконної процедури.

Лабораторна робота №7
Дослідження взаємодії додатка з користувачем

Мета лабораторної роботи – досліджувати можливості функцій Win32 API зі створення візуального інтерфейсу додатка, одержати практичні навички написання й настроювання програм, які містять у ресурсних файлах опис курсорів, бітових образів, меню, а також відображення інформації про поточний стан у рядку стану вікна додатка; досліджувати характеристики, порядок опису елементів керування діалогових вікон, написати й налагодити програму, що здійснює настроювання заданих параметрів за допомогою діалогових вікон.

Перед виконанням лабораторної роботи студент повинен знати: особливості й шляхи організації користувальницького меню додатка, підтримку гарячих клавіш, порядок опису ресурсів і способи приєднання їх до додатка, склад, класи елементів керування, які застосовуються в діалогових вікнах, їхні формати опису у файлах ресурсів, механізми обміну повідомленнями.

Після виконання лабораторної роботи студент повинен уміти розробляти Windows-додатки з використанням елементів взаємодії з користувачем.

Короткі теоретичні відомості

Етапи створення додатка

Процес розробки додатків Windows можна розбити на кілька основних етапів:

- створення функції WinMain() і інших базових функцій. При використанні MFC ці дії виконуються автоматично в класі CwinApp;
- розробка меню, діалогового вікна й інших необхідних ресурсів, включення їх у файл сценарію ресурсів;
- створення за допомогою редактора ресурсів унікальних покажчиків миші, значків й інших растрових зображень (не обов'язковий);
- створення за допомогою редактора ресурсів додаткових діалогових вікон (не обов'язковий);
- компіляція проекту (компіляція й об'єднання модулів і ресурсів з використанням файла проекту).

Інструменти розробки додатків засобами Visual C++

Компілятор Visual C++ містить ряди убудованих редакторів ресурсів. Щоб відкрити список доступних редакторів, у меню View потрібно вибрати пункт Resource View. Клацнувши правою кнопкою миші на файлі проекту (який має розширення .rc), ви одержите можливість додавати різні типи ресурсів Windows, необхідні вам для створення додатка. Редактори ресурсів дозволяють швидко змінювати існуючі й створювати власні значки, покажчики миші, точкові малюнки, діалогові вікна й багато чого іншого. Крім того, ви можете створювати свої унікальні шрифти й використати їх у діалогових вікнах.

Файл ресурсу містить дані, що включають у виконуючий файл додатки. Але якщо розібратися в технічних деталях, то виявиться, що ці дані розміщаються не в сегменті даних програми. Коли програма завантажується в пам’ять для виконання, ресурси звичайно залишаються на диску. Як приклад, наведемо ситуацію, коли користувач перший раз викликає вікно About. Перш ніж Windows зможе відобразити це вікно, система повинна звернутися до жорсткого диска й завантажити відповідні дані з виконуваного файла в пам’ять.

Файл rc.exe – це компілятор ресурсів. Додаток Windows багаторазово використає такі ресурси, як діалогові вікна, меню й значки. Кожний з них попередньо повинен бути описаний у файлі ресурсів, або файлі сценарію ресурсів. Цей файл створюється, модифікується й

доповнюється за допомогою редакторів ресурсів. За допомогою компілятора ресурсів файл сценарію перетвориться у файл скомпільованих ресурсів, дані з якого уставляються потім в остаточний виконуючий файл додатка. Такий метод обробки ресурсів дозволяє додаткам зберігати інформацію про всі свої ресурси безпосередньо у виконаному файлі.

Редактори ресурсів

В Visual C++ убудований цілий набір редакторів ресурсів, кожний з яких орієнтований на роботу з ресурсами певного типу. Ці редактори надають у наше розпорядження повний набір засобів розробки графічних ресурсів, а також допомагають створювати меню й діалогові вікна – основні засоби введення даних у Windows. Усе це істотно полегшує створення користувальницького інтерфейсу.

Значки, покажчики миші й точкові малюнки

Ресурси даного типу створюються за допомогою різних редакторів, але тому що всі вони відносяться до растрових зображень, робиться це за одним принципом. Редактори значків і покажчиків миші дозволяють створювати апаратно-незалежні зображення, у тому розумінні, що їхній зовнішній вигляд не буде залежати від дозволу монітора.

Вікно редактора значка й покажчика миші показане на рис. 7.1 і 7.2.

У центрі вікна перебуває область редагування, де можна створити значок, покажчик миші або точковий малюнок. Споконвічно розмір цієї області становить 32x32 піксела. Лівіше області редагування розташовується область перегляду. Тут користувач може побачити, як зображення (з реальними розмірами) буде виглядати в системі.

Створити власний значок або покажчик миші дуже просто. Найкраще для цієї мети використати значок або покажчик миші, пропонований за замовчуванням майстром проекту, і відредагувати його відповідно до своїх потреб або смаків. Перейдіть у меню View, виберіть команду Resource View, а потім відкрийте список ресурсів, розташований у правій частині вікна. Якщо в списку потрібний вам тип ресурсу відсутній, виберіть із контекстного меню (викликається щигликом правої кнопки миші) команду Add Resource, у вікні, що відкрилося, укажіть потрібний ресурс і натисніть кнопку New, після чого такий з'явиться на панелі Resource View. Натисніть клавішу Enter, відзначивши необхідний вам тип ресурсу в списку на панелі Resource View, або двічі клацніть на його піктограмі або імені в цьому ж списку (рис. 7.1).

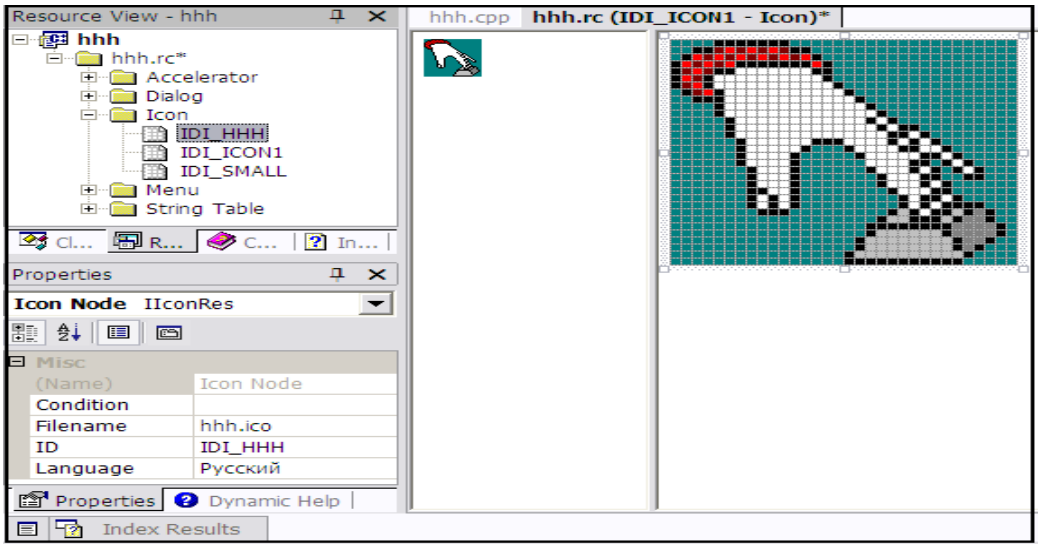

Рис. 7.1. **Створення значка за допомогою редактора зображення**

Редактор надає великий вибір кольорів для малювання значків, а для малювання покажчиків миші – набір півтонів. Щоб зробити потрібні кольори активними, досить клацнути мишею на зразках цих кольорів у палітрі. Після цього можна приступати до малювання значка, покажчика миші або точкового малюнка для свого додатка (рис. 7.2).

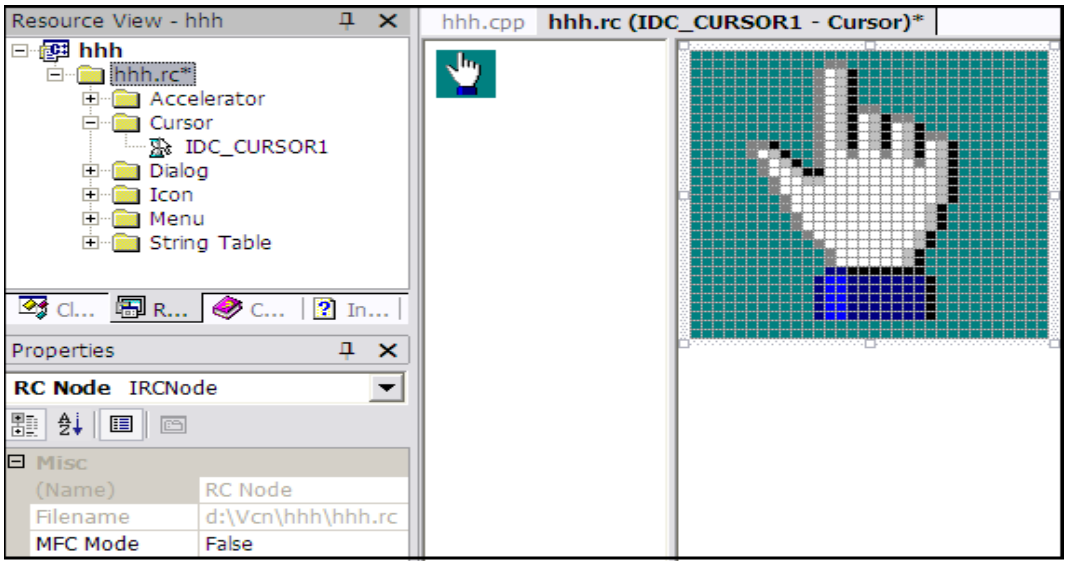

Рис. 7.2. **Редактор зображення в процесі створення нового покажчика миші**

Створення меню

Меню - це список команд і опцій програми. У ряді випадків пункти меню можуть бути представлені растровими зображеннями. Вибрати елемент меню можна за допомогою миші, клавіатури або певних

сполучень клавіш. У відповідь на це Windows пошле додатку повідомлення із вказівкою того, який елемент був задіяний.

Створення меню. Виберіть у меню View команду Resource View, клацніть двічі на файлі .rc, укажіть потрібний ресурс меню й виконайте подвійний щиглик на ньому. Вікно майстра, що з'явилося, за замовчуванням буде містити готове меню, яке при необхідності можна відредагувати за допомогою редактора меню. Альтернативний варіант – використання текстового редактора, у якому набирається опис меню мовою оголошення ресурсів. Але ми настійно рекомендуємо створювати меню за допомогою редактора меню.

Редактор ресурсів може прочитати опис меню з файла сценарію ресурсів (розширення .rc) або з файла скомпільованих ресурсів (розширення .res). Перший файл містить опис ресурсів у текстовому форматі. Якщо є заголовний файл із оголошеннями констант, використовуваних в описі меню, цей файл можна включити у файл ресурсів.

На рис. 7.3 показано меню. Його можна відредагувати за допомогою редактора меню.

В опис меню можна включати різні стилі й атрибути. Наприклад, можна задати маркування обираних елементів спеціальними мітками, указати, що певний пункт меню споконвічно недоступний (виділений сірими кольорами) або що він є простою лінією-роздільником, вирівняти елементи меню по стовпчиках, призначити елементу меню атрибут довідки.

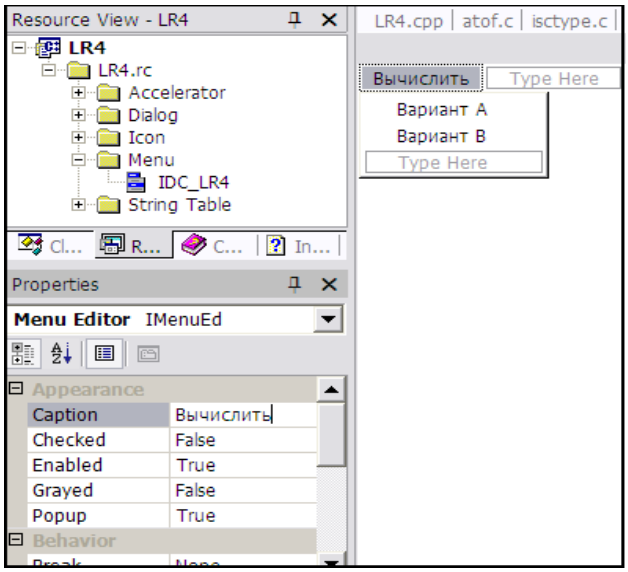

Рис. 7.3. **Редактор ресурсу, використовуваний для створення нового меню**

Меню й компілятор ресурсу меню. Застосувавши стандартні правила, Windows створить меню, що має подібні для всіх додатків зовнішній вигляд і принципи функціонування. Після компіляції буде утворений файл ресурсів з розширенням .res. Цей файл буде використаний компоновщиком для зв'язування ресурсів з виконуючим файлом додатка (exe).

Опис найпростішого меню зрозуміти нескладно. Так виглядає файл сценарію ресурсу:

```
// Menu
IDC_LR4 MENU
BEGIN
POPUP "Обчислити"
BEGIN
MENUITEM "Варіант A",          IDM_ABOUT1
MENUITEM "Варіант B",          IDM_ABOUT2
END
END
```

Вивчаючи цей лістинг, ви напевно помітили ряд ключових слів, використовуваних в описах меню, а саме MENU, POPUP, MENUITEM. Замість операторів BEGIN й END можна використати фігурні дужки ({}).

Опис меню має ім'я IDC_LR4. За ім'ям меню потрібне ключове слово MENU. У даному прикладі описується спадаюче меню "Обчислити", що відкривається шляхом активізації. Елементи в рядку розташовуються в порядку їхнього опису. Якщо вони не містяться в одному рядку, то автоматично додається нова. У кожен момент часу на екрані може бути відкрито тільки одне підменю.

Для того щоб зв'язати з елементом меню клавішу швидкого виклику, у його описі перед відповідною буквою ставиться символ амперсанда (&), причому в меню дана буква буде підкреслена. За допомогою даного символу можна вибирати потрібні пункти меню, використовуючи клавіатуру. Пункт меню можна також вибрати, помістивши покажчик миші на відповідний пункт і клацнувши лівою кнопкою миші. Коли спадаюче меню обрано, Windows негайно відкриває його на екрані, розташовуючи під відповідним пунктом у рядку меню. Кожне ключове слово MENUITEM позначає опис одного пункту з даного меню.

Після назви команди в описі меню йде її ідентифікатор або константа, які містяться в одному із заголовних файлів. Якщо ідентифікатор присутній, він може бути поміщений разом зі значенням, ідентифікованим у заголовному файлі, наприклад, IDM_ABOUT1 й IDM_ABOUT2. Тут IDM – це префікс ідентифікаторів команд меню, а IDD – префікс ідентифікаторів діалогових вікон. Починати префікс із ID загальноприйнято, але не обов'язково. Важливо те, що кожний елемент меню має власний ідентифікатор. Майстер проекту, як видно із наведеного нижче лістингу, містить список таких ідентифікаторів в окремому файлі.

```
// Microsoft Visual C++ generated include file.
// Used by LR4.rc
#define IDC_MYICON              2
#define IDD_LR4_DIALOG          102
#define IDS_APP_TITLE           103
#define IDD_ABOUTBOX1           103
#define IDM_ABOUT               104
#define IDD_ABOUTBOX2           104
#define IDM_EXIT                105
#define IDI_LR4                 107
#define IDI_SMALL               108
#define IDC_LR4                 109
#define IDR_MAINFRAME           128
#define IDC_EDIT2               1006
#define IDC_EDIT3               1007
#define IDC_SUM                 1010
#define IDM_ABOUT1              32771
#define IDM_ABOUT2              32772
#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    129
#define _APS_NEXT_COMMAND_VALUE     32773
#define _APS_NEXT_CONTROL_VALUE     1012
#define _APS_NEXT_SYMED_VALUE       110
#endif
#endif
```

У цьому лістингу ви можете знайти числові значення, асоційовані з ID-значеннями меню.

Діалогові вікна

Використання меню варто розглядати як найпростіший спосіб уведення інформації користувачем. Більш розробленим методом передачі даних у програму є застосування діалогових вікон. Можливість уведення даних за допомогою діалогових вікон забезпечує логічність дій при роботі з Windows-додатком.

У діалоговому вікні користувач може вибирати опції зі списку, установлювати різні прапорці, здійснювати пряме введення в поля тексту й цілих числових значень, а також непряме введення дійсних значень. У діалогових вікнах іноді використовуються й спеціальні елементи керування. Прикладом такого може служити поле зі списком, що становить комбінацію однорядкового поля редагування й списку. Діалогові вікна не тільки служать основним засобом уведення даних у програму, але й істотно полегшують програмування, тому що багато операцій з елементами керування діалогових вікон в Windows автоматизовані.

Щоб відкрити діалогове вікно, досить, як правило, вибрати відповідну команду меню.

Графічний дизайн діалогового вікна конвертується у файл опису ресурсу. Редактор діалогових вікон забезпечує стандартний вигляд і коректну роботу таких файлів, дозволяючи зчитувати й зберігати їх як у текстовому, так й у скомпільованому форматі (.res). Використання текстових файлів спрощує комбінування в одному файлі специфікацій меню й діалогових вікон.

Принципи роботи діалогових вікон. Діалогове вікно становить дочірнє вікно програми, що з'являється при виборі користувачем певної команди з меню. Після того як у діалоговому вікні будуть зроблені необхідні установки, Windows зможе використати нову інформацію для подальшої роботи.

За способом створення вікна діляться на модальні й немодальні. Більшою мірою поширені вікна першого типу. У випадку активізації модального діалогового вікна всі інші вікна й команди додатка стають недоступними доти, поки користувач не закінчить роботу із цим вікном, звичайно за допомогою щиглика на кнопці ОК, Cancel або ін. При натисканні кнопки ОК запускається процедура обробки нових даних,

уведених користувачем, а натискання кнопки Cancel повертає програму до вихідного стану й приводить до скасування всіх уведених даних. У Windows кнопкам ОК й Cancel відповідають стандартні ідентифікатори IDOK й IDCANCEL зі значеннями 1 і 2 відповідно.

Немодальні діалогові вікна більше нагадують стандартні вікна додатків. Користувач може вільно переходити між таким діалоговим вікном і його батьком. Немодальні діалогові вікна використовуються в тому випадку, коли необхідно паралельно працювати з декількома вікнами. Прикладом подібного вікна може служити палітра кольорів у редакторі растрових зображень.

Розробка діалогового вікна

Для діалогового вікна необхідно скористатися відповідним редактором. Редактор діалогових вікон викликається в процесі реалізації ланцюжка дій (рис. 7.4).

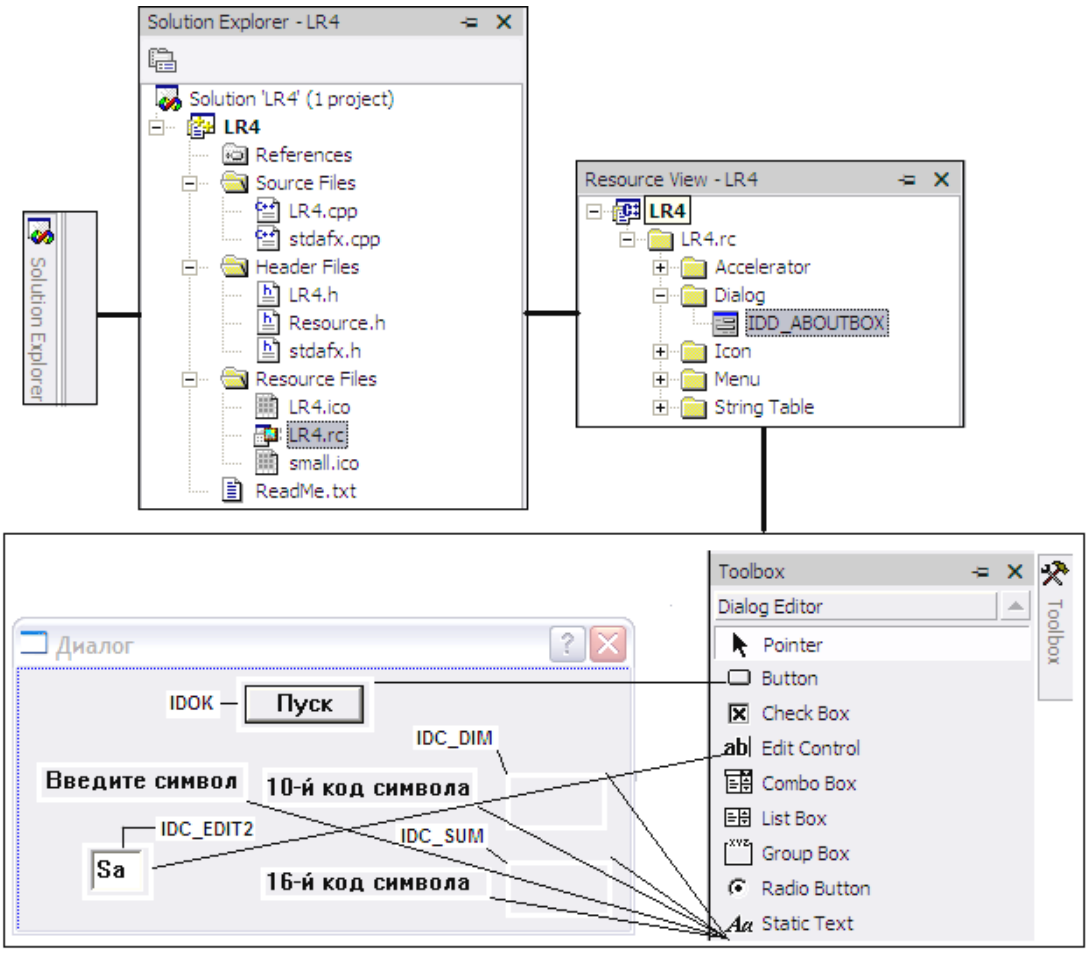

Рис. 7.4. **Послідовність дій при створенні діалогового вікна**

Розміщення елементів керування. Перш ніж приступати до розробки вікна за допомогою редактора ресурсів, варто розібратися в численьних елементах керування, які можна використати в діалогових вікнах. Перелік таких інструментів поданий на рис. 7.4 праворуч (потрібний інструмент викликається щигликом на його імені).

Інструменти компілятора Visual C++ поряд з написами позначені значками:

- напис (Static Text). Становить собою довільний текст, який можна розмістити в будь-якому місці діалогового вікна (наприклад, біля текстового поля) і вказати в такий спосіб його призначення. Значок цього елемента – велика й маленька букви "A";

- рамка групи (Group Box). Оточує групу логічно пов'язаних елементів керування. У верхньому лівому куті рамки автоматично з'являється напис. Значок цього елемента – прямокутна рамка з текстом у верхній частині;

- прапорець (Check Box). Становить собою маленьке квадратне поле, у якому за допомогою щиглика мишею (або за допомогою клавіатури) користувач може встановлювати або знімати мітки. Праворуч від прапорця редактор діалогових вікон автоматично додає пояснювальний напис. Звичайно використовуються групи прапорців, що становлять набори взаємозалежних опцій. У такому випадку користувач може встановити один або кілька прапорців. Значок цього елемента – квадратне поле, яке містить символ "x" або іншу мітку;

- поле зі списком (Combo Box). Це комбінація двох елементів керування: поля й списку. З його допомогою користувач або вибирає елементи зі списку, або додає у список нові елементи. Windows надає у ваше розпорядження кілька типів полів зі списком. Значок даного елемента – три прямокутні поля;

- горизонтальна смуга прокручування (Horizontal Scroll Bar). Звичайно використовується у зв'язку з вікном або елементом керування, що містять текстов чи графічну інформацію. Значок цього елемента – стрілки, спрямовані в різні сторони;

- лічильник (Spin Control). Становить комбінацію двох кнопок, розташованих одна над іншою й утримуючих різнонаправлені стрілки. Звичайно з лічильником пов'язане поле для введення числових значень. Щиглик на відповідній кнопці лічильника приведе до збільшення або

зменшення значення поля. Цей елемент функціонує подібно координатному маніпулятору для керування курсором миші з коліщам прокручування компанії Microsoft. Значок даного елемента – дві піраміди з вершинами, спрямованими нагору й униз;

- регулятор (Slider Control). За замовчуванням складається з горизонтальної смуги з бігунком. Змінивши відповідну властивість, регулятор можна зробити вертикальним. Звичайно регулятор вико ристовується для покрокової зміни пов'язаного з ним значення в заданому діапазоні. Значок цього елемента – горизонтальна смуга з бігунком;

- керований список (List Control). Становить прямокутну область зі списком елементів (це можуть бути й маленькі зображення значків). Значок даного елемента – дев'ять маленьких зображень усередині прямокутника;

- сторінка (Tab Control). Використовується в тих випадках, коли діалогове вікно містить занадто багато різних опцій. Замість того щоб нескінченно збільшувати розмір вікна для виведення на екран всіх опцій, можна створити багатосторінкове вікно, кожна сторінка якого буде представлена своїм ярличком. Таким чином, опції діалогового вікна будуть розбиті на категорії й розміщені на різних сторінках, що перемикають щигликом миші. Значок даного елемента – багатосторінкова папка;

- малюнок (Picture Control). Це прямокутна область, куди установлюється графічне зображення. Значок даного елемента – зображення малюнка;

- поле (Edit Box Control). Прямокутна область, у яку користувач може ввести рядок тексту. Розмір поля можна змінювати залежно від довжини рядків, що вводять. Введений текст може інтерпретуватися або безпосередньо як набір символів або ціле число, або програмно як дійсне число. Значком даного елемента служать символи "ab";

- кнопка (Button). Звичайно є засобом видачі команд на зразок закриття вікна або скасування виконаних установок. За замовчуванням кнопки містять тільки написи, але можуть бути позначені значком або невеликим зображенням. Розмір кнопки можна змінювати. Значок цього елемента – округлений прямокутник;

- перемикач (Radio Button). Становить маленький кружок, праворуч від якого розташований напис. Перемикачі, як і прапорці, звичайно розміщуються групами, але особливістю групи перемикачів є те, що в ній можна вибрати тільки один перемикач. Значок даного елемента – маленьке око;

- список (List Box). Становить прямокутну область із набором текстових елементів. Таким, зокрема, є список файлів поточного каталогу. Значок цього елемента – прямокутник із двома стрілками, спрямованими в протилежні сторони;

- вертикальна смуга прокручування (Vertical Scroll Bar). Звичайно використовується у пов'язуванні з вікном або елементом керування, що містять текстову або графічну інформацію. Значок цього елемента – дві стрілки, спрямовані нагору й униз;

- індикатор виконання програми (Progress Control). Призначений для візуального відображення ходу виконання програмою якого-небудь завдання. Він становить смугу, що заповнюється кольорами (у напрямку зліва направо). Значок даного елемента – зображення смуги;

- гарячі клавіші (Hot Key). Це клавіші або сполучення клавіш, за допомогою яких можна швидко вибирати такі опції, як пункти меню. Значок цього елемента – комбінація клавіші й пальця;

- дерево (Tree Control). Відображає список елементів у вигляді деревовидної структури. За допомогою дерева зручно відображати ієрархічні структури. Значок даного елемента – зображення ієрархічної структури;

- анімація (Animate Control). Це елемент керування, що дозволяє відображати відеокліпи у форматі AVI (Audio Video Interleaved). Кліп становить коротку послідовність растрових зображень. Ця техніка використовується для створення анімаційних покажчиків миші. Значок даного елемента – дві рамки;

- користувальницький елемент керування (Custom Control). Служить для впровадження в діалогове вікно будь-якого розробленого користувачем елемента керування. З появою технології Active така методика перейшла в розряд застарілих, а користувальницький елемент керування існує лише для забезпечення сумісності зі старими проектами. Значок цього елемента – зображення особи.

Для того щоб додати в макет діалогового вікна новий елемент керування, необхідно вибрати його на панелі, установити покажчик миші в потрібне місце вікна й клацнути кнопкою миші. Згодом такий елемент керування можна буде перемістити в інше місце або змінити. Ви можете встановити розмір елемента керування й безпосередньо при його розміщенні. Для цього потрібно при натиснутій лівій кнопці миші простягнути покажчик по діагоналі області, що повинен заповнити елемент.

Компонування діалогового вікна

За допомогою миші розмістіть поле й напис у потрібному місці й задайте їхній розмір. Щиглик мишею всередині макета діалогового вікна дозволить відредагувати заголовок вікна або текстовий рядок, що належить виділеному елементу. Скомпоноване діалогове вікно, готове до введення даних, наведене на рис. 7.5.

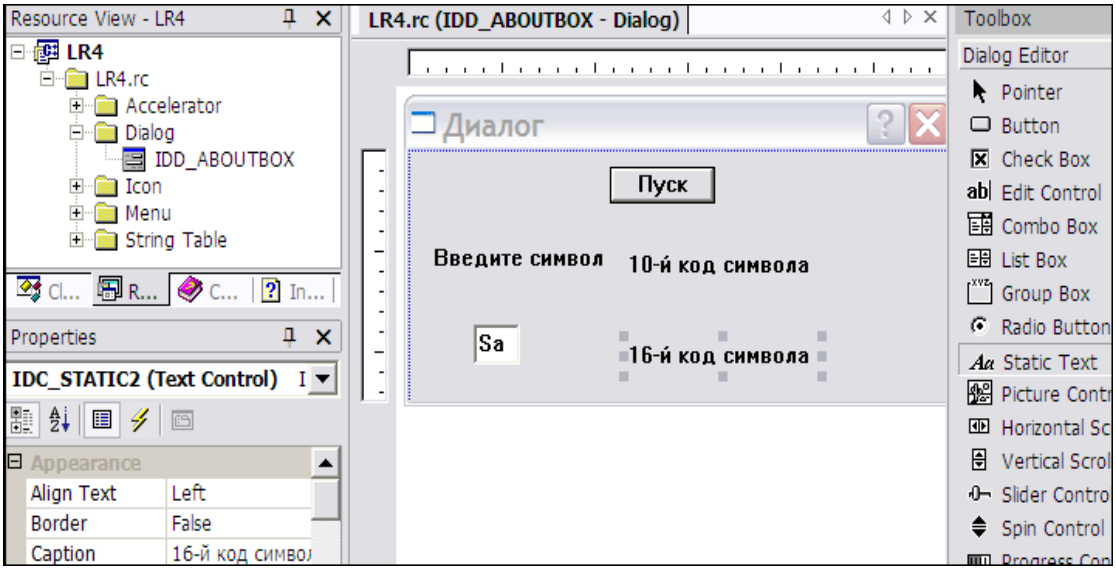

Рис. 7.5. **Діалогове вікно у процесі конструювання**

Перевірка запису ресурсу. Після того як ресурс буде збережений у вигляді .rc-файла, можна переглянути файл сценарію. Для того щоб прочитати дані, що описують діалогове вікно, варто скористатися яким-небудь редактором тексту, що дозволяє працювати з даними ASCII-формату.

Фрагмент файла опису діалогового вікна наведений нижче:

```
// Dialog
//
IDD_ABOUTBOX DIALOGEX 22, 17, 175, 75
STYLE DS_SETFONT | DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_CONTEXTHELP
CAPTION "Діалог"
FONT 8, "System", 0, 0, 0x0
```

```
BEGIN
DEFPUSHBUTTON    "Пуск",IDOK,65,5,34,11,WS_GROUP
EDITTEXT         IDC_EDIT2,21,52,15,12,ES_AUTOHSCROLL
LTEXT            "10-й код символу",IDC_STATIC,71,30,59,10
LTEXT            "16-й код символу",IDC_STATIC,71,57,61,9
LTEXT            "",IDC_DIM,144,30,22,12
LTEXT            "",IDC_SUM,140,55,25,12
LTEXT            "Уведіть символ",IDC_STATIC,8,28,57,12
END
```

Діалоговому вікну привласнене ім'я IDD_ABOUTBOX. Редактор ресурсів додав різні значення його елементів відповідно до встанов.лених вами розмірів. В описі стилю зазначено, зокрема, що дане вікно є модальним (DS_MODALFRAME) і дається перелік використовуваних у ньому елементів керування.

Перший елемент керування становить кнопку Пуск. Текст у лапках – це напис на кнопках. Натискання кнопки реалізується за допомогою ідентифікатора IDOK.

Другий елемент становить вікно редагування й уведення. Ідентифікатор вікна IDC_EDIT2.

П'ять елементів керування LTEXT містять текст. Два елементи з п'яти мають порожні лапки. У ці вікна виводиться числова інформація з додатка.

Рекомендації з підготовки до виконання лабораторної роботи

Ресурсами називаються деякі дані, які визначаються ще до початку роботи програми, особливим чином додаються у виконуваний файл і використовуються при роботі програми. Яскравими прикладами таких даних є:

- іконки;
- курсори миші;
- використовувані в програмі зображення;
- рядок символів;
- меню;
- прискорювачі клавіатури;
- діалогові вікна;
- шрифти;
- ресурси, обумовлені користувачем.

Крім того, що ресурси визначаються до початку роботи програми й додаються у виконуючий файл, у них є ще одна характерна риса. При завантаженні bin-файла у пам'ять, ресурси у пам’ять не завантажуються. Тільки у випадку, якщо той або інший ресурс потрібен для роботи програми, програма сама завантажує ресурс у пам’ять.

Можливість використання того або іншого атрибута як ресурсу не означає, що програміст не може створювати ці атрибути в програмі.

Усі ресурси, заздалегідь визначені в Win32 API, називаються стандартними.

Існує п'ять типів однорядкового опису ресурсу:

- BITMAP;
- CURSOR;
- ICON;
- FONT;
- MESSAGETABLE.

Кожний із цих операторів завантажує файл даних зазначеного типу в дані ресурсу. Після включення цих даних у дані ресурсу можуть застосовуватися функції LoadBitmap(), LoadCursor() і LoadIcon() для безпосереднього доступу до відповідних даних у програмі. Для завантаження повідомлень із файла ресурсу служить функція FormatMessage().

Існує п'ять типів багаторядкового опису ресурсу:

- ACCELERATORS;
- DIALOG;
- MENU;
- RCDATA;
- STRINGTABLE.

Багаторядкові типи опису ресурсів легко розпізнати. У них для визначення блоків даних ресурсу використовуються оператори BEGIN й END. Для виконання лабораторної роботи особливу увагу варто приділити вивченню таких видів ресурсів: піктограми (icon), курсори (cursor), бітові зображення (bitmaps), меню (menu), гарячі клавіші (accelerators), діалогові вікна (dialog), а також набору функцій для роботи з ресурсами. Функції, які створюють або читають об'єкти з ресурсів, повертають покажчик на об'єкт, що використовується в процедурах введення-виведення. Особливу увагу варто приділити таким групам функцій:

- читання об'єктів з ресурсів (LoadCursor(), LoadIcon(), LoadString(), LoadResource(), LoadBitmap(), LoadMenu());

- вибір об'єктів (SelectObject());
- керування об'єктами меню (AppendMenu(), DeleteMenu(), InsertMenu(),ModifyMenu(), RemoveMenu(), CetSubMenu() і т. д.);
- керування рядком стану вікна (CreateStatusWindow()).

Бітові прапори, які описують початковий стан, вид пунктів меню визначаються виходячи з даних табл. 7.1.

Таблиця 7.1

Бітові прапори, що визначають поводження й вид елемента меню

Прапор	Опис
MF_BITMAP	Замість рядка в якості меню застосовується bitmap
MF_BYCOMMAND	Значення, наведене в параметрі uItem функції EnabledMenuItem, становить ідентифікатор пункту меню
MF_BYPOSITION	Значення, наведене в параметрі uItem функції EnabledMenuItem, становить відносну позицію пункту меню зі звітом від нуля
MF_CHECKED	Елемент відмічуваний (зі "значком")
MF_DISABLED	Елемент заборонений
MF_GRAYED	Елемент заборонений і відображається сірими кольорами
MF_HILITE	Елемент підсвічений
MF_MENUBREAK	Горизонтальні меню розміщують наступні елементи в новому рядку, а вертикальні – у новому стовпці
MF_MENUBARBREAK	Те ж, що й попереднє, але у випадку вертикального меню стовпці розділяють вертикальною лінією
MF_MOUSESELECT	Елемент обраний мишею
MF_OWNERDRAW	За промальовування елемента відповідає не система, а програма
MF_POPUP	Елемент викликає поява POPUP-меню більше низького рівня
MF_SEPARATOR	Горизонтальна риса
MF_STRING	Як елементи меню використовується рядок символів
MF_SYSMENU	Елемент із системного меню

Вибір довільного пункту меню може в операційній системі Windows супроводжуватися комбінацією клавіш. У цьому випадку пункт меню вибирається автоматично. При виконанні лабораторної роботи варто врахувати:

- по-перше, необхідність створення у файлі ресурсів таблиці гарячих клавіш;
- по-друге, можливість використання як клавіші або код ASCII-символу, або код віртуальної клавіші (VK);

- по-третє, можливість використання при комбінуванні клавіш керування SHIFT, CTRL або ALT.

Крім того, кожне натискання комбінації оперативних клавіш генерує повідомлення WM_COMMAND. Для перетворення в циклі обробки повідомлень програми уведених комбінацій у команди, зазначені в таблиці оперативних клавіш, застосовується функція TranslateAccelerator().

Якщо функція TranslateAccelerator() повертає ненульове значення, це означає, що вона успішно перетворила комбінацію клавіш і відправила команду у функцію обробки повідомлень програми. Додаток не повинен повторно дозволяти звичайним функціям TranslateMessage() і DispatchMessage() повторно обробляти ці комбінації клавіш. Для запобігання такої ситуації в циклі обробки повідомлень програми у функції WinMain() повинна застосовуватися наступна структура:

```
while( GetMessage( &msg, NULL, 0, 0) )
{
if (!hAccel  || !TranslateAccelerator(hWnd, hAccel, &msg))
{
TranslateMessage( &msg );
DispatchMessage( &msg );
}
}
```

Для того щоб відобразити інформацію про поточний стан програми, виконувані операції і режими, у ній може використатися елемент керування, що називають вікном або рядком стану (status bar).

Рядок стану може бути включений в опис діалогового вікна у файлі ресурсів. Але тому що вікно стану є стандартним вікном, то, природно, що для його створення можуть бути використані функції й стандартні функції CreateWindow() і CreateWindowEX(). При цьому як ім'я класу вікна необхідно задати "STATUSCLASSNAME". Проте, для створення вікна стану передбачена й окрема функція CreateStatusWindow().

Особливістю цієї функції й, відповідно, рядка стану виступає єдиний власний стиль, SBARS_SIZEGRIP, що дозволяє в правий кут рядка стану додати "ручку". Однак наявність у рядка стану єдиного стилю не заважає комбінувати його зі стандартними стилями вікон, наприклад, з WS_CHILD і WS_VISIBLE.

Після того, як рядок стану промальований, іноді буває необхідно розділити його на кілька панелей для того, щоб у кожній панелі

відображати інформацію, логічно не пов'язану з відображуваної в інших панелях. Для того щоб зробити це, необхідно послати рядку стану повідомлення SB_SETPARTS. При цьому wParam цього повідомлення повинен визначати число панелей, а lParam повинен містити покажчик на масив цілих чисел (число елементів масиву повинне дорівнювати wParam). Кожен елемент у цьому масиві повинен визначати позицію (у координатах батьківського вікна) правої межі відповідної частини. Якщо елемент дорівнює -1, то межею панелі вважається права межа рядка стану.

Додаток може визначити число панелей, на які розділений рядок стану, і їхньої координати за допомогою повідомлення SB_GETPARTS. Якщо lParam цього повідомлення дорівнює нулю, то функція, що послала повідомлення, повертає число панелей рядка її стану. При цьому значення wParam ролі не грає. Для того щоб одержати координати панелей, wParam повинен визначати панелі, для яких потрібно одержати координати. А lParam повинен указувати на масив цілих чисел, у який будуть записані координати. У цьому випадку функція також повертає число панелей.

Для того щоб відобразити певний текст у рядку стану, потрібно послати йому повідомлення SB_SETTEXT. У якості wParam цього повідомлення використовується результат логічного додавання двох величин. Перша (iPart) – номер (від нуля) панелі, у якій необхідно відобразити текст. Друга (utype) визначає, як буде виглядати текст. У якості uType можуть бути використані значення, наведені в табл. 7.2.

Таблиця 7.2

Можливі типи рядка стану

Тип	Опис
SBT_NOBORDERS	Панель промальовується без обмежувальних ліній
SBT_POPOUT	Панель промальовується опуклої форми
SBT_RTLREADING	Використовується для мов, у яких читання йде справа наліво, як, наприклад, в арабській мові
SBT_OWNERDRAW	За промальовування панелі відповідає батьківське вікно

Для того, щоб прочитати текст у панелі, необхідно стропі стану послати повідомлення WM_GETTEXT. wParam цього повідомлення повинен містити номер панелі, а lParam – покажчик на рядок, у якому буде записаний текст, що міститься в панелі.

Це основні повідомлення, використовувані при роботі з рядком стану.

Найбільш використовуваним видом ресурсу є діалогові вікна. Діалогові вікна бувають модальними й немодальними. Модальні вікна не дають користувачеві можливості працювати з іншими вікнами, створеними додатком, що породив діалогове вікно, але дозволяють переключатися на роботу з іншими додатками. Для того щоб користувач міг продовжити роботу з іншими вікнами свого додатка, необхідно завершити роботу з діалоговим вікном. Немодальні діалогові вікна не вимагають свого завершення для продовження роботи, і користувач може під час роботи з ними вільно переключатися на будь-який додаток.

При виконанні завдань лабораторної роботи необхідно врахувати наступні особливості діалогових вікон.

Перша особливість діалогових вікон стосується можливості використання як стилів, застосовуваних для опису звичайних вікон (табл. 7.3), так і стилів, застосовуваних тільки в діалогових вікнах (табл. 7.4).

Таблиця 7.3

Перелік бітових прапорів стилю вікна

Прапор	Опис
1	2
WS_BORDER	У вікна є тонка обмежуюча рамка
WS_CAPTION	WS_BORDER | WS_DLGFRAME
WS_CHILD	Створюється дочірнє вікно, що має за замовчуванням тільки робочу область, меню вікна цього стилю не мають
WS_CHILDWINDOW	Те ж, що й WS_CHILD
WS_CLIPCHILDREN	При промальовуванні батьківського вікна область, займана дочірніми вікнами, не промальовується
WS_CLIPSIBLINGS	Дочірнє вікно, яке має цей стиль і перекриває інше дочірнє вікно, при промальовуванні перекриває області, що не промальовується
WS_DISABLED	Створюється вікно, у якому спочатку заборонене одержання даних, уведених користувачем
WS_DLGFRAME	У вікна є рамка, що звичайно буває в діалогових вікнах
WS_GROUP	Вікно є першим вікном групи
WS_HSCROLL	У вікна є горизонтальна лінійка прокручування
WS_ICONIC	Те ж, що й WS_MINIMIZE
WS_MAXIMIZE	Створюється споконвічно максимізоване вікно

Закінчення табл. 7.3

1	2
WS_MAXIMIZEBOX	У вікна є кнопка максимізації
WS_MINIMIZE	Створюється споконвічно мінімізоване вікно
WS_MINIMIZEBOX	У вікна є кнопка мінімізації
WS_OVERLAPPED	Вікно має заголовок і рамку, що обрамляє
WS_OVERLAPPEDWINDOW	WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_THICKFRAME | WS_MINIMIZEBOX | WS_MAXIMIZEBOX
WS_POPUP	Створюється спливаюче вікно
WS_POPUPWINDOW	WS_SYSMENU | WS_BORDER | WS_POPUP
WS_SIZEBOX	Те ж, що й WS_THICKFRAME
WS_SYSMENU	У вікна є системне меню
WS_TABSTOP	Вікно може одержувати клавіатурний фокус при натисканні користувачем клавіші Tab
WS_THICKFRAME	У вікна є досить товста рамка, що дозволяє йому змінювати розміри. Заголовка у вікна немає.
WS_TILED	Те ж, що й WS_OVERLAPPED
WS_TILED_WINDOW	Те ж, що й WS_OVERLAPPEDWINDOW
WS_VISIBLE	Створюється споконвічно відображуване вікно
WS_VSCROLL	У вікна є вертикальна лінійка прокручування

Таблиця 7.4

Стилі діалогового вікна

Прапор	Ефект
1	2
DS_ABSALIGN	Позиціювати діалогове вікно щодо лівого верхнього кута екрана
DS_SYSMODAL	Створити системне модальне діалогове вікно
DS_3DLOOK	Створити діалогове вікно, що має зорову ілюзію тривимірності
DS_FIXEDSYS	Замість SYSTEM_FONT використовується SYSTEM_FIXED_FONT
DS_NOFAILCREATE	Діалогове вікно створюється, незважаючи на те, що при його створенні відбулися помилки
DS_LOCALEDIT	Створити діалогове вікно типу елемента керування
DS_SETFONT	Визначається шрифт, що буде використаний у діалоговому вікні. Система Windows передає дескриптор шрифту кожному елементу керування в діалоговому вікні за допомогою повідомлення WM_SETFONT

Закінчення табл. 7.4

1	2
DS_MODALFRAME	Створити діалогове вікно з модальною рамкою діалогового вікна
DS_NOIDLEMSG	Подавати повідомлення WM_ENTERIDLE, призначені для батьківського вікна, поки відображено дане діалогове вікно
DS_SETFOREGROUND	Помістити діалогове вікно на передній план
DS_CONTROL	Створити діалогове вікно, що також служить як дочірнє вікно іншого діалогового вікна, подібно сторінці у вікні властивостей
DS_CENTER	Діалогове вікно міститься в центрі робочої області
DS_CENTERMOUSE	Установлювати по центру курсор миші в діалоговому вікні
DS_CONTEXTHELP	В області заголовка діалогового вікна для контекстної довідки помістити знак питання (?)

Друга особливість полягає в наявності в Win32 визначених класів вікон для елементів керування діалогових вікон. До таких класів відносяться:

кнопки (клас "button");

списки (клас "listbox");

комбіновані списки (клас "combobox");

вікна редагування (клас "edit");

смуги прокручування (клас "scrollbar");

статичні елементи (клас "static").

У кожного класу є свій певний набір стилів (табл. 7.5 – 7.10), які визначають зовнішній вигляд і поводження елементів керування, що відносяться до даного класу. Керування вікном кожного класу, а також одержання інформації від нього виробляється за допомогою обміну керуючими повідомленнями.

Таблиця 7.5

Стилі вікон класу "кнопки"

Стиль	Опис
1	2
BS_3STATE	Створити прапорець, що має три стани – включений, відключений і недоступний (виділений сірими кольорами). Автоматично стан не змінює
BS_AUT03STATE	Те ж, що й попереднє, за винятком того, що прапорець змінює свій стан, коли його вибирає користувач

Продовження табл. 7.5

1	2
BS_AUTOCHECKBOX	Створюється CheckBox, що автоматично змінює свій стан при виборі користувачем
BS_AUTORADIOBUTTON	Те ж, що й BS_RADIOBUTTON, за винятком того, що перемикач стає обраним, коли на ньому клацає користувач, а всі інші перемикачі в групі стають невибраними
BS_BITMAP	Створити кнопку з растровим зображенням
BS_BOTTOM	Розміщає текст у нижньому краї прямокутника, виділеного для розміщення кнопки
BS_CENTER	Розміщає текст по горизонталі в центрі прямокутника, виділеного для розміщення кнопки
BS_CHECKBOX	Створюється прапорець із написом (CheckBox), показаним праворуч, якщо не використовується стиль BS_LEFTTEXT
BS_DEFPUSHBUTTON	Створюється звичайна кнопка, що спрацьовує при натисканні "Enter" навіть тоді, коли не обрана. Ця кнопка має широку чорну рамку
BS_FLAT	Створити плоску кнопку, що не має тривимірного затінення
BS_GROUPBOX	Створити поле із заголовком, показаним у лівому верхньому куті
BS_ICON	Створити кнопку, на якій відображена піктограма
BS_LEFT	Розміщає текст у лівому краї прямокутника, виділеного для розміщення кнопки
BS_LEFTTEXT	Текст міститься ліворуч від RadioButton'а або CheckBox'а, те ж, що й BS_RIGHTBUTTON
BS_MONO	Кнопка може мати тільки один рядок тексту напису
BS_MULTILINE	При необхідності текст розбивається на кілька рядків
BS_NOTIFY	Дозволяє посилання батьківському вікну нотифікацій них повідомлень BN_DBLCLK, BN_KILLFOCUS й BN_SETFOCUS
BS_OWNERDRAW	За промальовування кнопки відповідає програма, а не система. Батьківське вікно приймає повідомлення WM_MEASUREITEM при створенні кнопки й повідомлення WM_DRAWITEM у будь-який час, коли потрібно вивести на екран кнопку. Цей стиль не слід поєднувати з будь-якими іншими стилями кнопки
BS_PUSHBUTTON	Створюється звичайна кнопка, що виводить повідомлення WM_COMMAND у батьківське вікно, коли відбувається її вибір
BS_PUSHLIKE	Робить CheckBox або RadioButton зовні схожими на PushButton
BS_RADIOBUTTON	Створюється Radio Button, автоматично стан не змінюється
BS_RIGHT	Розміщає текст у правому краї прямокутника, виділеного для розміщення тексту

Закінчення табл. 7.5

1	2
BS_RIGHTBUTTON	RadioButton або CheckBox розміщаються праворуч від напису
BS_TEXT	Усередині або поруч із кнопкою відображається текст
BS_TOP	Розміщає текст у верхньому краї прямокутника, виділеного для розміщення кнопки
BS_USERBUTTON	Застарілий стиль, необхідно використати BS_OWNERDRAW
BS_VCENTER	Розміщає текст по вертикалі в центрі прямокутника, виділеного для розміщення кнопки

Таблиця 7.6

Стилі вікон класу "списки"

Стиль	Опис
1	2
LBS_DISABLENOSCROLL	Показувати на списку вертикальну лінійку прокручування, навіть якщо список не містить достатнє число елементів для прокручування. Лінійка прокручування недоступна доти, поки не з'явиться достатнє число елементів. За замовчуванням, якщо список не містить достатнього числа елементів, лінійка прокручування схована
LBS_EXTENDEDSEL	Дозволяє списку із множинним вибором використати для виділення клавішу Shift разом з мишею або інші клавіатурні комбінації
LBS_HASSTRING	Застосовується зі списком, виведеним власником, для вказівки того, що в список повинні бути уведені рядки. Потім для одержання конкретного елемента в додатку можна використати повідомлення LB_GETTEXT
LBS_MULTICOLUMN	Створити багатостовпцевий список, що може прокручуватися по горизонталі. Для визначення ширини стовпців використовується повідомлення LB_SETCOLUMNWIDTH
LBS_MULTIPLESEL	Дозволити вибирати кілька елементів, клацаючи на кожному елементі для вибору або скасування вибору
LBS_NODATA	Застосовується зі списком, виведеним користувачем, для вказівки того, що в списку немає даних. Цей стиль використовується, коли загальне число елементів у списку перевищує 1000. Повинен також застосовуватися стиль LBS_OWNERDRAWFIXED, а стилі LBS_SORT або LBS_HASSTRINGS не можуть використовуватися
LBS_NOINTEGRALHEIGHT	Список створюється точно такого ж розміру, що зазначений у програмі, вирівнювання не виробляється

Закінчення табл. 7.6

1	2
LBS_NOREDRAW	Список не одержує повідомлень WM_PAINT при внесенні змін. Для зміни цього стилю може використовуватися повідомлення WM_SETREDRAW
LBS_NOSEL	Елементи списку видні, але виділення заборонене
LBS_NOTIFY	Посилає повідомлення батьківському вікну про щиглика або подвійного щиглика клавішею миші
LBS_OWNERDRAWFIXED	Батьківське вікно відповідає за промальовування елементів, всі елементи списку однакової висоти
LBS_OWNERDRAWVARIABLE	Те ж, що й попереднє, але елементи списку можуть бути різної висоти
LBS_SORT	Автоматично сортувати рядки, що додають до списку за допомогою повідомлення LB_ADDSTRING
LBS_STANDARD	LBS_NOTIFY | LBS_SORT | WS_VSCROLL | WS_BORDER
LBS_USETABSTOPS	Список при виведення вхідних у нього рядків повинен розгортати символи табуляції. Передбачені за замовчуванням позиції табуляції розташовані із кроком в 32 одиниці виміру довжини діалогового вікна. Для установки інших позицій табуляції застосовується повідомлення LB_SETTABSTOPS
LBS_WANTKEYBOARDINPUT	Власник списку одержує повідомлення WM_VKEYTOITEM щораз, коли користувач натискає яку-небудь клавішу й список одержує фокус уведення

Таблиця 7.7

Стилі поля класу "комбіновані списки"

Стиль	Опис
1	2
CBS_AUTOHSCROLL	Дозволити горизонтальне прокручування в елементі керування редагуванням поля зі списком
CBS_DISABLENOSCROLL	Вертикальна лінійка прокручування повинна бути видимою, навіть якщо видні всі елементи в списку. Якщо лінійка прокручування не потрібна для відображення всіх елементів у списку, виведення її на екран відмінне. Звичайно лінійка прокручування відображається тільки якщо буде потреба
CBS_DROPDOWN	Створити поле зі списком, що випадає. Список стає видимим тільки будучи обраним

Закінчення табл. 7.7

1	2
CBS_DROPDOWNLIST	Створити поле зі списком, що випадає. Редагування не дозволене; відображається тільки обраний елемент
CBS_HASSTRINGS	Використовується з полем зі списком, виведеним власником, для вказівки того, що до поля зі списком будуть додані рядки. Потім для вибірки конкретного елемента в додатку можна використати повідомлення CB_GETLBTEXT
CBS_LOWERCASE	Показати, що в елементі керування редагуванням дозволене введення тільки символів нижнього регістру, символи верхнього регістру, що вводять, перетворяться в символи нижнього регістру
CBS_NOINTEGRALHEIGHT	Задається розмір поля зі списком. За замовчуванням система Windows визначає розмір поля зі списком так, щоб не можна було відобразити тільки частину елемента
CBS_OEMCONVERT	Уведений текст буде перетворений з набору символів Windows у набір символів OEM, а потім знову – у набір Windows. Це забезпечує правильне перетворення при використанні функції AnsiToOem
CBS_OWNERDRAWFIXED	Створити поле зі списком, виведене власником. Батьківське вікно одержує повідомлення WM_MEASUREITEM при створенні поля зі списком і повідомлення WM_DRAWITEM, коли потрібно вивести елемент
CBS_OWNERDRAWVARIABLE	Те ж, що й CBS_OWNERDRAWFIXED, за винятком того, що розмір кожного елемента в списку можна визначати окремо. Для елемента в поле зі списком викликається повідомлення WM_MEASUREITEM перед викликом повідомлення WM_DRAWITEM для цього елемента
CBS _SIMPLE	Створити просте поле зі списком, де список елементів завжди показаний і поточний вибір перебуває в елементі керування редагуванням
CBS_SORT	Сортувати рядки, внесені у список з використанням повідомлення CB_ADDSTRING
CBS_UPPERCASE	Показати, що потрібно вводити в елементи керування редагуванням тільки символи верхнього регістру, перетворюючи символи нижнього регістру в символи верхнього регістру по мірі їх уведення

Таблиця 7.8

Стилі поля класу "вікна редагування"

Стиль	Опис
ES_AUTOHSCROLL	При необхідності текст у вікні редагування скролірується по горизонталі
ES_AUTOVSCROLL	При необхідності текст у багаторядковому вікні редагування скролірується по вертикалі
ES_CENTER	Текст у вікні редагування вирівнюється по правому краю
ES_LEFT	Текст у вікні редагування вирівнюється по лівому краю
ES_LOWERCASE	Текст, що вводить, перетворюється в малі літери
ES_MULTILINE	Створюється багаторядкове вікно редагування
ES_NOHIDESEL	При втраті вікном редагування фокуса введення виділення з тексту не знімається
ES_NUMBER	Дозволяється здійснювати введення тільки цифр
ES_OEMCONVERT	Символи, що вводять, з одного набору перетворяться в символи з іншого набору
ES_PASSWORD	Усі символи, що вводяться, відображаються у вигляді зірочок
ES_READONLY	Текст у вікні редагування можна тільки переглядати, але не редагувати
ES_RIGHT	Текст у вікні редагування вирівнюється по центру
ES_UPPERCASE	Текст, що вводять, перетворюється у великі букви
ES_WANTRETURN	При натисканні клавіші Enter у багаторядковому вікні система вставляє в текст символ повернення каретки

Таблиця 7.9

Стилі поля класу "смуги прокручування"

Стиль	Опис
1	2
SBS_BOTTOMALIGN	Установити нижній край лінійки прокручування урівень із нижнім краєм прямокутника, визначеного параметрами функції CreateWindow
SBS_HORZ	Створити горизонтальну лінійку прокручування
SBS_LEFTALIGN	Установити лівий край лінійки прокручування урівень із лівим краєм прямокутника, визначеного параметрами функції CreateWindow
SBS_RIGHTALIGN	Установити правий край лінійки прокручування урівень із правим краєм прямокутника, визначеного параметрами функції Create Window
SBS_SIZEBOX	Створити блок керування розміром. Якщо стилі SBS_SIZEBOXBOTTOMRIGHTALIGN або SBS_SIZEBOXTOPLEFTALIGN не зазначені, то висота, ширина й позиція блоку керування розміром мають значення, які визначені параметрами функції CreateWindow

Закінчення табл. 7.9

1	2
SBS_SIZEBOXBOTTOMRIGHTALIGN	Сполучити нижній правий кут блоку керування розміром з нижнім правим кутом прямокутника, визначеного параметрами функції CreateWindow
SBS_SIZEBOXTOPLEFTALIGN	Сполучити лівий верхній кут блоку керування розміром з лівим верхнім кутом прямокутника, визначеного параметрами функції CreateWindow
SBS_SIZEGRIP	Те ж, що й SBS_SIZEBOX, але з активізованим краєм
SBS_TOPALIGN	Установити верхній край лінійки прокручування урівень із верхнім краєм прямокутника, визначеного параметрами функції CreateWindow
SBS_VERT	Створити вертикальну лінійку прокручування

Таблиця 7.10

Стилі вікон класу "статичні елементи"

Стиль	Опис
1	2
SS_BITMAP	Створити статичний елемент керування з растровим зображенням. Параметри nWidth й nHeight ігноруються й перераховуються з урахуванням розмірів растрового зображення
SS_BLACKFRAME	Створити прямокутник, для виведення якого на екран застосовуються такі ж кольори, як і для рамок вікон
SS_BLACKRECT	Створити прямокутник, зафарбований такими ж кольорами, як і рамки вікон
SS_CENTER	Створити статичний елемент керування з вирівнюванням тексту по центру й переносом слів на наступний рядок, якщо буде потреба
SS_ETCHEDFRAME	Виводити на екран елемент керування з утопленою рамкою
SS_ETCHEDHORZ	Аналогічний стилю SS_ETCHEDFRAME, за винятком того, що втопленими будуть відображені тільки верхня й нижня сторони рамки
SS_ETCHEDVERT	Аналогічний стилю SS_ETCHEDFRAME, за винятком того, що втопленими будуть відображені тільки ліва й права сторони рамки
SS_GRAYFRAME	Створити прямокутник, відображуваний такими ж кольорами, як і вікно екрана (робочого стола)
SS_CRAYRECT	Створити прямокутник, зафарбований такими ж кольорами, як фон екрана

Закінчення табл. 7.10

1	2
SS_ICON	Створити статичний елемент керування, що відображає піктограму. Параметри nWidth й niHeight ігноруються й перераховуються відповідно до розмірів піктограми
SS_LEFT	Створити статичний текстовий елемент керування, де заданий текст вирівняний по лівому краю й при необхідності розбитий на рядки
SS_LEFTNOWORDWRAP	Створити статичний текстовий елемент керування, де заданий текст вирівняний по лівому краю. Текст не розбивається на рядки, але символи табуляції розвертаються. Текст, що виходить за кінець рядка, відтинається
SS_NOPREFIX	Відключити інтерпретацію символу амперсандта (&) як ознаки оперативної клавіші
SS_NOTIFY	Посилати батьківському вікну повідомлення STN_CLICKED й STN_DBLCLK, коли користувач клацає або двічі клацає на елементі керування
SS_OWNERDRAW	За виведення на екран статичного елемента керування відповідає його власник
SS_REALSIZEIMAGE	Заборонити зміну розмірів статичного елемента керування відповідно до розмірів піктограми або растрового зображення
SS_RIGHT	Створити статичний текстовий елемент керування, де текст вирівняний по правому краю й при необхідності розбитий на рядки
SS_RIGHTJUST	Нижній правий кут статичного елемента керування зі стилем SS_BITMAP або SS_ICON залишається нерухомим при зміні розмірів елемента керування. Змінюється тільки положення верхньої й лівої сторін відповідно до розмірів растрового зображення або піктограми
SS_SIMPLE	Створити простий статичний текстовий елемент керування з текстом, вирівняним по лівому краю
SS_SUNKEN	Виводити частково втоплену рамку навколо статичного елемента керування
SS_WHITEFRAME	Створити прямокутник, що має при відображенні на екрані такі ж кольори, як і тло вікон
SS_WHITERECT	Створити прямокутник, зафарбований такими ж кольорами, як і тло вікна

Приклад

Обчислити вирази a і b (рис. 7.6) при заданих x, y і z. Програму оформити у вигляді графічного додатка. Уведення й виведення інформації організувати за допомогою діалогових вікон.

$$x := -0.62 \quad y := 0.82 \quad z := 25$$

$$a := y^{(|x|)^{\frac{1}{3}}} + (\sin(y-3))^3 \quad a = 0.293 \qquad b := y \cdot \frac{\left(\operatorname{atan}(z) - \frac{\pi}{6}\right)}{\left[|x| + \frac{1}{(y^2+1)}\right]} \quad b = 0.678$$

Рис. 7.6. **Вирази a і b**

Меню головного вікна один пункт "Обчислити" і два випадаючих підпункти "Варіант А", "Варіант В" наведені на рис. 7.7.

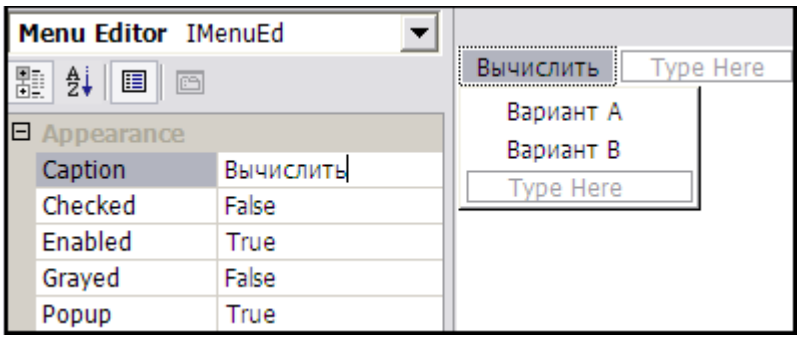

Рис. 7.7. **Структура меню**

Під це меню віконна функція зміниться й прийме вигляд:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
WPARAM wParam, LPARAM lParam)
{
        switch (message)
        {
        case WM_COMMAND:
                switch (LOWORD(wParam))
                {
```

```
                case IDM_ABOUT1:
                    DialogBox(hInst,          (LPCTSTR)IDD_ABOUTBOX1,
hWnd, (DLGPROC)About1);
                    break;
                case IDM_ABOUT2:
                    DialogBox(hInst,          (LPCTSTR)IDD_ABOUTBOX2,
hWnd, (DLGPROC)About2);
                    break;
                case IDM_EXIT:
                    DestroyWindow(hWnd);
                    break;
                default:
                    return DefWindowProc(hWnd, message, wParam,
lParam);
                }
                break;
            case WM_DESTROY:
                PostQuitMessage(0);
                break;
            default:
                return DefWindowProc(hWnd, message, wParam, lParam);
            }
            return 0;
        }
```

Діалогові вікна для організації уведення й виведення, а також обчислення виразів a і b мають вигляд (рис. 7.8):

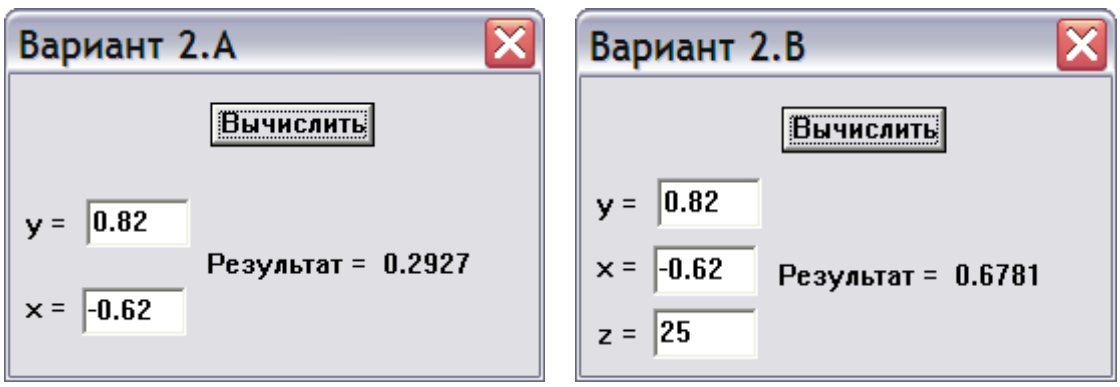

Рис. 7.8. **Діалогові вікна**

Функції взаємодії з діалоговими вікнами й обчислення виразів a й b наведені нижче:

```
LRESULT CALLBACK About1(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
	{
		switch (message)
		{
		case WM_COMMAND:
			switch (LOWORD(wParam))
			{
			case IDOK:
			{
			char sz[15],sz[15];
			double x,y,a;
			GetDlgItemText(hDlg,IDC_EDIT2,sz,12) ;
			x = atof(sz);
			GetDlgItemText(hDlg,IDC_EDIT3,sz,12) ;
			y = atof(sz);
			x=abs(x);
			x=pow(x,1./3);
			x=pow(y, x);
			a=sin(y-3.);
			a=pow(a,3.);
			a=a+x;
			sprintf(sz,"%7.4f",a);
			SetWindowText(GetDlgItem(hDlg,IDC_SUM),sz);
			}
			break;
			case IDCANCEL:
			{
			EndDialog(hDlg, LOWORD(wParam));
			return TRUE;
			}
			break;
			}
```

```
        }

        return FALSE;
    }
    LRESULT CALLBACK About2(HWND hDlg, UINT message, WPARAM
wParam, LPARAM lParam)
    {
        switch (message)
        {
        case WM_COMMAND:
            switch (LOWORD(wParam))
            {
            case IDOK:
            {
            char sz[15],sz[15];
            double x,y,z,b;
            GetDlgItemText(hDlg,IDC_EDIT2,sz,12) ;
            x = atof(sz);
            GetDlgItemText(hDlg,IDC_EDIT3,sz,12) ;
            y = atof(sz);
            GetDlgItemText(hDlg,IDC_EDIT4,sz,12) ;
            z = atof(sz);
            b=y*(atan(z)-3.1415/6);
            x=abs(x)+1./(y*y+1);
            b=b/x;
            sprintf(sz,"%7.4f",b);
            SetWindowText(GetDlgItem(hDlg,IDC_SUM),sz);
            }
            break;
            case IDCANCEL:
            {
            EndDialog(hDlg, LOWORD(wParam));
            return TRUE;
            }
            break;
            }
```

```
        }

        return FALSE;
    }
```

Фрагменти функцій About1 й About2, виділені жирним шрифтом, для кожного варіанта будуть відрізнятися.

Прототипи діалогових функцій About1 й About2 мають вигляд:

```
LRESULT CALLBACK About1(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK About2(HWND, UINT, WPARAM, LPARAM);
```

Робота графічного додатка в комплексі наведена на рис. 7.9.

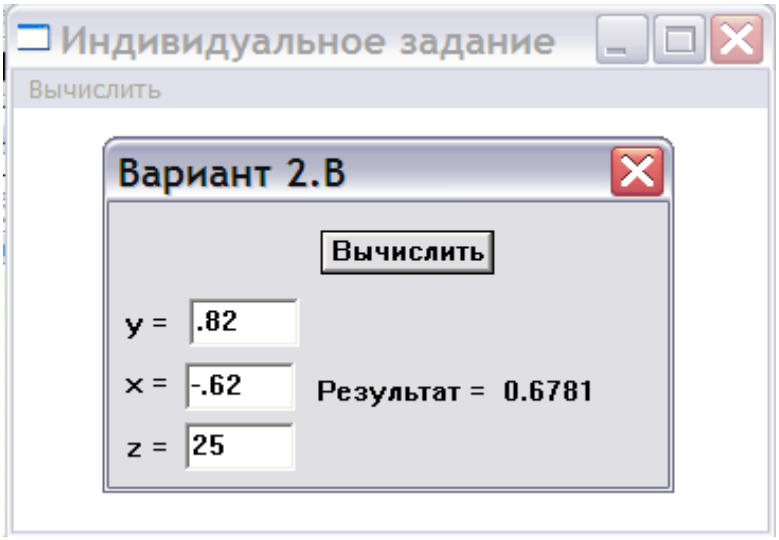

Рис. 7.9. **Графічний додаток**

Завдання до лабораторної роботи

Завдання 1

Побудувати головне меню додатка, що має деревоподібну структуру. Один з пунктів меню повинен бути поданий у вигляді заданого зображення. Вибір кожного пункту меню повинен ідтримуватися набором гарячих клавіш.

Варіант 1

Головне меню містить три групи команд:

Головне меню містить розділ "Малюнок" з іменами чотирьох геометричних фігур, зображення яких зберігається у вигляді окремих

файлів на диску. Після вибору фігури відобразити її в клієнтській частині вікна. При натисканні правої клавіші миші над будь-якою відображеною фігурою на місці натискання відобразити плаваюче меню з відповідними обраній фігурі командами.

Створити контекстне меню, що містить назви можливих малюнків, а також вихід із програми, що повинен бути логічно відділений. Виклик цього меню здійснюється тільки в рамках малюнка, розташованого в клієнтській області екрана виконуючого додатка.

Третя група виконує заміну назв пунктів меню з російської мови на українську і навпаки.

Тип зображення – яскраво-червона зірка.

У рядку стану повідомляти про дії з малюнками й поточні координати курсору миші по горизонталі й вертикалі.

Варіант 2

Головне меню містить три групи команд:

Головне меню містить розділ "Файл" з рядками "Створити", "Відкрити" й "Вихід". При виборі рядків "Створити" або "Відкрити" створити вікно, що містить розділ меню "Фігури" зі списком імен геометричних фігур; при виборі імені відобразити фігуру з таким ім'ям у клієнтській області головного вікна й позначити його знаком оклику. При повторному виборі імені забрати фігуру й видалити галочку.

Створити контекстне меню, що містить всі пункти головного меню, перелік фігур так, щоб центр меню по горизонталі й вертикалі збігався з координатами x і y курсору відповідно.

Третя група виконує заміну назв пунктів меню з російської мови на німецьку і навпаки.

Тип зображення – діаграма.

У рядку стану відображати поточну інформацію про фігуру.

Варіант 3

Головне меню містить три групи команд:

Головне меню містить розділ "Фігури" з іменами геометричних фігур: коло, квадрат, трикутник, прямокутник і шестикутник. При виборі назви фігури в головне меню додати розділ з назвою фігури й переліком основних її параметрів у цьому розділі. При повторному натисканні повинні зникнути цей розділ і галочка.

Друга група пунктів у меню містить 3 пункти, кожний з яких складається ще з двох, розділених сепаратором. Вибір кожного пункту

повинен супроводжуватися відкриттям свого власного невеликого вікна, меню якого складається з одного пункту "Вихід". При його виборі закривається дане вікно. У рядку заголовка необхідно відображати унікальну назву, що відповідає обраному пункту головного меню.

Третя група виконує заміну назв пунктів меню з української мови на німецьку і навпаки.

Тип зображення – жовтий тюльпан.

Про дії з фігурами й стан відповідних рядків меню повідомляти в рядку стану.

Варіант 4

Головне меню містить три групи команд:

Головне меню містить розділ "Файл" з пунктами "Створити", "Знайти", "Зберегти", "Закрити", "Відправити" й "Вихід", а розділ "Виправлення" – рядки "Вирізати", "Вклеїти" й "Копіювати". Після вибору команди "Відправити" виробляється видалення розділу "Виправлення", а сам пункт заміняється на "Відновити". При виборі "Відновити" зробити відновлення розділу "Виправлення".

Під пунктом меню розташувати панель інструментів. Кнопки панелі інструментів повинні відображати всі можливі варіанти вибору меню.

Третя група виконує заміну назв пунктів меню з української мови на англійську і навпаки.

Тип зображення – довільна машина.

У рядку стану повідомляти про дії з рядками меню.

Варіант 5

Головне меню містить три групи команд:

Робочу область вікна повністю займають два тимчасових вікна. Головне меню першого вікна містить розділ "Файл" з рядками "Відкрити", "Закрити" й "Вихід". Якщо вибрати рядок "Відкрити", то в другому вікні з'являється головне меню з розділом "Виправлення". Якщо вибрати пункт "Закрити", то в другому вікні видаляється головне меню з розділом "Виправлення".

Команди меню продублювати кнопками на панелі інструментів у кожному з вікон.

Третя група виконує заміну назв пунктів меню з англійської мови на російську і навпаки.

Тип зображення – комп'ютер, що включає системний блок, монітор і клавіатуру.

Рядки стану всіх вікон відображають інформацію про активність вікон і переміщенні по рядках меню.

Варіант 6

Головне меню містить три групи команд:

Головне меню містить розділ "Файл" з рядками "Новий", "Відкрити" й "Вихід" і розділ "Допомога" з рядками "Зміст" й "Про програму". При виборі рядка "Зміст" з'являються рядки "Введення", "Частина 1", "Частина 2" ..., а при виборі рядка "Частина ..." з'являються рядки "Розділ 1", "Розділ 2"...

При запуску другого екземпляра додатка повідомити про заборону запуску декількох екземплярів, на передній план перемістити перший екземпляр додатка, 3 рази змінити тло його вікна, видаючи звукове попередження, і завершити роботу другого екземпляра.

Третя група виконує заміну назв пунктів меню з української мови на російську і навпаки.

Тип зображення – герб України.

У рядку стану відобразити шлях до поточного рядка меню.

Варіант 7

Головне меню містить три групи команд:

Перша група містить рядки "Сховати", "Показати", "Масштаб", "Властивості". Рядок "Масштаб" указує на тимчасове меню із чотирьох залежних рядків: "50%", "100%", "150%" й "200%", при виборі один з них відзначається рожевим квадратом.

При виборі одного з пунктів меню другої групи (задати самостійно) виробляється дозвіл, заборона або виділення сірими кольорами іншого пункту меню. За замовчуванням пункти меню повинні бути дозволені.

Третя група виконує заміну назв пунктів меню з української мови на російську і навпаки.

Тип зображення – довільний головний убір.

У рядку стану повідомляти про рядки меню, включаючи стан рядків.

Варіант 8

Головне меню містить три групи команд:

Головне меню містить розділ "Користувачі", у якому перелічені рядки з іменами типів користувачів (студенти, викладачі, інженери й т. д.). Після вибору типу користувача цей розділ зникає й з'являється розділ "Дані", у якому перераховані загальні для всіх типів користувачів і типові тільки для обраного типу рядка дані.

При виборі одного з пунктів меню другої групи виробляється додавання нового рядка меню із двома пунктами, що дозволяють додавати (видаляти) нові рядки меню. Обмежити загальну кількість додавань.

Третя група виконує заміну назв пунктів меню з української мови на німецьку і навпаки.

Тип зображення – портрет.

Рядок стану містить поточну інформацію про користувача й дані, навіть якщо вибір був зроблений за допомогою акселератора.

Варіант 9

Головне меню містить три групи команд:

Головне меню містить розділ "Кольори" із трьома іменами стандартних кольорів Windows і розділ "Фігури" з іменами трьох плоских фігур. Можливі комбінації сполучень кольорів і фігур зберігати в .bmp файлі. Після вибору кольорів і фігури відобразити в клієнтській області фігуру обраними кольорами, а відповідні рядки меню відзначити символом "V".

Забезпечити вибір відповідної кольорової фігури на панелі інструментів.

Третя група виконує заміну назв пунктів меню з російської мови на німецьку і навпаки.

Тип зображення – рука людини.

У рядку стану забезпечити видачу підказки про кольори й тип фігури.

Варіант 10

Головне меню містить дві групи команд:

На місці натискання правою кнопкою миші відобразити контекстне меню, відзначені галочкою елементи якого вказують на тимчасові меню, рядки яких служать перемикачами.

При виборі пунктів меню другої групи виробляється динамічна зміна головного меню й набору гарячих клавіш. При кожному виборі цього пункту меню виробляється додавання нових пунктів у нього з відповідними гарячими клавішами F1, F2, F3 і т. д. до 12. При виборі цього нового пункту меню повинне видаватися вікно повідомлення, у якому підтверджувався б вибір доданого пункту (наприклад, за допомогою різних рядків заголовка).

Третя група виконує заміну назв пунктів меню з української мови на німецьку і навпаки.

Тип зображення – сонце.

Рядок стану відбиває шлях переміщення курсору по рядках меню.

Варіант 11

Головне меню містить три групи команд:

Розділ "Файл" головного меню містить рядки "Створити", "Відкрити", "Демоверсія" й "Вихід". При виборі рядка "Створити" або "Відкрити" створити вікно, що перекривається, з розділом меню "Ефекти". Список рядків розділу "Ефекти" залежить від стану рядка "Демоверсія".

Друга група пунктів меню управляє новими вікнами. При виборі будь-якого пункту меню виробляється створення нового вікна, а пункт меню повинен бути відображений як обраний символом "§". Повторне натискання на цей пункт меню повинне робити закриття відповідного вікна й зняття символу "§". Набір гарячих клавіш повинен складатися з комбінації мінімум 3 клавіш.

Третя група виконує заміну назв пунктів меню з української мови на англійську і навпаки.

Тип зображення – довільний птах.

Стандартні команди меню продублювати кнопками панелі інструментів. У додатковому вікні видавати повідомлення про рядки меню.

Варіант 12

Головне меню містить три групи команд:

Розділ "Користувачі" головного меню містить список користувачів. При виборі користувача в головному меню з'являються додаткові розділи. При зміні користувача міняються й ці розділи. Про дії з рядками меню повідомляти в рядку стану.

Друга група пунктів меню повинна мати п'ять взаємовиключних підпунктів, при цьому після кожного третього вибору будь-якого пункту меню він повинен стати сірим. Вибір кожного пункту повинен відзначатися спеціальним символом "~".

Третя група виконує заміну назв пунктів меню з російської мови на англійську і навпаки.

Тип зображення – сніжинки.

У рядку стану забезпечити підказку про користувачів або виводити дані про кількість обраних пунктів меню.

Варіант 13

Головне меню містить три групи команд:

Робочу область вікна додатка повністю займає тимчасове вікно з порожнім головним меню. Головне меню вікна додатка містить розділ "Файл" з рядками "Відкрити" й "Закрити" (заблоковано). При виборі

команди "Відкрити" створити головне меню тимчасового вікна з розділом "Виправлення" з рядками "Вирізати", "Копіювати" й "Видалити". Після цього заблокувати рядок "Відкрити" і розблокувати команду "Закрити". При виборі рядка "Закрити" повернутися до вихідного стану. Команди меню продублювати кнопками панелі інструментів.

Друга група пунктів меню повинна керувати першою групою, шляхом установки будь-якого пункту в будь-який стан. Підтвердити графічно установку в стан відповідною галочкою біля пунктів меню й активним або сірим станом кнопок на панелі інструментів.

Третя група виконує заміну назв пунктів меню з російської мови на німецьку і навпаки.

Тип зображення – оргтехніка.

У рядку стану забезпечити підказку про можливі дії користувачеві з обраними пунктами меню.

Варіант 14

Головне меню містить три групи команд:

Створити плаваюче меню для вибору й установки виду курсору миші. При виборі імені виду курсор миші повинен прийняти відповідний вигляд. Передбачити повернення в початковий стан.

При виборі одного з пунктів меню другої групи (задати самостійно) у вікні повідомлення видається інформація про зазначений пункт меню, що включає розмір структури, стиль меню, максимальну висоту меню, пензель, застосовувану для відображення, контекстний ідентифікатор довідки.

Третя група виконує заміну назв пунктів меню з російської мови на українську і навпаки.

Тип зображення – футбольний м'яч.

Про дії з курсором миші повідомляти в рядку стану мовою, установленою у п. 3.

Варіант 15

Головне меню містить три групи команд:

Головне меню містить розділ "Файл", у якому перераховані пункти з іменами команд "Створити", "Відкрити" й "Вихід". Після вибору команд "Створити" або "Відкрити" додаються рядки "Зберегти" й "Друк", а також розділ "Виправлення" з рядками "Вирізати", "Вклеїти" й "Копіювати".

При виборі одного з пунктів меню другої групи (задати самостійно) виробляється додавання нового пункту в головне меню додатки, причому кількість таких додавань не повинна перевищувати кількості

вихідних пунктів меню, а вибір цього пункту меню приводить до його видалення.

Третя група виконує заміну назв пунктів меню з німецької мови на українську і навпаки.

Тип зображення – корабель.

У рядку стану забезпечити підказки до команд меню при наведенні на них курсору миші.

Варіант 16

Головне меню містить три групи команд:

Головне меню містить розділ "Файл", у якому перераховані рядки з іменами команд "Створити", "Відкрити" й "Вихід", які можуть бути відзначені як залежні перемикачі. Після вибору рядків "Створити" або "Відкрити" додати розділ "Виправлення" з командами "Вирізати", "Вклеїти" й "Копіювати", які можуть бути відзначені як незалежні прапорці.

За допомогою функцій Win32 API визначити пункт меню в другій групі, на якому встановлений курсор миші, й зробити цей пункт утопленим (тобто розташованим нижче стосовно інших пунктів меню)

Третя група виконує заміну назв пунктів меню з української мови на англійську і навпаки.

Тип зображення – літак.

Рядок стану відбиває шлях переміщення курсору по рядках меню.

Варіант 17

Головне меню містить три групи команд:

Головне меню містить розділ "Файл" з рядками "Створити", "Відкрити", "Видалити" й "Вихід". При виборі рядка "Створити" або "Відкрити" створити вікно, що містить меню з розділами "Виправлення" й "Ефекти". При виборі рядка "Видалити" створити вікно, що містить меню з розділами "Виправлення" й "Ефекти".

При виборі одного з пунктів меню другої групи (задати самостійно) виробляється дозвіл, заборона або виділення сірими кольорами іншого пункту меню. За замовчуванням пункти меню повинні бути дозволені.

Третя група виконує заміну назв пунктів меню з української мови на російську і навпаки.

Тип зображення – прапор України.

У рядку стану забезпечити підказки до команд меню при наведенні на них курсору миші, а також виводиться додаткова підказка про призначені пунктам меню комбінації оперативних клавіш.

Варіант 18

Робочу область вікна додатка, повністю займає тимчасове вікно з порожнім головним меню. Головне меню вікна додатка містить розділ "Файл" з рядками "Відкрити" й "Закрити" (заблоковано). При виборі команди "Відкрити" створити головне меню тимчасового вікна з розділом "Виправлення" з рядками "Вирізати", "Копіювати" й "Видалити". Після цього заблокувати рядок "Відкрити" і розблокувати команду "Закрити". При виборі рядка "Закрити" повернутися до вихідного стану.

У другій групі пунктів меню (задати самостійно) виробляється додавання нового рядка меню, що містить один новий пункт, вибір якого приводить до видалення цього рядка меню.

Вибором одного з пунктів меню "Виправлення" повинна здійснюватися заміна назв пунктів меню з російської мови на англійську і навпаки.

Тип зображення – довільне найменування валюти.

У рядку стану забезпечити підказки до команд меню при наведенні на них курсору миші, а також виводиться додаткова підказка про призначені пунктам меню комбінації оперативних клавіш мовою, що задана в п. 3.

Варіант 19

Головне меню містить три групи команд:

Головне меню містить розділ "Файл" з іменами команд "Створити", "Відкрити", "Зберегти", "Закрити", "Друк" й "Вихід", розділ "Виправлення" з рядками "Вирізати", "Вклеїти" й "Копіювати". Після вибору команди "Закрити" видалити розділ "Виправлення".

Друга група пунктів у меню містить 4 пункти, при цьому довільний вибір одного з них повинен бути відзначений перемикачем "(". За замовчуванням у вихідному стані обраним повинен бути передостанній елемент із групи.

Третя група виконує заміну назв пунктів меню з української мови на англійську і навпаки.

Тип зображення – карикатура.

У рядку стану дублювати найменування обраних пунктів меню залежно від установленої в п.3 мови.

Варіант 20

Робочу область вікна додатка займають два тимчасових вікна. Головне меню першого тимчасового вікна містить розділ "Файл" з командами "Створити", "Відкрити", "Демоверсія" й "Вихід". Передбачити можливість маніпулювання станом меню "Демоверсія" із другого вікна.

Головне меню другого вікна містить два різних деревоподібних меню додатка. При виборі однієї з опцій довільного меню (задати самостійно) додатка по черзі завантажується одне зі створених меню, які встановлюються в якості головного меню.

Вибором одного з пунктів меню першого вікна здійснюється заміна всіх назв пунктів меню в обох вікнах з української мови на німецьку і навпаки.

Тип зображення – молодий півмісяць.

При наведенні на пункт меню у вигляді іконки слід вказувати в рядку стану шлях до файла, що містить її образ.

Завдання 2

У головне меню додатка, виконане в завданні 1, додати пункти, що відповідають таким діалоговим вікнам:

1. Діалоговому вікну настроювання виду меню.

Настроювання виду меню включають додавання й видалення нових пунктів меню, зміну назви існуючих пунктів. Діалог повинен включати:

- з однієї сторони, вибір пункту меню за допомогою заданого в п.1 індивідуального завдання елемента керування;
- з іншої сторони, можливі стани (звичайний, заборонений й сірий) і можливі операції з пунктами меню (додати, видалити або модифікувати);
- мати обмеження на мінімальну й максимальну кількість пунктів меню, що видаляються / додаються (відповідно до п. 1 індивідуального завдання).

2. Діалоговому вікну з інформацією про автора, номер групи, призначення програми і довільним графічним об'єктом (наприклад, особистою фотографією).

3. Індивідуальному діалоговому вікну, заданого в п. 2 свого варіанта.

Вибір кожного пункту меню повинен підтримуватися довільним набором гарячих клавіш.

Варіант 1

Тип елемента керування: Combobox. Мінімальне число пунктів меню – 4, максимальне число пунктів меню – 12.

Створити макет калькулятора, що містить поле уведення чисел, кнопки цифр від 0 до 9, десяткової точки, арифметичної дії ("·", "/") і знака "=".

Варіант 2

Тип елемента керування: Listbox. Мінімальне число пунктів меню – 3, максимальне число пунктів меню – 11.

У першій сторінці блокнота встановити стиль вирівнювання тексту ("По лівій межі", "По правій межі" й "По центру"), у другій сторінці – шрифт ("Звичайний", "Напівжирний", "Курсив" й "Напівжирний курсив"). Після натискання кнопки ОК у центрі клієнтської області вікна блокнота обраним стилем і шрифтом відобразити текст "Пробний висновок".

Варіант 3

Тип елемента керування: Combobox. Мінімальне число пунктів меню – 7, максимальне число пунктів меню – 15.

У діалоговому вікні розташувати тимчасове вікно. У центрі цього вікна намалювати поле з 9 клітинок й у кожнім полі вивести його значення (від 1 до 9). Поля виділяти різними кольорами, обраними за допомогою одного з 9 довільних кольорів.

Варіант 4

Тип елемента керування: Listbox. Мінімальне число пунктів меню – 6, максимальне число пунктів меню – 20.

У чотирьох комбінованих списках діалогового вікна перебувають прізвища, імена та по батькові студентів (задати самостійно в кількості не менш 10) і номера їхніх телефонів (подібність із реальними можна не забезпечувати). Передбачити адекватний і синхронний вибір даних із чотирьох списків. При виборі в будь-якому списку (наприклад, у списку прізвищ) забезпечити вибір відповідних даних в інших списках. Після натискання клавіші "Застосувати" у заголовку головного вікна додатка відобразити дані про студента.

Варіант 5

Тип елемента керування: Combobox. Мінімальне число пунктів меню – 5, максимальне число пунктів меню – 12.

Перша сторінка блокнота запитує ім'я користувача, пароль і його підтвердження. У випадку успішної реєстрації користувача розкрити другу сторінку зі списком доступних цьому користувачеві даних. Після натискання кнопки ОК блокнота відобразити ім'я користувача й обрані дані.

Варіант 6

Тип елемента керування: Listbox. Мінімальне число пунктів меню – 7, максимальне число пунктів меню – 13.

Діалогове вікно містить тимчасове вікно. У центрі цього вікна намалювати кругову діаграму. Кількість секторів й їхні процентні частки задати за допомогою органів керування панелі (при необхідності обмежити). Для вибору кольорів зафарбовування секторів використати стандартну панель вибору кольорів.

Варіант 7

Тип елемента керування: Combobox. Мінімальне число пунктів меню – 4, максимальне число пунктів меню – 15.

При виборі рядка меню "Установити пароль" створити діалогове вікно, де за допомогою клавіатури ввести ім'я користувача й пароль, потім кнопку ОК. Уведення пароля супроводжувати показом символу "*". Після цього створити іншу діалогову панель для підтвердження введених даних. Після підтвердження даних завершити роботу вікна й відобразити введені дані.

Варіант 8

Тип елемента керування: Listbox. Мінімальне число пунктів меню – 3, максимальне число пунктів меню – 16.

У тимчасовому вікні створити порожні списки для зберігання імен файлів і шляхів до них. За допомогою стандартної панелі вибрати імена файлів у будь-якому каталозі й записати їхні дані в списки тимчасового вікна. Тимчасове вікно повинне обробляти клавіатурні повідомлення так само, як і модальні вікна.

Варіант 9

Тип елемента керування: Combobox. Мінімальне число пунктів меню – 4, максимальне число пунктів меню – 14.

Діалогове вікно містить 3 горизонтальні смуги прокручування для регулювання значень RGB-складових кольору й 3 статичні органи керування для їх відображення. Нижче смуг прокручування перебуває тимчасове вікно, у якому відображається текст "Пробний висновок" поточним значенням кольору на прозорому тлі. Після будь-якої зміни складових відобразити текст цими кольорами.

Варіант 10

Тип елемента керування: Listbox. Мінімальне число пунктів меню – 5, максимальне число пунктів меню – 16.

Вікно містить розділ меню "Вид" з рядками "Звичайний", "Стислий" й "Скорочений". На звичайній панелі розташувати тимчасове вікно із тлом "Робочий стіл", горизонтальну смугу прокручування й кілька

статичних органів. На стислій панелі повинні бути відсутні статичні органи, а на скороченій панелі відсутня і смуга прокручування. При зміні виду змінити й розміри панелі.

Варіант 11

Тип елемента керування: Combobox. Мінімальне число пунктів меню – 7, максимальне число пунктів меню – 18.

У діалоговому вікні розташувати список з іменами констант системних кольорів. Правіше списку відобразити тимчасове вікно. Вибрати назву системних кольорів і нажати кнопку "Готово". Після цього обраними кольорами зафарбувати робочу область тимчасового вікна. У заголовку цього вікна відобразити ім'я обраної константи.

Варіант 12

Тип елемента керування: Listbox. Мінімальне число пунктів меню – 3, максимальне число пунктів меню – 14.

Діалогова панель містить два списки пунктів меню, кнопки <<, >>, ОК. і "Назад". Після натискання кнопки << (або кнопки >>) обрані в правому (або в лівому) списку рядка переслати в лівий (або в правий). При натисканні кнопки "Назад" скасувати останній перенос. Після натискання кнопки ОК завершити роботу.

Варіант 13

Тип елемента керування: Combobox. Мінімальне число пунктів меню – 4, максимальне число пунктів меню – 12.

Перша сторінка блокнота містить назви книг (задати самостійно в кількості не менш 8), друга – назви розділів обраної книги (не менш 2), назви параграфів обраної глави. Усі сторінки містять вікно уведення й кнопки "Додати", "Видалити" й "Властивості". Після натискання кнопки "Додати" уміст вікна введення додається в список. Після натискання кнопки "Видалити" видаляється обрана назва зі списку. Після вибору рядка в списку й натискання кнопки "Властивості" розкрити наступну сторінку. Передбачити кнопки навігації між сторінками в обох напрямках.

Варіант 14

Тип елемента керування: Listbox. Мінімальне число пунктів меню – 5, максимальне число пунктів меню – 13.

Створити діалогове вікно для відображення номера телефону, що набирає. Номер набирати за допомогою розташованих на поверхні панелі кнопок від "0" до "9". Якщо перша цифра дорівнює "8", то забороняти на кілька секунд уведення даних.

Варіант 15

Тип елемента керування: Combobox. Мінімальне число пунктів меню – 6, максимальне число пунктів меню – 16.

Діалогова панель містить вікно уведення, два списки й кнопки "Вліво", "Вправо", "Готово" й "Назад". Після натискання кнопки "Вліво" (або кнопки "Вправо") уміст вікна введення або обраний у правому (або в лівому) списку рядок переслати в лівий (або в правий) список. Роботу завершити після натискання кнопки "Готово". При натисканні кнопки "Назад" скасувати останній перенос.

Варіант 16

Тип елемента керування: Listbox. Мінімальне число пунктів меню – 7, максимальне число пунктів меню – 15.

Діалогова панель містить список студентів групи й список їхнього рейтингу (під рейтингом мається на увазі середній бал попередньої сесії). Після вибору студента на панелі намалювати прямокутник, висота якого пропорційна рейтингу студента. Нижче прямокутника вивести ім'я студента, вище – рейтинг.

Варіант 17

Тип елемента керування: Combobox. Мінімальне число пунктів меню – 6, максимальне число пунктів меню – 12.

При виборі рядка меню "Установити пароль" створити панель, де за допомогою клавіатури ввести ім'я користувача й пароль, потім нажати кнопку ОК. Уведення пароля відображати символом "(". Після цього створити іншу діалогову панель для підтвердження введених даних. Після підтвердження даних завершити роботу панелі й відобразити введені дані.

Варіант 18

Тип елемента керування: Listbox. Мінімальне число пунктів меню – 3, максимальне число пунктів меню – 14.

Створити діалогове вікно завдання тексту виведеного рядка, кольори букв, кольори тіла тексту. Після вибору атрибутів виведення відобразити зазначений рядок із цими атрибутами в клієнтській області вікна.

Варіант 19

Тип елемента керування: Combobox. Мінімальне число пунктів меню – 4, максимальне число пунктів меню – 13.

Створити два списки. Між ними розташувати кнопки "Перемістити". Лівий список повинен бути споконвічно заповнений списком товарів

народного споживання. Якщо вибрати рядок і нажати одну із кнопок, то обраний рядок повинен переміститися зліва направо або навпаки. Передбачити відхилення від проведених дій.

Варіант 20

Тип елемента керування: Combobox. Мінімальне число пунктів меню – 5, максимальне число пунктів меню – 11.

Перша сторінка блокнота містить назви книг, друга – назви глав обраної книги, третя – назви параграфів обраної глави. Усі сторінки містять вікно введення й кнопки "Додати" й "Готово". Після натискання кнопки "Додати" уміст вікна введення додати в список. Після вибору рядка в списку й натискання кнопки "Готово" розкрити наступну сторінку. Якщо більше сторінок немає, завершити роботу блокнота.

Контрольні запитання до завдання 1

1. Перелічіть основні види меню й укажіть розходження між ними.
2. Чим відрізняються плаваючі меню від інших видів меню?
3. Яким чином можна послати повідомлення WM_COMMAND з ідентифікатором команди, якщо пов'язаний із цією командою рядок заблокований?
4. Чим відрізняється генерація команди за допомогою акселератора від генерації команди шляхом вибору рядка меню?
5. У якому порядку потрібно описати основні об'єкти додатка, якщо таблиця акселераторів змінюється при обробці повідомлень?
6. У чому перевага віртуального коду в акселераторах?
7. Яким чином батьківське вікно ідентифікує повідомлення від органів керування?
8. Які способи передачі повідомлень між вікнами існують й у чому вони відрізняються?
9. Що відрізняє стан кнопок панелі інструментів від стану рядків меню?
10. Якою константою визначають стиль вікна панелі інструментів, якщо всі кнопки панелі повинні посилати повідомлення, що сповіщають?

Контрольні запитання до завдання 2

1. Які основні етапи створення органів керування?
2. У чому перевага використання комбінованого списку порівняно зі звичайним списком?

3. Чим відрізняється функція вікна діалогового вікна від функцій звичайних вікон?

4. Які дії й у якій послідовності виконують для створення немодального діалогового вікна?

5. Як створюються стандартні діалогові вікна?

6. Що таке нотифікаційні повідомлення?

7. Що буде, якщо при створенні комбінованого списку іншим аргументом указати на деякий рядок?

8. Які органи керування не використовують ідентифікатори?

9. Які органи керування і в якому випадку не використовують дескриптори вікон?

10. У чому перевага використання комбінованого списку порівняно зі звичайним списком?

Лабораторна робота №8
Розробка програм на керованому C++

Мета лабораторної роботи – придбання практичних навичок роботи з елементами керованого C++.

Перед виконанням лабораторної роботи студент повинен знати основи застосування масивів і структур у керованому C++.

Після виконання лабораторної роботи студент повинен уміти розробляти прості програми з використанням контейнерів даних мовою C++/CLI.

Короткі теоретичні відомості й історичний аспект

Середовище .NET Framework

.NET Framework — центральна частина Visual C++ 2005, як і всіх інших засобів розробки .NET компанії Microsoft. Середовище .NET Framework складається із двох елементів: загальномовного виконуючого середовища (Common Language Runtime – CLR), у якому виконуються ваші програми, і набору бібліотек, називаних бібліотеками класів .NET Framework. Бібліотека класів .NET Framework забезпечує функціональну підтримку, що необхідна вашому коду при виконанні під керуванням CLR, незалежно від застосовуваної мови програмування, тому програми .NET, написані на C++,C# або будь-якій іншій мові, що підтримує .NET Framework, використовують ті самі бібліотеки .NET.

Існують два принципово різні види додатків C++, які можна розробляти в Visual C++ 2005. Ви можете писати додатки, які виконуються на вашому комп'ютері як "рідні" (native). Ці програми будемо називати рідними програмами C++. Такі програми пишуться на версії мови C++, визначеної стандартом ISO/ANSI. Ви також можете розробляти програми, що виконуються під керуванням CLR і реалізовані за допомогою розширеної версії C++, що зветься C++/CLI. Ці програми ми будемо називати програмами CLR, або програмами C++/CLI.

Загальномовне виконуюче середовище (CLR)

CLR – це стандартизоване середовище виконання програм, написаних на широкому діапазоні високорівневих мов, включаючи Visual Basic, C# і, зрозуміло, C++. Специфікації CLR у цей час убудовані в стандарт ECMA (European Association for Standardizing Information and Computer Systems – Європейська асоціація зі стандартизації інформаційних й обчислювальних систем) інфраструктури загальної мови (Common Language Infrastructure – CLI) – ECMA-335, а також в аналогічний стандарт ISO – ISO/IEC 23271, тому CLR є реалізацією цього стандарту. Ви бачите, чому C++ для CLR називається C++/CLI – це C++ для CLI, тому досить імовірно, що незабаром з'являться компілятори C++/CLI для інших операційних систем, які реалізують CLI.

CLI – це, по суті, специфікація середовища віртуальної машини, що дозволяє додаткам, написаним на різноманітних високорівневих мовах програмування, виконуватися в різних системах без зміни й перекомпіляції оригінального вихідного коду. CLI специфікує – стандарт проміжної мови віртуальної машини, у який компілюється вихідний код високорівневої мови програмування. B.NET Framework – ця проміжна мова називається Microsoft Intermediate Language (MSIL). Код цієї проміжної мови в остаточному підсумку при виконанні програми перетворюється на машинний код за допомогою оперативного компілятора (just-in-time – JIT). Звичайно, код проміжною мовою CLI може функціонувати тільки в середовищі, для якого існує реалізація CLI.

CLI також визначає загальний набір типів даних, називаний загальною системою типів (Common Type System – CTS), що повинен використовуватися програмами, написаними на будь-якій мові, орієнтованими на реалізацію CLI. CTS специфікує те, як застосовуються типи даних усередині CLR, і містить у собі набір визначених типів. Ви

можете також визначати власні типи даних, але їхнє визначення повинне підкорятися ряду правил, щоб вони були погодженими з CLR. Наявність стандартизованої системи типів для подання даних дозволяє компонентам, написаним на різних мовах, обробляти дані уніфікованим способом і забезпечує можливість інтеграції компонентів, написаних на різних мовах, в один додаток.

Код C++, виконуваний під керуванням CLR, називається керованим C++, тому що дані й код перебувають під контролем CLR. У програмах CLR звільнення пам'яті, динамічно виділеної для розміщення даних, здійснюється автоматично, що дозволяє виключити головне джерело помилок "рідних" додатків C++. Код C++, що виконується поза CLR, іноді називається некерованим C++, оскільки CLR у його виконанні не бере участь. У некерованому C++ ви повинні самостійно піклуватися про виділення й очищення пам'яті під час виконання програми, і вам доведеться самостійно забезпечувати безпеку, що убудована в CLR. Ми також будемо називати некерований C++ рідним C++, тому що він компілюється безпосередньо в рідний машинний код.

На рис. 8.1 показані основні варіанти вибору, які у вас є при розробці додатків C++.

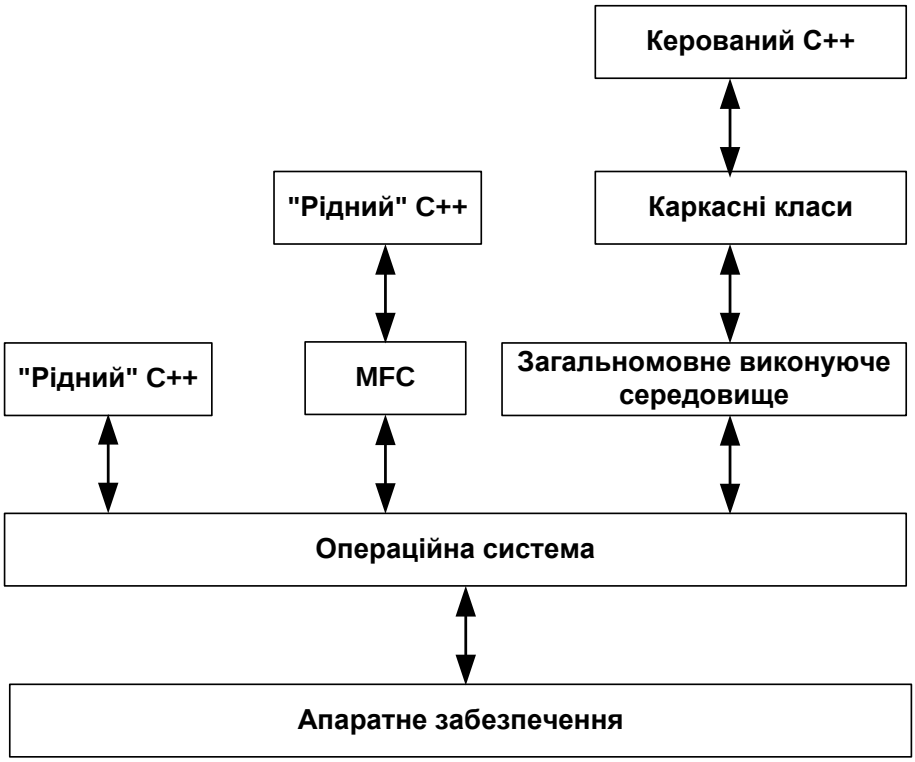

Рис. 8.1. **Основні варіанти вибору, доступні при розробці додатків на C++**

C++/CLI – інший привселюдно доступний проект підтримки CLI-програмування мовою C++. Перша спроба була представлена фірмою Microsoft як Managed Extensions (керовані розширення до C++), неофіційно вона називалася Managed C++ ("Керований C++"). Ця мова була представлена у двох випусках Visual C++ (2002 й 2003) і продовжує підтримуватися як застарілий режим у Visual C++ 2005.

Синтаксис, прийнятий у Visual Studio .NET 2002, був максимально наближений до існуючого стандарту C++, тобто до Стандарту ISO C++ (ISO C++ Standard). Відповідно до цього стандарту будь-які розширення мови повинні були відповідати правилам для мовних розширень. Крім інших обмежень, це означало, що ключові слова повинні були починатися з подвійного підкреслення (__). Таким чином, C++ з керованими розширеннями мав настільки незграбний синтаксис, що навряд чи його можна було вважати відповідним CLR.

Група C++ в Microsoft зрозуміла, що для того щоб зробити програмування на C++ приємним й естетично зробленим, а також використати всі переваги CLR, потрібно змінити синтаксис. І це означало, що потрібно зробити радикальний крок – відступити від Стандарту ISO на мову C++.

Однак Microsoft вирішила змінити стандарти, і якщо довелося відступити від Стандарту ISO на C++, то замість того щоб винайти "нестандартну" мову, було вирішено ввести новий стандарт. Так народився стандарт C++/CLI, який і був реалізований у Visual C++ 2005.

Використовуючи C++/CLI можна створювати керовані код і дані. Однак у цей час C++ є єдиною з усіх мов .NET, за допомогою якої можна створювати також і некеровані код і дані. Фактично, керовані й некеровані код і дані на C++ можуть бути визначені в тому самому вихідному файлі, і до деякої міри ці два світи можуть взаємодіяти. Хоча використання керованих коду й даних має багато переваг, воно може призвести до зниження продуктивності й втрати гнучкості. Тому в багатьох випадках C++ виявляється кращим вибором для створення програм. Іншою причиною вибору з усіх мов .NET саме C++ може бути бажання вдосконалювати знання C++ й існуючі напрацювання на цій мові.

Використання розширень керованого C++

При розробці керованого коду на Visual C++ використовуються кілька нових ключових слів, а розширення компілятора C++, що дозволяє створювати додатки для .NET, викликається за допомогою параметра /CLR (Компіляція для виконання в загальномовному середовищі). Цей параметр указує компіляторові, що в кінцевому файлі варто застосовувати

набір інструкцій проміжної мови IL, а не звичайний набір інструкцій процесора. Нові ключові слова використовуються при створенні керованого коду й не підтримуються при створенні звичайного некерованого коду. Хоча наявність або відсутність параметра /CLR (Компіляція для виконання в загальномовному середовищі) повністю визначає, чи буде компілятор генерувати керований (проміжною мовою IL) або некерований код, можна задавати режим компіляції для окремих частин програми. Це здійснюється за допомогою прагм #pragma:

#pragma managed // Наступний код компілюється як керований.

#pragma unmanaged // Наступний код компілюється як некерований.

Якщо задано параметр компілятора /CLR (Компіляція для виконання в загальномовному середовищі), то при відсутності директив #pragma вихідний код за замовчуванням компілюється як керований. При відсутності параметра /CLR (Компіляція для виконання в загальномовному середовищі) прагми #pragma компілятором ігноруються, а код компілюється як некерований.

```
#include "stdafx.h"
#include <stdio.h>
using namespace System;
#pragma unmanaged
void OutputUnmanage(char*s)
{
        printf("OutputUnmanage : %s \n",s);
}
#pragma managed
void OutputManage(String ^s)
{
        Console::WriteLine ("OutputManage   : {0}",s);
}
int main()
{
        char line[20]="We output this text";

    OutputUnmanage(line);
    OutputManage(gcnew String(line));
    return 0;
}
```

У наведеному прикладі функція OutputUnmanage, що виводить переданий як параметр текст на екран, компілюється як некерована. Якщо "забути" після неї вставити директиву #pragma managed, то компілятор видасть кілька повідомлень про помилки, оскільки далі в тексті програми використовуються конструкції, припустимі тільки в керованому C++, зокрема String^ й Console::WriteLine.

Програмування на C++/CLI

C++/CLI пропонує безліч розширень і додаткових можливостей, основні з яких наступні:

Усі фундаментальні типи даних ISO/ANSI можуть бути використані в програмах C++/CLI, але тут вони мають деякі додаткові властивості в певних контекстах.

C++/CLI надає власний механізм клавіатурного введення й виведення в командний рядок консольних програм.

C++/CLI надає операцію safe_cast, яка гарантує, що операція приведення приведе до генерації коду, що перевіряє.

C++/CLI надає альтернативну можливість перерахування на основі класів і забезпечує більшу гнучкість, чим оголошення enum з ISO/ANSI C++.

Специфіка C++/CLI: фундаментальні типи даних

Фундаментальні типи ISO/ANSI C++ можна використати у своїх програмах C++/CLI, і з арифметичними операціями вони працюють так само, як й у рідному C++. Крім того, в C++/CLI визначені два додаткових цілочислових типи займають по 8 байтів. Діапазони їхніх значень наступні:

long long від -9 223 372 036 854 775 808 до
9 223 372 036 854 775 807
unsigned long long від 0 до 18 446 744 073 709 551 615

Щоб специфікувати літерали типу long long, необхідно додавати LL або ll до цілого значення. Наприклад:

long long big = 123456789LL;

Літерал типу unsigned long long указується додаванням ULL або ull до цілого значення:

unsigned long long huge = 999999999999999ULL;

Хоча всі операції з фундаментальними типами, що ви бачили раніше, працюють аналогічним чином з C++/CLI, імена фундаментальних

типів у програмах C++/ CLI мають інший зміст і надають додаткові можливості в деяких ситуаціях. Фундаментальний тип у програмі C++/CLI – це клас типу значення, і може поводитися як звичайне значення або як об'єкт, якщо обставини того вимагають.

Усередині мови C++/CLI кожен фундаментальний тип ISO/ANSI відображається на клас типу значення, визначений у просторі імен System. Тобто в програмі на C++/CLI імена фундаментальних типів є скороченнями для асоційованих з ними класових типів. Це дозволяє трактувати значення фундаментального типу як прості значення або, при необхідності, як автоматично перетворений об'єкт асоційованого типу класу. Фундаментальні типи, обсяг займаної ними пам'яті й відповідні їм типи класів показані в табл. 8.1.

Таблиця 8.1

Фундаментальні типи C++/CLI

Фундаментальний тип	Розмір (у байтах)	Клас значень CLI
bool	1	System: -.Boolean
char	1	System::Sbyte
signed char	1	System::Sbyte
unsigned char	1	System::Byte
short	2	System::Intl6
unsigned short	2	System::Uintl6
int	4	System::Int32
unsigned int	4	System::Uint32
long	4	System::Int32
unsigned long	4	System::Uint32
long long	8	System::Int64
unsigned long long	8	System::UInt64
float	4	System:-.Single
double	8	System::Double
long double	8	System:-.Double
wchar_t	2	System::Char

Оскільки імена фундаментальних типів ISO/ANSI C++ служать псевдонімами для імен класів у програмі C++/CLI, у принципі можна використати в коді C++/CLI і ті, і інших. Наприклад, ви вже знаєте, як записувати оператори, що створюють цілі змінною й змінні із плаваючою крапкою:

```
int count = 10; double value = 2.5;
```

Ви можете використати імена класів, що відповідають фундаментальним типам, і компілювати програму без яких-небудь проблем:

System::Int32 count = 10; System::Double value = 2.5;

Хоча це зовсім законно, все-таки у своєму коді ви повинні використати імена фундаментальних типів, такі, як int й double, замість імен класів System::Int32 й System::Double. Причина в тому, що відображення імен фундаментальних типів на імена класів і назад – це функціональність компілятора Visual C++. Інші компілятори не обов'язково реалізують такі перетворення.

Створення простого керованого консольного додатка

Для створення консольного додатка на керованому C++ варто виконати наступну послідовність операцій:

1. Запустіть на виконання Visual Studio. NET (2005 або більш нову)
2. Виберіть послідовно пункти меню File – New Project (Файл – Створити новий проект) для того, щоб відкрити діалог New Project (Створення проекту). Для активізації вікна створення нового проекту (рис. 8.2) можна використати й комбінацію клавіш Ctrl + Shift + N.

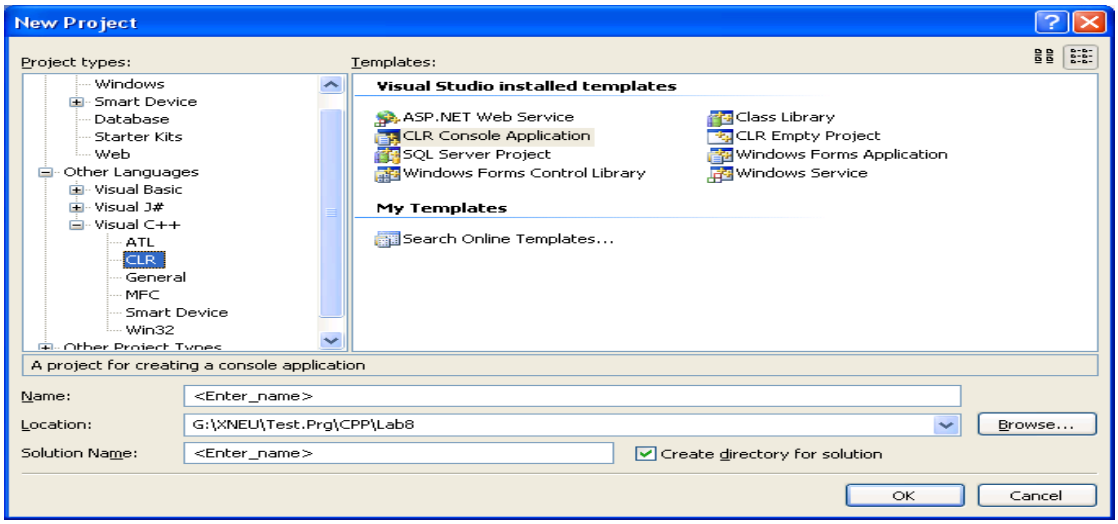

Рис. 8.2. **Вікно створення проекту**

3. У вікні (рис. 8.2), що з'явилося, в Project types вибрати мову Visual C++, а в Templates – CLR Console Application.
4. Внизу вікна вкажіть місце розташування проекту (Location) і його ім'я (Name) і натисніть ОК (рис. 8.3).

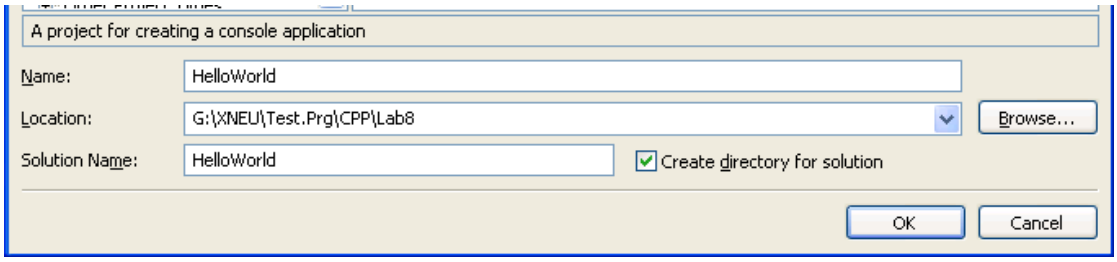

Рис. 8.3. **Місце розташування та ім'я проекту**

У результаті виконаних дій буде створений проект, що виводить на екран один рядок "Hello World":

```
// HelloWorld.cpp : main project file.
#include "stdafx.h"
using namespace System;
int main(array<System::String ^> ^args)
{
   Console::WriteLine(L"Hello World");
   return 0;       }
```

Отриманий проект можна запустити на виконання, наприклад, за натисканням Ctrl+F5, і одержати наступний результат (рис. 8.4).

Рис. 8.4. **Результат роботи програми**

У випадку, якщо програма не використовує параметри, передані їй у командному рядку, то функцію main можна переписати так:

```
int main()
{
 . . . . .
}
```

У наведеному прикладі можна побачити нову конструкцію

```
using namespace System;
```

яка описує (або говорять "повідомляє") так званий простір імен. Простори імен не застосовуються в бібліотеках, що підтримують MFC,

але стандартна бібліотека ANSI C++ і бібліотеки, що підтримують CLR й Windows Forms, використовують їх дуже інтенсивно.

Ви вже знаєте, що всі імена, використовувані в стандартній бібліотеці ISO/ANSI C++, визначені в просторі імен std. Це значить, що все імена, що зустрічаються в стандартній бібліотеці, мають додаткове кваліфікуюче ім'я — std, тому cout, наприклад — це насправді std: : cout.

Оголошення в попередньому прикладі простору імен System представляє простір імен C++, прямо відповідний простору імен .NET, що має ту ж назву.

Повна назва класу складається з назви простору імен, за яким ідуть дві двокрапки й назва класу, наприклад, System::Console. Хоча вираження using namespace, у попередньому прикладі не використовується, воно дозволяє використати короткі імена класів, наприклад, Console замість System::Console.

Символ 'L' перед рядком "Hello World" в операторі виведення

Console::WriteLine(L"Hello World");

означає, що це Unicode-рядок, тобто кожен символ займає 2 байти.

Стандартне введення-виведення

Клас System::Console забезпечує підтримку стандартного ввведення-виведення. Метод ReadLine класу System::Console зчитує уведений із клавіатури рядок як текстовий.

String^ line = Console::ReadLine ();

Цей оператор читає повний вхідний рядок тексту, завершений натисканням клавіші <Enter>. Змінна line має тип String^ і зберігає посилання на рядок, що виходить у результаті виконання функції Console::ReadLine(). Символ ^, указує, що це дескриптор (handle), що посилається на об'єкт типу String.

Прочитати окремий символ можна застосувавши для цього функцію Console::Read().

char ch = Console::Read();

Можна також прочитати натискання клавіші функцією Console::ReadKey ().

Функція Console::ReadKey () повертає натиснуту клавішу у вигляді об'єкта класу ConsoleKeyInfo, що становить клас типу значення, визначений у просторі імен System. Оператор читання натиснутої клавіші:

ConsoleKeyInfo keyPress = Console::ReadKey(true);

Аргумент true функції ReadKey () придушує відображення натиснутої клавіші в командному рядку. Аргумент false (або відсутність аргумента)

змушує функцію відображати символ натиснутої клавіші. Результат виконання функції зберігається в keyPress. Щоб ідентифікувати символ, що відповідає натиснутій клавіші (або клавішам), необхідно використати вираження keyPress.KeyChar. Таким чином, щоб вивести повідомлення, що показує символ натиснутої клавіші, потрібно скористатися наступним оператором:

Console::WriteLine(L"Натиснута клавіша відповідає символу: {0}", keyPress.KeyChar);

За допомогою методів Write й WriteLine класу System::Console на консоль виводиться текстовий рядок, і, говорячи про метод WriteLine, також символ нового рядка. Простіше всього уведення з консолі виконується шляхом зчитування в об'єкт string (Рядок) з наступним перетворенням у необхідний тип даних. Щоб виконати це перетворення можна використати методи Toxxx класу System::Convert.

У наступному прикладі такий метод використовується для введення з консолі температури в градусах Фаренгейта, перетворення текстового рядка в число, обчислення температури в градусах Цельсія й виведення на консоль значень температури в градусах Фаренгейта й Цельсія.

```
#include "stdafx.h"
using namespace System;
// Клас із методами для введення значень із консолі
value class InputWrapper // клас "збирача сміття" InputWrapper
{
public:
//  Уведення цілого числа
int getInt(String ^pprompt) // pprompt – підказка на екрані
{
 Console::Write(pprompt);          // Виведення на екран підказки
 String ^pbuf = Console::ReadLine(); // Читання рядка із клавіатури
 return Convert::ToInt32(pbuf);     // Перетворення рядок – ціле число
}
//  Уведення матеріального числа
double getDouble(String ^pprompt) // pprompt – підказка на екрані
{
 Console::Write(pprompt);               // Виведення на екран підказки
 String ^pbuf = Console::ReadLine(); // Читання рядка із клавіатури
```

```
        return Convert::ToDouble(pbuf);    //    Перетворення    строка    –
                                           //матеріальне число
      }
      //  Уведення десяткового числа
      Decimal getDecimal(String ^pprompt) // pprompt – підказка на екрані
      {
       Console::Write(pprompt);            // Виведення на екран підказки
       String ^pbuf = Console::ReadLine(); // Читання рядка із клавіатури
       return Convert::ToDecimal(pbuf);  //    Перетворення    строка    –
                                           //матеріальне число
      }
      //  Уведення рядка
      String ^getString(String ^pprompt) // pprompt – підказка на екрані
      {
       Console::Write(pprompt);    // Виведення на екран підказки
       String ^pbuf = Console::ReadLine();     // Читання рядка із клавіатури
       return pbuf;
      };
      }; // End of InputWrapper

      // Головна програма
      int main(array<System::String ^> ^args)
      {
        InputWrapper ^piw = gcnew InputWrapper;          // об'єкт класу
//InputWrapper
        int numTemp = piw->getInt("How many temp's? ") ; // Скільки
//вводимо температур?
        for (int i = 0; i < numTemp; i++){
         int fahr = piw->getInt("Temp. (Fahrenheit): "); // Фаренгейт
         int celsius = (fahr - 32) * 5/9;              // Цельсія
         Console::WriteLine ("Fahrenheit = {0}", fahr.ToString()); // Виведення
// (за Фаренгейтом)
         Console::WriteLine("Celsius = {0}", celsius);          //   Виведення (за
// Цельсієм)
        }
       return 0;
      }
```

Помітимо, що першим аргументом методу WriteLine є форматуюча строка.

Наприклад, при першому виклику методу WriteLine форматуючий рядок має вигляд:

"Fahrenheit={ 0} ",

де { 0} – заглушка, що вказує, що на це місце варто вставити другий аргумент WriteLine. Число, поміщене у фігурні дужки, визначає, який саме з наступних за форматуючим рядком аргументів варто вивести в зазначеному місці (природно, нумерація починається з нуля). У нашому прикладі це число 0, тому що за форматуючим рядком потрібен тільки один аргумент. Аргументи, що підставляють, можуть бути декількох типів, включаючи рядки або впаковані значення, що й показано в прикладі. Наведемо приклад роботи програми, у якому перетворення температур виробляється два рази (рис. 8.5):

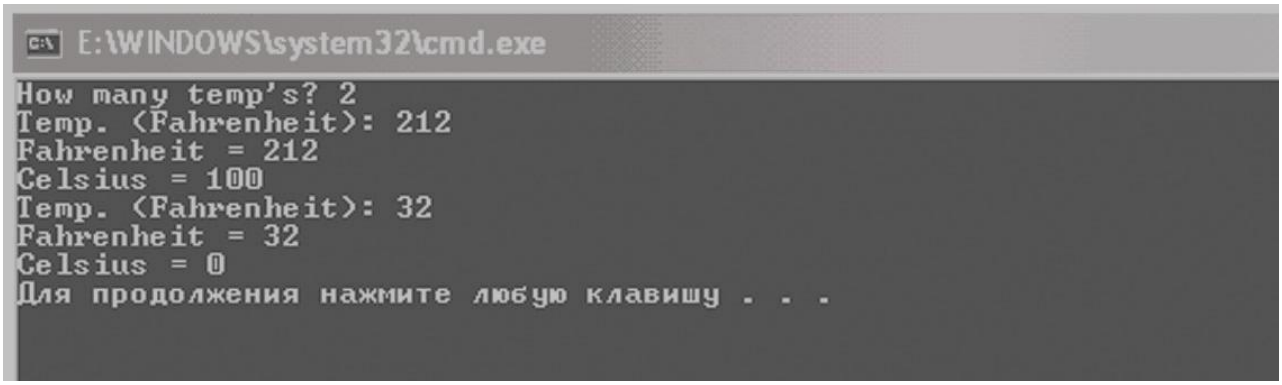

Рис. 8.5. **Результат роботи програми**

У наступній програмі продемонстровано, як виводити дані в деяких форматах за допомогою методу WriteLine. Для цього застосовуються коди форматування. Щоб одержати більш докладну інформацію про коди форматування, використовуваних у методі WriteLine (співпадаючих, до речі, з кодами для методу string::Format), варто звернутися до документації по .NET SDK.

```
#include "stdafx.h"
using namespace System;
void main(void)
{
Console::WriteLine("{0:C}\n{1:D}\n{2:E}\n{3:F}\n{4:G}\n{5:N}\n{6:X}",
        100, // поле валюти (currency)
        200, // десяткове число (decimal)
```

```
            300, // експонента (exponent)
            400, // з фіксованою крапкою (fixed point)
            500, // загальний (general)
            600, // число (number)
            700// шістнадцятирічне (hexadecimal)
        );
    }
```

Будучи запущеної на виконання, програма виводить на екран наступний результат (рис. 8.6):

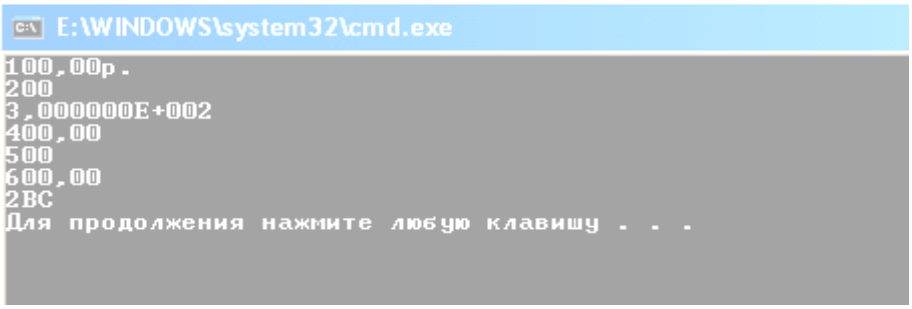

Рис. 8.6. **Результат роботи програми**

Таблиця 8.2

Часто використовувані специфікатори формату

Специфікатор формату	Опис
C або c	Виводить значення в грошовому форматі
D або d	Виводить ціле в десятковому виді. Якщо вказати більш високу точність, кількість десяткових знаків у числі, то воно буде доповнено нулями ліворуч
E або e	Виводить значення із плаваючою крапкою в науковій нотації, тобто з експонентою. Значення точності вказує кількість десяткових розрядів після крапки
F або f	Виводить значення із плаваючою крапкою як число з фіксованою крапкою у формі ±ddd.dd...
G або g	Виводить значення в найбільш компактному виді, залежно від його типу й зазначеної точності. Якщо точність не зазначена, приймається значення точності за замовчуванням
N або n	Виводить десяткове значення із плаваючою крапкою, використовуючи при необхідності роздільник – кому між групами за трьома розрядами
X або x	Виводить ціле в шістнадцятирічному виді. Шістнадцятирічні цифри виводяться у верхньому або нижньому регістрі залежно від того, зазначений X або x

Взагалі можна писати специфікацію формату у вигляді {n, w:Axx}, де n – значення індексу, що вказує номер аргумента, який випливає за форматним рядком, w – необов'язкова специфікація ширини поля, A – односимвольна специфікація формату значення, а xx – необов'язкове одно- або двозначне число, що задає точність виведення значення. Специфікація ширини поля – ціле зі знаком. Значення буде вирівняно вправо в поле w, якщо ширина позитивна, і вліво – якщо негативно. Якщо значення займає менше знаків, чим зазначено в w, то виведення доповнюється пробілами; якщо значення не міститься завширшки w, то специфікація ширини ігнорується. Ще один приклад:

Console::WriteLine(L"Пакетів:{0,3} Вага: {1,5:F2} фунтів.", packageCount, packageWeight);

Кількість пакетів виводиться в поле шириною 3 знаки, а вага – у поле шириною 5, тому в результаті одержимо:

Пакетів: 25 Вага: 7.50 фунтів.

Розглянемо ще один приклад найпростішої програми, що читає натискання клавіш і виводить інформацію на екран:

```
#include "stdafx.h"
using namespace System;

int main()
{
Console::WriteLine(L"Натисніть комбінацію клавіш — для виходу натисніть Esc");
ConsoleKeyInfo keyPress;
do
{
        keyPress = Console::ReadKey(true);
        Console::Write(L"Ви натиснули");
        if(safe_cast<int>(keyPress.Modifiers)>0)
        Console::Write(L" {0},", keyPress.Modifiers);
        Console::WriteLine (L" {0} що дає символ {1} ",
                        keyPress.Key, keyPress.KeyChar);
 }while(keyPress.Key != ConsoleKey::Escape);
return 0;
}
```

Програма виводить на екран наступний результат (рис. 8.7):

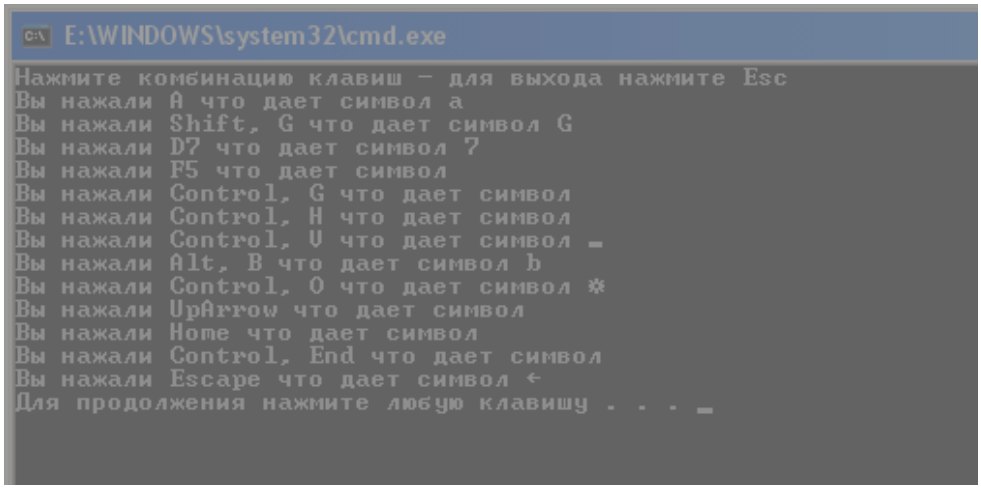

Рис. 8.7. **Результат роботи програми**

У розглянутому прикладі використовується операція safe_cast, що призначена для явних приведень типів у середовищі CLR. У більшості випадків ви можете без проблем використати static_cast для приведення одного типу до іншого в програмах C++/CLI, але оскільки є виключення, які приводять до повідомлень про помилки, краще все-таки використати saf ecast. Приведення safe_cast застосовується так само, як static_cast. Наприклад:

double valuel = 10.5; double value2 =15.5;

int whole_number = safe_cast<int>(valuel) + safe_cast<int>(value2);

Останній оператор приводить кожне зі значень типу double до типу int перед тим, як скласти їх і привласнити результат wholenumber.

Динамічне виділення пам'яті в програмах на C++/CLI

Динамічне виділення пам'яті в CLR працює інакше, і CLR підтримує свою власну "купу" пам'яті, що повністю незалежна від купи рідного C++. CLR автоматично очищає пам'ять, що виділена в купі CLR і необхідність у якій відпала, тому програмістові потрібно використати операцію delete у програмах, написаних для CLR. CLR може також час від часу впорядковувати пам'ять купи, щоб уникнути фрагментації. Таким чином, CLR виключає ймовірність витоків пам'яті і її фрагментацій. Керування очищенням купи, що передбачає CLR, описується як зборка сміття (сміття становить собою відкинуті змінні й об'єкти, а купа, якою управляє CLR, називається купою, що очищається збирачем сміття). Для виділення пам'яті в програмі C++/CLI необхідно використати операцію gcnew замість new, і префікс gc відбиває той факт, що пам'ять виділяєть-

ся в купі, що очищається, а не в рідній купі C++, де за все господарство програміст відповідає самостійно.

Збирач сміття CLR здатний видаляти об'єкти й звільняти пам'ять, що вони займали, коли необхідність у них відпадає. Виникає очевидне запитання: як може знати збирач сміття про те, коли об'єкт купи більше не потрібний? Відповідь досить проста. CLR відслідковує кожну змінну, котра посилається на кожен об'єкт купи, і коли не залишається змінна, утримуюча адреса даного об'єкта, це значить, що на нього неможливо послатися в програмі, а тому він може бути вилучений.

Оскільки процес зборки сміття може містити в собі стискання області пам'яті купи для виключення фрагментації невикористаних блоків, адреса елемента даних, що ви зберігаєте в купі, може змінитися. Отже, ви не можете застосовувати звичайні рідні покажчики C++, оскільки місце розташування даних, що вказують, змінюється, і такі покажчики стають недійсними. Необхідний спосіб доступу до об'єктів у купі, що дозволяє обновляти адреси, коли збирач сміття переміщає в ній елементи даних. Ця можливість забезпечується двома способами: за допомогою дескрипторів, які відслідковують, що становлять деякі аналоги покажчиків з рідного C++, і за допомогою посилань, що відслідковують, – еквівалента рідних посилань C++ у програмах CLR.

Дескриптори, що відслідковують

Дескриптори, що відслідковують, (tracking handle) мають подібність із рідними покажчиками C++, однак є й істотні відмінності. Дескриптор зберігає адресу, і адреса, що у ньому втримується, автоматично обновляється збирачем сміття, якщо об'єкт, на який він посилається, переміщається під час стискання купи. Однак не можна застосовувати арифметику адрес до таких покажчиків, як це робиться з "рідними" покажчиками, крім того, не дозволяється приведення дескрипторів.

Усі об'єкти, створені в купі CLR, повинні забезпечуватися дескрипторами. Всі об'єкти посилальних типів класів зберігаються в купі й тому створені вами змінні, які посилаються на такі об'єкти, повинні бути дескрипторами. Наприклад, тип класу String – це посилальний тип класу, тому змінні, які посилаються на об'єкти String, повинні бути відслідковуючими дескрипторами. Пам'ять для типів класів значень за замовчуванням виділяється в стеці, але ви можете також розмістити їх у купі, використовуючи для цього операцію gcnew.

Оголошення дескрипторів, що відслідковують

Для специфікації дескриптора типу необхідно помістити символ : (часто називаний "капелюхом") слідом за ім'ям типу. Наприклад, ось так можна оголосити дескриптор, що відслідковує, по імені proverb, що може зберігати адресу об'єкта String:

```
String^ proverb;
```

Це визначає змінну proverb – дескриптор, що відслідковується як тип String^. Коли з'являється дескриптор, він автоматично ініціалізується нулем, тому ні на що не посилається. Для явної установки дескриптора в нуль служить ключове слово nullptr:

```
proverb = nullptr;   // Установити дескриптор в null
```

Зверніть увагу, що тут не можна використати 0, як це можна робити з рідними покажчиками. Якщо ініціалізувати дескриптор нулем, то значення 0 перетвориться в тип об'єкта, на який буде посилатися дескриптор, і в нього буде поміщена адреса цього нового об'єкта.

Звичайно, можна явно ініціалізувати дескриптор при його оголошенні. Ось інший оператор, що визначає дескриптор об'єкта String:

```
String^ saying = L"Деякий довгий рядок";
```

Цей оператор створює в купі об'єкт String, що містить рядок праворуч від операції присвоювання; адреса нового об'єкта міститься в saying. Типом рядкового літерала тут є wchar_t*, а не String. Визначення класу String дає можливість такому літералу бути використаним для створення об'єкта типу String.

А ось так створюється дескриптор для типу значення:

```
int^ value = 99;
```

Цей оператор створює дескриптор value типу int^, і значення в купі, на якому він указує, ініціалізуються числом 99. Необхідно пам'ятати, що в цьому випадку створюється різновид покажчика, тому value не може брати участь в арифметичних операціях без його розіменування. Для розіменування дескриптора, що відслідковує, служить операція * – так само, як це робиться з покажчиками. Наприклад, нижче поданий оператор, що використовує значення, зазначене дескриптором, що відслідковує, в арифметичній операції:

```
int result = 2*(*value)+15;
```

Масиви CLR

Масиви CLR відрізняються від масивів рідного C++. Пам'ять для масиву CLR виділяється в керованій купі, але це ще не все. Масиви CLR

мають убудовану функціональність, якої немає у масивів рідного C++. Тип змінної масиву специфікується ключовим словом array. При цьому також необхідно вказати тип елементів масиву між кутовими дужками, що випливають за ключовим словом array, тому загальна форма змінної, яка посилається на одновимірний масив, array<тип_елемента>^. Оскільки масиви CLR створюються в купі, змінні масивів – це дес.криптори, що відслідковують завжди. Приклад оголошення змінної масиву:

array<int>^ data;

Змінна масиву data може зберігати посилання на одновимірний масив елементів типу int.

Ви можете створити масив CLR, використовуючи операцію gcnew, одночасно з оголошенням змінної масиву:

array<int>^ data = gcnew array<int> (100) ; // Створити масив для зберігання 100 цілих чисел

Цей оператор створює одновимірний масив за іменем data (зверніть увагу, що змінна масиву – дескриптор, що відслідковує, тому не можна забувати про символ "капелюха", що випливає за специфікацією типу елемента, укладеного в кутові дужки). Кількість елементів указується в круглих дужках, що випливають за специфікацією типу масиву, також укладена в кутові дужки. Тобто в цьому випадку це буде масив, що містить 100 елементів типу int.

Як й у масивах рідного C++, елементи масивів CLR індексуються, починаючи з нуля, тому можна встановити значення елементів масиву data у такий спосіб:

for(int i = 0 ; i<100 ; i++) data[i] = 2* (i+1);

Цикл установлює значення елементів рівними 2, 4, 6, і т. д. до 200.

У попередньому циклі кількість оброблюваних елементів у циклі задана як літеральне значення. Але краще використати властивість Length масиву, що зберігає число елементів масиву:

for(int i = 0 ; i < data->Length ; i++)

 data[i] = 2*(i+l);

Щоб звернутися до властивості Length, тут використовується операція ->, тому що data – дескриптор, що відслідковує, який працює подібно покажчику. Ця властивість зберігає кількість значень як 32-бітне ціле. Можна одержати довжину масиву і як 64-бітне ціле через властивість LongLength.

Змінна масиву може зберігати адресу будь-якого масиву того ж рангу (ранг – це кількість вимірів, що у випадку масиву data дорівнює 1) і типу елементів. Наприклад:

data = gcnew array<int> (45) ;

Цей оператор створює новий одновимірний масив з 45 елементів типу int і зберігає його адресу в data. Вихідний масив відкидається.

Також можна створити масив, указати набір початкових значень елементів:

array<double>^ samples = { 3.4, 2.3, 6.8, 1.2, 5.5, 4.9. 7.4, 1.6};

Розмір масиву визначається кількістю ініціюючих значень, укладених у фігурні дужки, у цьому випадку – 8, причому значення привласнюються елементам послідовно.

Звичайно, елементи масиву можуть бути будь-якого типу, тому дуже легко можна створити масив рядків:

array<String>^ names = { "Jack", "Jane", "Joe", "Jessica", "Jim", "Joanna"};

Елементи цього масиву ініціалізуються рядками, які укладені у фігурні дужки, і кількість цих рядків визначає число елементів у масиві. Об'єкти String розміщаються в купі CLR, тому типом елементів цього масиву є тип дескрипторів, що відслідковують – String^.

Якщо ви оголосите змінну масиву, не ініціалізуючи її, то повинні потім явно створити масив, якщо хочете використати список ініціюючих значень. Наприклад:

array<String^>^ names; // Оголошення змінної масиву

names = gcnew array<String>{ "Jack", "Jane", "Joe", "Jessica", "Jim", "Joanna"};

Другий оператор створює масив й ініціалізує його рядками у фігурних дужках. Без явного виклику gcnew цей оператор не скомпілюється.

Ви можете використати функцію Clear (), що визначена в класі Array для установки будь-якої послідовності числових елементів масиву в нульове значення, наприклад:

// Установити всі елементи в нуль.

Array::Clear(samples, 0, samples->Length);

Перший аргумент Clear () – масив, що повинен бути очищений, другий – індекс першого елемента, що очищає, а третій – кількість елементів, що підлягають очищенню. Таким чином, у цьому прикладі

всім елементам масиву samples привласнюється значення 0.0. Якщо ви застосуєте функцію Clear () до масиву дескрипторів, таких, як String^, то елементи встановлюються в null, а якщо застосувати її до масиву об'єктів bool, то вони одержать значення false.

Багатовимірні масиви

У програмах на C++/CLI можна створювати масиви із двома й більше вимірами; максимальна кількість вимірів масиву – 32, чого цілком достатньо в більшості випадків. Кількість вимірів масиву вказується в кутових дужках відразу після типу елемента й відокремлюється від нього комою. За замовчуванням масив має один вимір, от чому ми не специфікували його дотепер. Ось так можна створити двовимірний масив цілочислених елементів:

```
array<int,  2>^ values = gcnew array<int, 2>(4, 5);
```

Цей оператор створює двовимірний масив із чотирьох рядків і п'яти стовпців, так що всього він уміщає 20 елементів. Щоб звернутися до елемента багатовимірного масиву, ви специфікуєте набір значень індексу – по одному для кожного виміру; вони вказуються між квадратними дужками слідом за ім'ям масиву й розділяються комами. Так можна встановити значення елементів двовимірного масиву цілих чисел:

```
int nrows = 4; int ncols = 5;
array<int, 2>^ values = gcnew array<int, 2>(nrows, ncols);
for(int i = 0;   i<nrows;   i++)
  for (int j =0;  j<ncols;   j++)
     values [i,j]   =   (i+l)*(j+l);
```

Вкладений цикл проходить по всіх елементах масиву. Зовнішній цикл проходить по рядках, а внутрішній – по кожному елементу в поточному рядку. Значення кожного елемента встановлюється рівним значенню, отриманому в результаті обчислення вираження (i+l)*(j + l), тому елементи першого рядка будуть установлені в 1, 2, 3, 4, 5, другого рядка – 2, 4, 6, 8, 10 і т. д., аж до останнього рядка, елементи якого будуть рівні 4, 6, 12, 16, 20.

Нотація доступу до елементів двовимірного масиву відрізняється від нотації, використовуваної з масивами "рідного" C++. Це не випадково. Масив C++/CLI не є масивом, подібним до масивів рідного C++. Розмірність масиву називається його рангом (rank), тому ранг масиву values з попереднього прикладу дорівнює 2. На відміну від цього,

масиви рідного C++ у дійсності завжди мають ранг 1, оскільки рідні масиви C++ із двома й більше вимірами в дійсності є масивами масивів.

З раніше наведених прикладів було видно, що тип класу String, визначений у просторі імен System, представляє рядок в C++/CLI (фактично рядки складаються з їхніх символів Unicode). Виражаючись точніше, він представляє рядок, що складається з послідовності символів типу System::Char. Об'єкти класу String надають величезний обсяг потужної функціональності, що значно полегшує обробку рядків.

Об'єкт String можна створити в такий спосіб:

System::String^ saying = L"Приклад створення рядка";

Змінна saying – дескриптор, що відслідковує, посилається на об'єкт String, ініціалізований рядком, що перебуває праворуч від знака =. Необхідно завжди використовувати дескриптори, що відслідковують, для збереження посилань на об'єкти String. Представлений тут рядковий літерал є рядком, що складається із широких символів, оскільки постачено префіксом L. Якщо пропустити префікс L, то одержимо рядковий літерал, що складається з 8-бітових символів, а так компілятор гарантує, що він буде перетворений у рядок широких символів.

У програмі можна звертатися до індивідуальних символів у рядку, використовуючи нотацію масивів, і перший символ рядка має індекс 0. Наприклад, для виведення третього символу рядка saying, використаємо наступний оператор:

Console::WriteLine("Третій символ у рядку: {0}", saying[2]);

Однак, таким чином, тобто звертаючись до них за індексом, можна лише читати окремі символи рядка, обновляти рядок подібним чином не вдасться. Рядкові об'єкти є незмінними (immutable), а тому не можуть бути модифіковані.

Об'єднання рядків

Для об'єднання рядків ви можете використати операцію +, щоб сформувати новий об'єкт String. Наприклад:

String^ name1 = L"Пат";

String^ name2 = L"Паташон";

String^ name3 = name1 + L" й " + name2;

Після виконання цих операторів name3 містить рядок "Пат і Паташон". Зверніть увагу на використання операції + для об'єднання об'єктів String з рядковими літералами. Також ви можете поєднувати

об'єкти String із числовими значеннями, або значеннями bool, при цьому одержуючи автоматичне їхнє перетворення в рядковий вид. Наступні оператори ілюструють сказане.

```
String^  str = L"Значення:   ";
String^  str1 = str +2.5;//У результаті – новий рядок: "Значення:   2.5".
String^  str2 = str + 25;// У результаті – новий рядок: "Значення:   25".
String^   str3 = str + true; // У результаті – новий рядок:   "Значення:
//True".
```

Можна також з'єднувати String із символом, але результат при цьому залежить від типу символу:

```
char ch =  'Z';
wchar_t wen =  'Z'
String str4 = str + ch;    // У результаті – новий рядок "Значення:   90".
String str5 = str + wch;  // У результаті – новий рядок "Значення:   Z".
```

У коментарях відбиті результати цих операцій. Символ типу char трактується як числове значення, так що ви одержуєте код символу, приєднаний до рядка. Символ wchar_t має той же тип, що й символи в об'єкті String (тип Char), тому символ додається до рядка.

Цикл foreach

При обробці масивів і рядків дуже часто використовують оператори циклів. Усі оператори циклів, які застосовуються в мові ISO/ANSI C++, також використовуються й у C++/CLI. Але мова C++/CLI пропонує ще один розкішний тип циклу, називаний for each. Він призначений спеціально для ітерації за об'єктами, що належать до певного набору. Застосуємо цикл for each для ітерації за символами рядка.

Використання циклу for each для доступу до кожного символу в String.

Створіть новий проект консольної програми CLR і модифікуйте код у такий спосіб:

```
// Аналіз рядка за допомогою циклу for each
#include "stdafx.h"
using namespace System;
int main()
{
 int vowels = 0;
 int consonants = 0;
 String^ proverb = L"This is a simple text for \"for each\" cycle" ;
```

```
for each (wchar_t ch in proverb)
{
  if(Char::IsLetter(ch))
    {
     ch = Char::ToLower(ch) ; // Перетворити в нижній регістр
     switch(ch)
     {
          case 'a':
          case 'e':
          case 'i':
          case 'o':
          case 'u':
               ++vowels;
                break;
          default:
                ++consonants;
                break;
       }
  }
}
Console::WriteLine(proverb);
Console::WriteLine(L"Текстовий рядок містить {0} голосних й {1} приголосних.",vowels, consonants);
return 0;
}
```

Результат роботи програми буде наступний (рис. 8.8):

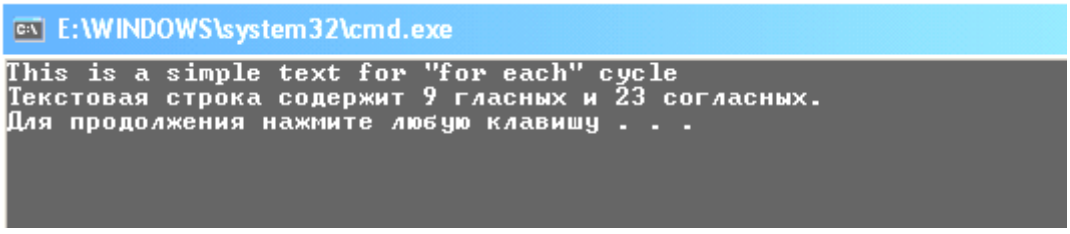

Рис. 8.8. **Результат роботи програми**

Можна також виконати ітерацію за всіма елементами масиву, скориставшись циклом for each:

```
array<int>~ values = {  3,   5,   6,   8,   б};
```

```
for each(int item in values)
{
item = 2*item + 1;
Console::Write("{0,5}",item);
}
```

Усередині циклу змінна item посилається на кожен елемент масиву по черзі. Перший оператор у тілі циклу заміняє значення поточного елемента подвоєним старим його значенням плюс одиниця. Другий оператор циклу виводить нове значення, вирівнюючи його вправо в поле шириною п'ять символів, тому в результаті виходить наступний виведення:

```
7    11    13    17    13
```

Розглянемо ще кілька прикладів використання рядків і масивів мовою C++/CLI.

Як уже було сказано раніше, об'єкт String – незмінний, тобто будучи один раз ініціалізованим, він не може бути змінений. Клас String містить методи, які можна використати для зміни об'єкта String, такі, як Insert (Вставка), Replace (Заміна) і PadLeft. Однак, у дійсності, зазначені методи ніколи не змінюють вихідний об'єкт. Замість цього вони повертають новий об'єкт String, що містить змінений текст. Якщо ми хочемо одержати можливість змінювати вихідні дані, то варто використати клас StringBuilder, а не на сам клас String. У наступному прикладі показано, що метод Replace (Заміна) не впливає на вміст вихідного об'єкта String (Рядок), але змінює вміст об'єкта StringBuilder:

```
#include "stdafx.h"
using namespace System;
using namespace System::Text; // для StringBuilder
void main(void)
{
  Console::WriteLine("String is immutable:"); // ("Рядок є незмінним: ") ;
  String ^ps1 = "Hello World";              // Рядок "Привіт, Світ"
  String ^ps2 = ps1->Replace('H','J');      // Заміна
  Console::WriteLine(ps1) ;
  Console::WriteLine(ps2);
  Console::WriteLine("StringBuilder can be modified:");
 // ("StringBuilder може змінюватися: ") ;
```

```
  StringBuilder ^psb1 = gcnew StringBuilder("Hello World");
// Привіт, Світ
  StringBuilder ^psb2 = psbl->Replace('H', 'J');          // Заміна
  Console::WriteLine(psbl);
  Console::WriteLine(psb2);
}
```

Результат роботи програми буде наступний (рис. 8.9):

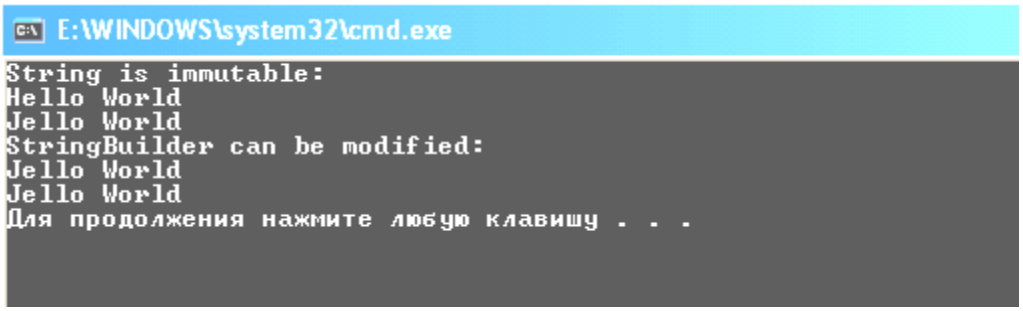

Рис. 8.9. **Результат роботи програми**

Метод ToString забезпечує подання об'єкта String для будь-якого керованого типу даних. Хоча метод ToString не є автоматично доступним для некерованих класів, він доступний для впакованих значимих й упакованих примітивних типів, таких, як int або float (із плаваючою крапкою).

Метод ToString найбільш часто використовується для виведення інформації, а також при налагодженні, і створювані керовані класи звичайно заміняють ToString так, щоб він повертав визначену розроблювачем, легку для читання інформацію про об'єкт. Метод Object::ToString просто повертає повне ім'я класу даного об'єкта і його реалізація (не особливо корисна) доступна через спадкування будь-якому керованому типу. Наступний приклад демонструє деякі аспекти роботи методу ToString:

```
#include "stdafx.h"
using namespace System;
// клас збирача сміття ClassWithToString
value class ClassWithToString
{
 public:
 virtual String ^ToString() override     // перевантаження методу
```

```
 {
  // повернути новий Рядок ("SomeClass – скасування");
  return gcnew String("SomeClass – override");
 }
};
value class ClassNoToString // клас збирача сміття ClassNoToString
{
};
void main(void)
{
  int i = 3;
  Console::WriteLine(i.ToString()); // перевантаження String^
  Console::WriteLine(i);            // перевантаження int
  ClassWithToString ^psc = gcnew ClassWithToString;
  Console::WriteLine(psc->ToString()); // перевантаження String^
  Console::WriteLine(psc);            // перевантаження Object^
  ClassNoToString ^psoc = gcnew ClassNoToString;
  Console::WriteLine(psoc->ToString());  // перевантаження String^
  Console::WriteLine(psoc);             // перевантаження Object^
  array<int>^ agc= gcnew array<int>(5);  // масив збирача сміття
  Console::WriteLine(agc->ToString());   // перевантаження String
  Console::WriteLine(agc);              // перевантаження Object*
}
```

Результат роботи програми наведений нижче (рис. 8.10):

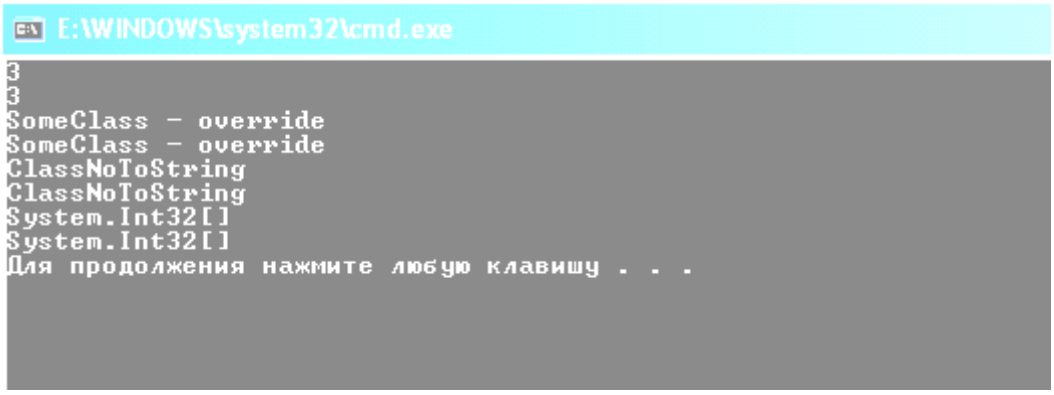

Рис. 8.10. **Результат роботи програми**

Зауважте, що метод ToString можна викликати явно як аргумент перевантаженого методу WriteLine об'єкта String, а можна викликати перевантажений метод WriteLine об'єкта String, що сам викличе метод ToString. Зауважте також, що навіть керований масив (який, насправді, є керованим типом) підтримує метод ToString.

Керовані рядкові літерали String (Рядок) і некеровані рядкові літерали ASCII й Unicode (завдяки автоматичному впакуванню) можна використати у вираженнях, у яких очікується використання керованого рядкового об'єкта String (Рядок). Однак, керований рядковий об'єкт String (Рядок) не можна використати там, де очікується поява змінних некерованих типів. Наступний приклад доводить це. Зверніть увагу на закоментовані рядки. Спробуйте їх розкоментувати і подивіться, що відбудеться.

```
//MixingStringTypes.cpp
#include "stdafx.h"
#include < stdio.h >
#include <stdlib.h>
#include <wchar.h> // для wchar__t
using namespace System;
void ExpectingManagedString(String ^str) // Рядок керований
{
	Console::WriteLine("From ManagedString : {0}",str);
}
void ExpectingASCIIString(char *str) // Рядок ASCII
{
	printf("From ASCIIString : %s \n",str);
}
void ExpectingUnicodeString(wchar_t *str) // Рядок Unicode
{
	printf("From UnicodeString : %S\n",str);
}

void main(void)
{
char *simp_str="Simple line";
wchar_t *uni_str=L"Unicode line";
```

```
String ^man_str=L"Manage line";
char    uni2simp[30];
// очікується керований тип
ExpectingManagedString(man_str);
ExpectingManagedString("Simple line - literal");
ExpectingManagedString(gcnew String(simp_str));
ExpectingManagedString(L"Unicode line - literal") ;
ExpectingManagedString(gcnew String(uni_str)) ;
Console::WriteLine();
// очікується простий символьний рядок
//ExpectingASCIIString((char*)man_str );
ExpectingASCIIString(simp_str);
ExpectingASCIIString("Simple line - literal") ;
wcstombs(uni2simp, uni_str, wcslen(uni_str);
ExpectingASCIIString(uni2simp) ;
// ExpectingASCIIString(L"Unicode line - literal") ;
Console::WriteLine();
// очікується рядок Unicode
ExpectingUnicodeString(uni_str);
ExpectingUnicodeString(L"Unicode line - literal");
//ExpectingUnicodeString ((wchar_t*)simp_str);
//ExpectingUnicodeString((wchar_t*)man_str);
}
```

Результат роботи має такий вигляд (рис. 8.11):

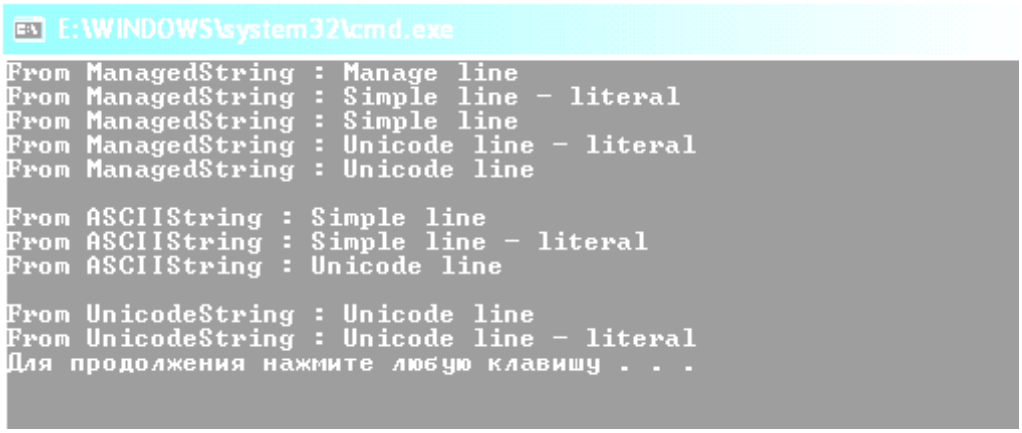

Рис. 8.11. **Результат роботи програми**

Наступний приклад показує, як можна використати оброблювач виключень при спробі доступу до неіснуючого елемента керованого масиву. Зверніть увагу, що масив містить п'ять елементів, а в циклі виробляється спроба установити значення шостого. Програма у звичайному C++ виконала б таку дію, змінивши уміст пам'яті за межами масиву. Ніхто не скаже точно, чим це могло б закінчитися. При перевірці коректності адреси виконуються дві дії: по-перше, запобігає зміна вмісту пам'яті за межами масиву; по-друге, програмі повідомляється, що виникла подібна ситуація, тим самим даючи можливість виправити помилку ще на стадії тестування. У звичайному C++ така помилка часто не проявляється доти, поки програма, за незрозумілими причинами, не припиняє роботу, звичайно в місці коду, що далеко стоїть від самої помилки.

```
#include "stdafx.h"
using namespace System;

void main ()
{
// керований масив з 5 елементів
array<int>^ intArray = gcnew array<int> (5);
for (int i=-2; i<6; i++)                    // більше чим є!!!
{
        try
        {
          intArray[i] = i;
          Console::WriteLine("Assigned A[{0}]={1}",i,i);
        }
        catch (IndexOutOfRangeException^ piore)
        {
          // повідомлення про настання виняткової ситуації
          Console::WriteLine("Oooops!");
          Console::WriteLine(piore->Message+" Index={0}",i);
    }
}
}
```

Результат роботи має такий вигляд (рис. 8.12):

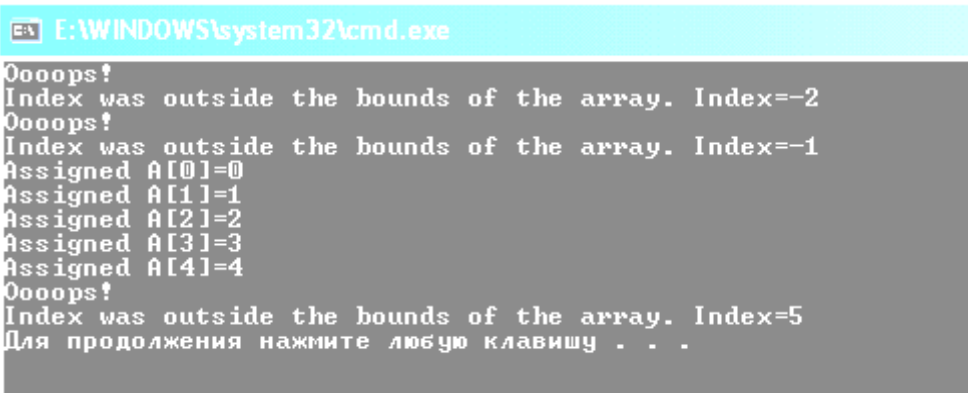

Рис. 8.12. **Результат роботи програми**

У наступному прикладі ілюструється робота з масивами, і рівняються керований двовимірний масив і звичайний некерований двовимірний масив. Зверніть ще раз увагу на те, що при роботі з некерованим масивом використовується старий синтаксис доступу до елементів масиву [][], тоді як при роботі з керованим масивом, що є істинно двовимірним, використовується синтаксис [,]. Хоча в цьому прикладі при використанні синтаксису [ ] [ ] кожний з підмасивів має однакову кількість елементів, в інших випадках вони можуть мати різні розміри (так званий масив з нерівним правим краєм). Синтаксис [,] припускає використання істинно прямокутного масиву.

```
// L8-13.cpp : main project file.
//Arrayl.cpp
#include "stdafx.h"
using namespace System;

void main ()
{
// керований одновимірний масив int
Console::WriteLine("managed 1D array of int");
array<int>^ intArray = gcnew array<int>(5);
for (int i=0; i<intArray->Length; i++)
{
        intArray[i] = i;
        Console::Write(intArray[i]); // Запис
        Console::Write("\t"); // Запис
}
```

```
Console::WriteLine();
// керований двовимірний масив рядків
Console::WriteLine("managed 2D array of Strings");
array<String^,2>^ str2DArray = gcnew array<String^,2> (2,3);
// новий Рядок
for(int row=0; row<str2DArray->GetLength(0); row++)
// цикл по рядках
{
 for (int col=0; col<str2DArray->GetLength(1); col++)
// цикл по стовпцях
 {
  str2DArray[row,col] = (row*10 + col).ToString();
  // str2DArray [рядок, стовпець] == (row*10 + стовпець).ToString();
  Console::Write(str2DArray[row,col]); // виведення елемента масиву
  Console::Write("\t");
 }
 Console::WriteLine();
}
Console::WriteLine();
// некерований двовимірний масив int (для порівняння)
Console::WriteLine("unmanaged 2D array of int");
int int2DArray[2][3];
for(int row=0; row<2; row++) // по рядках
{
 for(int col=0; col<3; col++) // по стовпцях
 {
  int2DArray[row][col] = row*10 + col;
  // int2DArray [рядок] [стовпець] = row*10 + стовпець;
  Console::Write(int2DArray[row][col]); // виведення елемента масиву
  Console::Write("\t");
 }
 Console::WriteLine();
}
}
// цикл по рядках
for(int row=0; row<str2DArray->GetLength(0); row++)
{
```

```
// цикл по стовпцях
 for (int col=0; col<str2DArray->GetLength(1); col++)
 {
  str2DArray[row,col] = (row*10 + col).ToString();
  // str2DArray [рядок, стовпець] == (row*10 + стовпець).ToString ();
  Console::Write(str2DArray[row,col]); // виведення елемента масиву
  Console::Write("\t");
 }
 Console::WriteLine();
}
Console::WriteLine();

// некерований двовимірний масив int (для порівняння)
Console::WriteLine("unmanaged 2D array of int");
int int2DArray[2][3];
for(int row=0; row<2; row++) // по рядках
{
 for(int col=0; col<3; col++) // по стовпцях
 {
  int2DArray[row][col] = row*10 + col;
  // int2DArray [рядок] [стовпець] = row*10 + стовпець;
  Console::Write(int2DArray[row][col]); // виведення елемента масиву
  Console::Write("\t");
 }
 Console::WriteLine();
}
}
```

Результат роботи має такий вигляд (рис. 8.13):

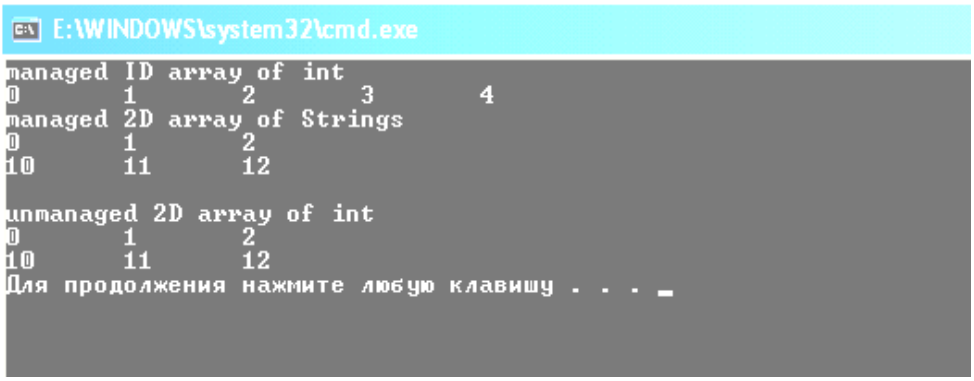

Рис. 8.13. **Результат роботи програми**

У висновку варто згадати про наявності в складі бібліотеки .NET класів колекцій, призначених для організації даних, у тому числі класи визначення списків, черг, стеків, словників.

Нижче наведений приклад використання динамічно зростаючого при необхідності масиву.

```
#include "stdafx.h"
using namespace System;
using namespace System::Collections;
int main()
{
  // Створюємо й ініціалізуємо ArrayList.
  ArrayList^ myAL = gcnew ArrayList;
  myAL->Add( "Програма" );
  myAL->Add( "№" );
  myAL->Add( "1" );
  myAL->Add( "Демонстрація колекцій" );
  myAL->Add( "!" );
  // Displays the properties and values of the ArrayList.
  Console::WriteLine( "myAL" );
  Console::WriteLine( "    Кількість: {0}", myAL->Count );
  Console::WriteLine( "    Ємність   : {0}", myAL->Capacity );
  Console::WriteLine();
  for each (String^ s in myAL)
  {
Console::WriteLine("Індекс:{0}Значення:{1}",myAL->IndexOf(s),s );
  }
}
```

Результат роботи має такий вигляд (рис. 8.14):

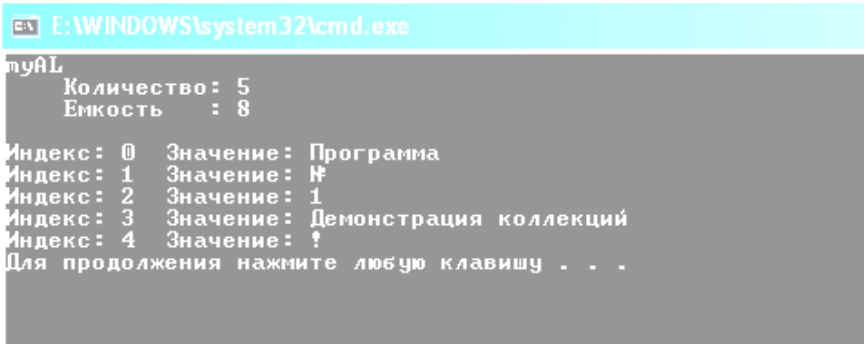

Рис. 8.14. **Результат роботи програми**

Завдання до лабораторної роботи

Варіант 1

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

суму негативних елементів масиву;

добуток елементів масиву, розташованих між максимальним і мінімальним елементами.

Упорядкувати елементи масиву за зростанням.

Варіант 2

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

суму позитивних елементів масиву;

добуток елементів масиву, розташованих між максимальним і мінімальним за модулем елементами.

Упорядкувати елементи масиву за убуванням.

Варіант 3

В одновимірному масиві, що складається з n цілих елементів, обчислити:

добуток елементів масиву з парними номерами;

суму елементів масиву, розташованих між першими й останнім нульовими елементами.

Перетворити масив таким чином, щоб спочатку розташовувалися всі позитивні елементи, а потім – усі негативні (елементи, рівні 0, уважати позитивними).

Варіант 4

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

суму елементів масиву з непарними номерами;

суму елементів масиву, розташованих між першими й останнім негативними елементами.

Стиснути масив, видаливши з нього всі елементи, модуль яких не перевищує 1. Елементи, що звільнилися наприкінці масиву, заповнити нулями.

Варіант 5

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

максимальний елемент масиву;

суму елементів масиву, розташованих до останнього позитивного елемента.

Стиснути масив, видаливши з нього всі елементи, модуль яких перебуває в інтервалі [a,b]. Елементи, що звільнилися наприкінці масиву, заповнити нулями.

Варіант 6

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

мінімальний елемент масиву;

суму елементів масиву, розташованих між першими й останнім позитивними елементами.

Перетворити масив таким чином, щоб спочатку розташовувалися всі елементи, рівні нулю, а потім – усі інші.

Варіант 7

В одновимірному масиві, що складається з n цілих елементів, обчислити:

номер максимального елемента масиву;

добуток елементів масиву, розташованих між першими й другим нульовими елементами.

Перетворити масив таким чином, щоб у першій його половині розташовувалися елементи, що стояли в непарних позиціях, а в другій половині – елементи, що стояли в парних позиціях.

Варіант 8

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

номер мінімального елемента масиву;

суму елементів масиву, розташованих між першими й другим негативними елементами.

Перетворити масив таким чином, щоб спочатку розташовувалися всі елементи, модуль яких не перевищує 1, а потім – усі інші.

Варіант 9

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

максимальний за модулем елемент масиву;

суму елементів масиву, розташованих між першими й другим позитивними елементами.

Перетворити масив таким чином, щоб елементи, рівні нулю, розташовувалися після всіх інших.

Варіант 10

В одновимірному масиві, що складається з n цілих елементів, обчислити:

мінімальний за модулем елемент масиву;

суму модулів елементів масиву, розташованих після першого елемента, рівного нулю.

Перетворити масив таким чином, щоб у першій його половині розташовувалися елементи, що стояли в парних позиціях, а в другій половині – елементи, що стояли в непарних позиціях.

Варіант 11

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

номер мінімального за модулем елемента масиву;

суму модулів елементів масиву, розташованих після першого негативного елемента.

Стиснути масив, видаливши з нього всі елементи, величина яких перебуває в інтервалі [a,b]. Елементи, що звільнилися наприкінці масиву, заповнити нулями.

Варіант 12

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

номер максимального за модулем елемента масиву;

суму елементів масиву, розташованих після першого позитивного елемента.

Перетворити масив таким чином, щоб спочатку розташовувалися всі елементи, ціла частина яких лежить в інтервалі [a,b], а потім – усі інші.

Варіант 13

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

кількість елементів масиву, що лежать у діапазоні від A до B;

суму елементів масиву, розташованих після максимального елемента.

Упорядкувати елементи масиву за убуванням модулів елементів.

Варіант 14

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

кількість елементів масиву, рівних 0;

суму елементів масиву, розташованих після мінімального елемента.

Упорядкувати елементи масиву за зростанням модулів елементів.

Варіант 15

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

кількість елементів масиву, більше C;

добуток елементів масиву, розташованих після максимального за модулем елемента.

Перетворити масив таким чином, щоб спочатку розташовувалися всі негативні елементи, а потім – усі позитивні (елементи, рівні 0, уважати позитивними).

Варіант 16

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

кількість негативних елементів масиву;

суму модулів елементів масиву, розташованих після мінімального за модулем елемента.

Замінити усі негативні елементи масиву їхніми квадратами й упорядкувати елементи масиву за зростанням.

Варіант 17

В одновимірному масиві, що складається з n цілих елементів, обчислити:

кількість позитивних елементів масиву;

суму елементів масиву, розташованих після останнього елемента, рівного нулю.

Перетворити масив таким чином, щоб спочатку розташовувалися всі елементи, ціла частина яких не перевищує 1, а потім – усі інші.

Варіант 18

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

кількість елементів масиву, менше C;

суму цілих частин елементів масиву, розташованих після останнього негативного елемента.

Перетворити масив таким чином, щоб спочатку розташовувалися всі елементи, що відрізняються від максимального не більше ніж на 20%, а потім – усі інші.

Варіант 19

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

добуток негативних елементів масиву;

суму позитивних елементів масиву, розташованих до максимального елемента.

Змінити порядок проходження елементів у масиві на зворотний.

Варіант 20

В одновимірному масиві, що складається з n речовинних елементів, обчислити:

добуток позитивних елементів масиву;

суму елементів масиву, розташованих до мінімального елемента.

Упорядкувати за зростанням окремо елементи, що стоять на парних місцях, і елементи, що стоять на непарних місцях.

Доповнення до лабораторної роботи (самостійно)

Виконати індивідуальний варіант завдання лабораторної роботи з використанням колекцій, надаваних C++/CLI.

Контрольні запитання

1. Які ключові слова використовуються при створенні керованого коду?
2. Як здійснюються стандартні введення й виведення у керованому C++?
3. Опис рядків у керованому C++.
4. Опис масивів у керованому C++.
5. У чому принципова відмінність стандартних масивів C++ від масивів у керованому C++?
6. Що таке "збирання сміття"?
7. Яким чином можна компілювати програму, чергуючи керований і некерований коди?

8. Як виробляється форматування виведення в керованому C++?
9. Що таке "дескриптори, що відслідковують"?
10. Що таке "ранг" масиву?

Рекомендована література

1. Павловская Т. А. C/C++. Программирование на языке высокого уровня. – СПб.: Питер, 2006. – 462 с.

2. Павловская Т. А. C++. Объектно-ориентированное программирование: Практикум. – СПб.: Питер, 2006. – 266 с.

3. Пирогов В. Ю. Программирование на Visual C++.NET. – СПб.: БХВ-Петербург, 2003. – 800 с.

4. Подбельский В. В. Язык C++: Учебное пособие. – 4-е изд. – М.: Финансы и статистика, 1999. – 560 с.

Зміст

Загальні положення 3
Лабораторна робота №1. Підготовка і рішення на ПК завдань обробки масивів з використанням покажчиків.....6
Лабораторна робота №2. Підготовка і рішення на ПК завдань з використанням рядків і файлів 20
Лабораторна робота №3. Підготовка і рішення на ПК завдань з використанням рядків і макросів.....34
Лабораторна робота №4. Підготовка і рішення на ПК завдань обробки масивів структур 41
Лабораторна робота №5. Підготовка і рішення на ПК завдань обробки масивів структур з використанням контейнерів.....57
Лабораторна робота №6. Дослідження структури Windows-додатка 74
Лабораторна робота №7. Дослідження взаємодії додатка з користувачем 92
Лабораторна робота №8. Розробка програм на керованому C++ 139
Рекомендована література 178

НАВЧАЛЬНЕ ВИДАННЯ

Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни

"ОСНОВИ ПРОГРАМУВАННЯ ТА АЛГОРИТМІЧНІ МОВИ"

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

Частина 2

Укладачі: **Лосєв Михайло Юрійович**
Парфьонов Юрій Едуардович
Федорченко Володимир Миколайович
Щербаков Олександр Всеволодович

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

Редактор **Грицай І. М.**
Коректор **Грицай І. М.**

План 2009 р. Поз. №228.
Підп. до друку Формат 60 × 90 1/16. Папір MultiCopy. Друк Riso.
Ум.-друк. арк. 11,25. Обл.-вид. арк. 14,06. 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.***

Методичні рекомендації до виконання
лабораторних робіт з навчальної дисципліни

"ОСНОВИ ПРОГРАМУВАННЯ ТА АЛГОРИТМІЧНІ МОВИ"

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

Частина 2