ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«КОМП'ЮТЕРНА АКАДЕМІЯ ШАГ»

Методичні вказівки

Практичні заняття

з дисципліни «Вступ до програмування ч.1(С / С ++)»

галузь знань **12 Інформаційні технології**

спеціальність **122 Комп'ютерні науки**

Львів - 2017

Методичні вказівки з практичних зайняти з навчальної дисципліни «Вступ до програмування ч.2 (C / C ++)» для студентів за спеціальністю 122 Комп'ютерні науки

розробник:
професор, д.т.н., професор кафедри _____

ЗМІСТ

Практичне заняття 1. Об'єкти, типи та значення.

1.1. Мета практичного заняття

Вивчити поняття змінної, об’єкта та типів. Навчитися реалізовувати безпечні типи та безпечні перетворення при вирішенні практичних задач.

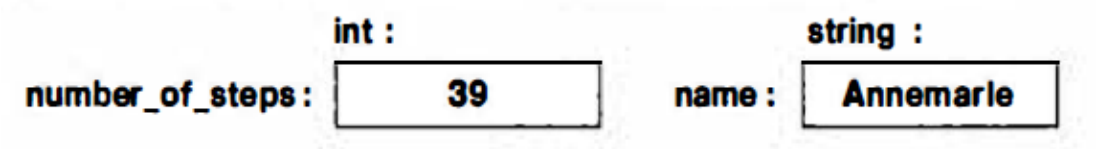
1.2. Теоретичний матеріал

1.2.1. Змінні

В принципі, не маючи можливості зберігати дані в пам'яті так, як це було зроблено в попередньому прикладі, за допомогою комп'ютера неможливо зробити нічого цікавого. "Місця", в яких зберігаються дані, називають об'єктами. Для доступу до об'єкту необхідно знати його ім'я. Іменований об'єкт називається змінною і має конкретний тип (такий, як int або string), що визначає, яку інформацію можна записати в об'єкт (наприклад, в змінну типу int можна записати число 123, а в об'єкт типу string - рядок символів "Hello, World! \ n "). а також які операції до нього можна застосовувати (наприклад, змінні типу int можна множити за допомогою оператора *, а об'єкти типу string можна порівнювати за допомогою оператора <=). Дані, записані в змінні, називають значеннями. Інструкція, що визначає змінну, називається (цілком природно) визначенням. причому у визначенні можна (і зазвичай бажано) задавати початкове значення змінної. Розглянемо наступний приклад:

```
string name = "Annemarie";
int nuinber_of_steps = 39;
```

Ці змінні можна зобразити таким чином:



Ми не можемо записувати в змінну значення неприйнятного типу.

```
string name2 = 39;                    // Помилка: 39 -не рядок
int nuinber_of_steps = "Annemarie";   // Помилка: "Annemarie" -
                                      // не ціла кількість
```

Компілятор запам'ятовує тип кожної змінної і дозволяє використовувати змінну лише так. як передбачено її типом, зазначеним у визначенні.

У мові C ++ передбачено досить широкий вибір типів. Однак можна створювати прекрасні програми, обходячись лише п'ятьма з них.

```
int number_of_steps = 39; // int - для цілих чисел
double flying_time = 3.5; // double - для чисел з плаваючою точкою
char decimal_point = '.'; // char - для СИМВОЛІВ
string name = "Annemarie"; // string - для рядків
bool tap_on = true;        // bool - для логічних змінних
```

Ключове слово double використовується з історичних причин: воно є скороченням від виразу "число з плаваючою точкою подвійної точності" ("double precision floating point.") Число з плаваючою точкою є комп'ютерне наближення математичної концепції дійсного числа.

Зверніть увагу на те, що кожен з цих типів має свій характерний спосіб запису.

```
39      // int: ціле число
3.5     // double: число з плаваючою точкою
1.1     // char: окремий символ в одинарних лапках
"Annemarie" // string: послідовність символів,
// обмежена подвійними лапками
true    // bool: або істина, або брехня
```

Інакше кажучи, послідовність цифр означає ціле число, окремий символ в одинарних лапках означає символ, послідовність цифр з десятковою крапкою являє собою число з плаваючою точкою, а послідовність символів, взятих в подвійні лапки (наприклад, "тисячу двісті тридцять чотири", "Howdy!" або "Annemarie"), позначає рядок.

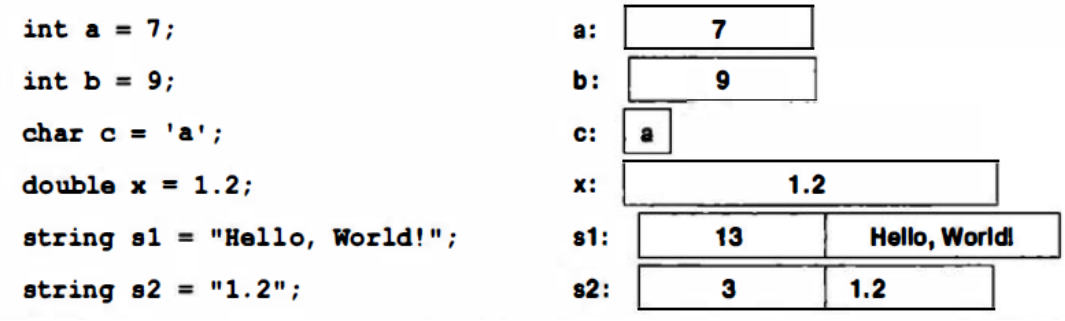
1.2.2. Типи і об'єкти

Поняття типу є центральним в мові C++ і більшості інших мов програмування. Розглянемо типи пильніше і трохи більш строго. Особливу увагу приділимо типам об'єктів, в яких зберігаються дані в процесі обчислень. Все це заощадить нам час в ході довгих обчислень і дозволить уникнути деяких непорозумінь.

- Тип визначає набір можливих значень і операцій, які виконуються над об'єктом.
- Об'єкт - ділянка пам'яті, в якій зберігається значення певного типу.
- Значення - набір бітів в пам’яті, що інтерпретується відповідно до типу.
- Змінна - іменований об'єкт.
- Оголошення - інструкція, що приписує об'єкту певне ім'я.
- Визначення - оголошення, що виділяє пам'ять для об'єкта.

Неформально об'єкт можна представити у вигляді ящика, в який можна покласти значення певного типу. У ящику для об'єктів типу int можна зберігати тільки цілі числа, наприклад 7, 42 і -399. У ящику для об'єктів типу string можна

зберігати символьні рядки, наприклад "Interoperability", "tokens:! @ # $% A & *" і "Old MacDonald had a farm". Графічно це можна уявити так:



Подання об'єкта типу string трохи складніше, ніж об'єкта типу int, так як тип string зберігає кількість символів в рядку. Зверніть увагу на те. що об'єкт типу double зберігає число, а об'єкт типу string - символи. Наприклад, змінна x містить число 1.2, а змінна s2 - три символи: '1', '. ' і 2'. Лапки навколо символу і строкових літералів в змінних не зберігаються.

Всі змінні типу int мають один і той же розмір: інакше кажучи. для кожної змінної типу int компілятор виділяє однакову фіксовану кількість пам'яті. У типовому настільному комп'ютері цей обсяг дорівнює 4 байтам (32 біта). Аналогічно фіксований розмір мають і об'єкти типів bool, char і double. У настільному комп'ютері змінні типу bool і char, як правило, займають один байт (8 біт), а змінна типу double - 8 байт. Зверніть увагу на те, що різні типи об'єктів займають різну кількість пам'яті в комп'ютері. Зокрема, змінна типу char займає менше пам'яті, ніж змінна типу int, а змінна типу string відрізняється від змінних типів double, int і char тим, що різні рядки можуть займати різну кількість пам'яті.

Сенс бітів, розміщених в пам'яті, повністю залежить від типу, використовуваного для доступу до цих бітам. Це слід розуміти так: пам'ять комп'ютера нічого не знає про типах: це просто пам'ять і більше нічого. Біти, розташовані в цій пам'яті, набувають сенсу, тільки коли ми вирішуємо, як інтерпретувати таку ділянку пам'яті. Така ситуація цілком типова при повсякденному використанні чисел. Що значить 12.5? Ми не знаємо. Це може бути 12.5 дол., 12.5 см або 12.5 кг. Тільки після того, як ми припишемо числу 12.5 одиницю виміру, воно набуде конкретного змісту.

Наприклад, один і той же набір бітів в пам'яті може становити число 120, якщо його інтерпретувати як змінну типу int, і символ 'x', якщо трактувати його як об'єкт типу char. Якщо поглянути на нього як на об'єкт типу string, то він взагалі втратить сенс, і спроба його використовувати призведе до помилки часу виконання програми. Цю ситуацію можна проілюструвати наступним чином (тут 1 і 0 означають значення бітів в пам'яті).

00000000 00000000 00000000 01111000

Цей набір бітів, записаних в ділянці пам'яті (слові), можна прочитати як змінну типу int(120) або char ('x', якщо враховувати тільки молодші 8 бітів). Біт - це одиниця пам'яті комп'ютера, яка може зберігати тільки або 0, або 1.

1.2.3. Безпека типів

Кожен об'єкт при визначенні отримує тип. Програма - або частина програми - є безпечною з точки зору використання типів (type-safe), якщо об'єкти використовуються тільки відповідно до правил, передбачених для їх типів. На жаль, існують операції, які не є безпечними з цієї точки зору. Наприклад, використання змінної до її ініціалізації є небезпечним.

```
int main ()
{
double x;              // Змінна x НЕ инициализирована,
                       // її значення не визначене
double y- x;           // Значення змінної y не визначено
double z = 2.0 + x; // Сенс операції + і значення змінної z
                       // не визначені
}
```

При спробі використовувати неініціалізованих змінну x реалізація може навіть видати помилку апаратного забезпечення. Завжди Ініціалізуйте свої змінні! У цього правила є лише кілька - дуже мало - винятків (наприклад, якщо змінна негайно використовується для введення даних). але ініціалізація змінних в будь-якому випадку - це хороша звичка, що запобігає безліч неприємностей.

Повна безпека типів є ідеалом і, отже. загальним правилом для мови. На жаль, компілятор мови C ++ не може гарантувати повну безпеку типів, але ми можемо уникнути її порушень, використовуючи хороший стиль програмування і перевірки часу виконання. Ідеально було б взагалі ніколи не використовувати властивості мови, безпеку яких компілятор не в змозі довести.

Така безпеку типів називається статичною. На жаль, це сильно обмежило б найбільш цікаві сфери застосування програмування. Очевидно, якби компілятор неявно генерував код, перевіряючий порушення безпеку типів, і перехоплював їх все, то це уповільнювало б програму і виходило б за рамки мови C ++. Якщо ми приймаємо рішення використовувати прийоми, які не є безпечними з точки зору використання типів, то повинні перевіряти себе самі і самостійно виявляти такі ситуації.

Ідеал безпеки типів неймовірно важливий при написанні коду. Ось чому ми поминаємо про нього так рано. Будь ласка, пам'ятайте про цю небезпеку і намагайтеся уникати її в своїх програмах.

Безпечні перетворення.

Вище ми бачили, що не можна безпосередньо складати об'єкти типу char або порівнювати об'єкти типів double і int. Однак в мові C ++ це можна зробити у

спосіб. При необхідності об'єкт типу char можна перетворити в об'єкт типу int, а об'єкт типу int - в об'єкт типу double. Розглянемо приклад.

```
char c = 'x';
int i1 = c;
int i2 = 'x';
```

Тут значення змінних i1 і i2 рівні 120, тобто 8-бітовому ASCII-коду символу 'x'. Це простий і безпечний спосіб отримання числового уявлення символу. Ми називаємо таке перетворення типу char в тип int безпечним, оскільки при цьому не відбувається втрати інформації: інакше кажучи. Ми можемо скопіювати результат, що зберігається в змінної типу int, назад в змінну типу char і отримати початкове значення.

```
char c2 = i1;
cout << c << '' << i1 << '' << c2 << '\ n';
```

Цей фрагмент програми виводить на екран наступний результат:

```
x 120 x
```

У цьому сенсі - що значення завжди перетворюється в еквівалентну значення або (для типу double) в найкраще наближення еквівалентного значення - такі перетворення є безпечними.

```
bool в char
bool в int
bool в double
char в int
char в doule
int в doube
```

Найбільш корисним є перетворення змінної типу int в змінну типу double, оскільки дозволяє використовувати комбінацію цих типів в одному вираженні.

```
double d1 = 2.3;
double d2 = d1 + 2; // Перед складанням 2 перетворюється в 2.0
if (d1 <0)              // Перед порівнянням 0 перетворюється в 0.0
        cout <<       "d1 - негативно ";
```

Для дійсно великих чисел типу int при їх перетворенні в змінні типу double ми можемо (в деяких комп'ютерах) втратити точність. Однак ця проблема виникає рідко.

Небезпечні перетворення.

Безпечні перетворення зазвичай не турбують програмістів і спрощують розробку програм. На жаль, мова C ++ допускає також (неявні) небезпечні перетворення. Під небезпечними перетвореннями ми маємо на увазі те, що значення може неявно перетворитися в значення іншого типу, яке не дорівнює початковому. Розглянемо приклад.

```
int main ()
{
int a = 20000;
char з = a;    // Спроба втиснути велике значення типу int
               // в маленьку змінну типу char
int b = c;
if (a! = b) //! = означає "не дорівнює"
        cout << "Ой !:" << a << "! =" << b << '\ n'; else
        cout << "Ого! Однак у нас дуже великий char \ n";
}
```

Такі перетворення називають "що звужують", оскільки вони поміщають значення в об'єкт, розмір якого дуже малий ("вузький") для його зберігання. На жаль, лише деякі компілятори попереджають про небезпечну ініціалізації змінної типу char значенням змінної типу int. Проблема полягає в тому, що тип int, як правило, набагато більше типу char, так що він може (в нашому випадку так і відбувається) зберігати значення типу int, яке неможливо уявити значенням типу char. Спробуйте проекспериментувати з наведеної далі програмою.

```
int main ()
{
double d = 0;
while (cin >> d) {// Повторюємо наступні інструкції,
                        // поки вводяться числа
int i = d;       // Спроба втиснути double в int
char з = i;     // Спроба втиснути int в char
int i2 = c;      // Отримуємо ціле значення типу char
cout << "d =" << d  // Початкове значення типу аоіИе
<< "i = n << i // Перетворене в значення int
<< 11 i2 == "<< i2 // Целочисленное значення
char << "char (" << c << ") \ n"; // Символ
}
}
```

Ви виявите, що багато числа призводять до "безглуздим" результатів. Образно кажучи, це відбувається, коли ви намагаєтеся перелити рідину з трилітрової банки в півлітрову. Компілятор дозволяє існують такі перетворення, незважаючи на їх небезпеку.

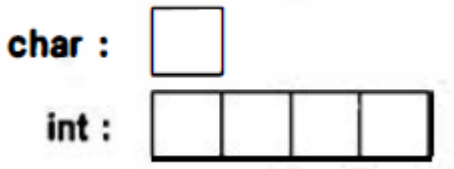
double в int
double в char
double в bool
int в char
int в bool
char в bool

Ці перетворення є небезпечними в тому сенсі, що значення, яке зберігається в змінної, може відрізнятися від присвоєного. Чому ця ситуація вважається проблемою? Тому що часто ви і не підозрюєте про те, що таке перетворення мало місце. Розглянемо приклад.

```
double x = 2.1;
                // Якийсь код
int y = x;      // Значення змінної y стає рівним 2
```

До моменту визначення змінної y ви вже могли забути, що змінна x має тип double, або випустити з уваги, що перетворення double в int призводить до усічення (округлення у напрямку до нуля). Результат цілком передбачуваний: сім десятих втрачені.

Перетворення int в char не породжує проблем з урізанням - ні тип int, ні тип char не можуть уявити дробову частину цілого числа. Однак змінна типу char може зберігати тільки дуже невеликі цілочисельні значення. У персональних комп'ютерах змінна типу char займає 1 байт. В той час як змінна типу int - 4 байта.



Таким чином, ми не можемо записати велике число, наприклад 1000, у змінну типу char без втрати інформації: значення "звужується". Розглянемо приклад.

```
int a = 1000;
char b = a; // Значення b стає рівним -24
```

Не всі значення типу int мають еквіваленти типу char, а точний діапазон значень типу char залежить від конкретної реалізації. На персональних

комп'ютерах значення типу char коливаються в діапазоні [-128,127], але мобільність програм можна забезпечити тільки в діапазоні [0,127], оскільки не кожен комп'ютер є ПК, і у різних комп'ютерів різні діапазони значень char, наприклад такі як [0,255].

Чому люди змирилися з проблемою звужують перетворень?

Основна причина носить історичний характер: мова C ++ успадкував звужують перетворення від попередника. мови C. До першого дня існування мови C ++ вже було безліч програм. написаних на мові C і містять звужують перетворення. Крім того, багато такі перетворення насправді не створюють жодних проблем, оскільки використовувані значення не виходять за межі допустимих діапазонів. і багато програмістів скаржаться, що "компілятори вказують їм. що треба робити". Зокрема. досвідчені програмісти легко справляються з проблемою небезпечних перетворень в невеликих програмах, хоча в більших програмах і в недосвідчених руках такі перетворення можуть стати джерелом помилок. Однак компілятори можуть попереджати програмістів про які звужують перетвореннях - і багато хто з них роблять це.

В C++ 11 вводиться запис ініціалізації. яка забороняє звужують перетворення. Наприклад. ми можемо (і повинні) переписати наведені вище проблемні приклади з використанням записи зі списками в фігурних дужках замість запису зі знаком присвоювання:

```
double x {2.7};      //    0!: \.
int y {x};           //    Помилка:   double -> int може бути звужує
int a {1000}; //     OK
char b {a};          //    Помилка:   int -> char може бути звужує
```

Коли ініціалізатор є цілочисельний літерал. компілятор в змозі перевірити його фактичне значення і прийняти значення, які не викликають звуження:

```
char b1 {1000};     // Помилка: звуження (в припущенні
                    // 8-бітового типу char)
char b2 {48};       // ОК
```

1.3. Задачі для розв'язання

На кожному етапі виконання завдання запускайте програму і переконуйтеся, що вона робить саме те, що ви очікували. Складіть список зроблених помилок, щоб ви могли уникати їх в майбутньому.

1. Напишіть програму, яка формує просту форму для письма на основі вхідної інформації.

Для початку наберіть програму

```
// Вважати і записати ім'я
#include "std lib facilities.h"
int main ()
{
        cout << "Введіть ваше ім'я (і натисніть 'enter'): \ n";
        strinq first_name; // first_name - змінна типу string
        cin >> first_наше; // Прочитуємо символи в first name
        cout << "Привіт," << first name << "! \ n";
}
```

запропонувавши користувачеві ввести своє ім'я і передбачивши виведення рядка "Привіт, first_name", де first_name - це ім'я, введене користувачем. Потім змінюйте програму в такий спосіб: змініть запрошення на рядок "Введіть ім'я адресата" та змініть висновок на рядок "Дорогий first_name".

2. Введіть одну або дві вступні фрази. наприклад "Як справи? У мене все добре. Я сумую за тобою". Переконайтеся, що перший рядок відокремлена від інших. Додайте ще декілька рядків на свій розсуд - це ж ваш лист.

3. Запропонуйте користувачеві ввести ім'я іншого приятеля і збережіть його в змінної friend_name. Додайте в ваш лист наступний рядок: "Чи давно ти зустрічав friend_name?".

4. Оголосіть змінну типу char с ім'ям friend_sex і ііціалізуйте її нулем. Запропонуйте користувачеві ввести значення m, якщо ваш друг - чоловік, і f - якщо жінка. Дайте змінної friend_sex введене значення. Потім за допомогою двох інструкцій if запишіть наступне.

Якщо один - чоловік, то напишіть рядок "Якщо ти побачиш friend_ name, будь ласка, попроси його зателефонувати мені".

Якщо один - жінка, то напишіть рядок "Якщо ти побачиш friend_ name, будь ласка, попроси її зателефонувати мені".

5. Запропонуйте користувачеві ввести вік адресата і надайте його змінної age, що має тип int. Ваша програма повинна вивести на екран рядок "Я чув, ти щойно відзначив день народження і тобі виповнилося age років". Якщо значення змінної age менше або дорівнює 0 або більше, або дорівнює 110, зателефонуйте іншому simple_error ("ти жартуєш!"), Використовуючи функцію simple_error () з заголовки std lib facilities.h.

6. Додайте в ваш лист наступне.

Якщо вашому другу менше 12 років, напишіть "На наступний рік тобі виповниться age + 1 років".

Якщо вашому другу 17 років, напишіть "У наступному році ти зможеш голосувати".

Якщо вашому другу більше 70 років, напишіть "Я сподіваюся, що ти не сумуєш на пенсії".

Переконайтеся, що ваша програма правильно обробляє кожне з цих значень.

7. Додайте рядок "Щиро твій", потім введіть дві порожні рядки для підписи і вкажіть своє ім'я.

8. Напишіть на мові C ++ програму, Яка перетворює милі в кілометри. Ваша програма повинна містити зрозуміле запрошення користувачеві ввести кількість миль. Вказівка: в одній милі 1.609 км.

9. Напишіть програму. яка нічого не робить. а просто оголошує ряд змінних з допустимими і неприпустимими іменами (наприклад, int double = 0;). і подивіться на реакцію на них компілятора.

10. Напишіть програму, яка пропонує користувачеві ввести два цілочисельних значення. Запишіть ці значення в змінні типу int з іменами val1 і val2. Напишіть програму. визначальну найменше та найбільше значення, а також суму, різниця, добуток і частку цих значень.

11. Змініть програму так, щоб вона просила користувача ввести два числа з плаваючою точкою і зберігала їх в змінних типу double. Порівняйте результати роботи цих двох програм для деяких вхідних даних на ваш вибір. Чи збігаються ці результати? Чи повинні вони збігатися? Чим вони відрізняються?

12. Напишіть програму. яка пропонує користувачеві ввести три цілих числа, а потім виводить їх у порядку зростання, розділяючи комами. Наприклад, якщо користувач вводить числа 10 4 6. то програма повинна вивести на екран числа 4, 6, 10. Якщо два числа збігаються. то вони повинні йти одне за іншим. Наприклад. якщо користувач вводить числа 4 5 4, то програма повинна вивести на екран 4, 4, 5.

13. Виконайте задачу 10 для трьох строкових значень. Так, якщо користувач вводить значення Steinbeck, Hemingway, Fitzgerald, то висновок програми повинен має вигляд Fitzgerald, Hemingway. Steinbeck.

14. Напишіть програму. перевіряє парність або непарність цілих чисел. Як завжди. Переконайтесь, що виведення програми ясний і повний. Інакше кажучи, не слід обмежуватися простою констатацією на кшталт "так" або "ні". Висновок повинен бути інформативним, наприклад "Число 4 є парним".

15. Напишіть програму, що перетворює слова "нуль "," два "і так далі в цифри 0, 2 , тощо. Коли користувач вводить число у вигляді слова, програма повинна вивести на екран відповідну цифру. Виконайте цю програму для цифр 0, 1,2, 3 і 4. якщо користувач введе що-небудь інше, наприклад фразу "дурний комп'ютер!", програма повинна відповісти "Я не знаю такого числа!"

16. Напишіть програму, приймаючу на вході символ операції з двома операндами і виводить на екран результат обчислення. наприклад:

+ 100 3.14

* 4 5

Вважайте символ операції в рядок з ім'ям operation і, використовуючи інструкцію if, з'ясуйте, яку операцію хоче виконати користувач, наприклад if (operation = "+"). Зчитувати операнди в змінні типу double. Виконайте операції +, -, * · /, plus, minus, mul і div, що мають очевидний сенс.

11. Напишіть програму. яка пропонує користувачеві ввести певну кількість 1 -, 5, 1 0-, 25-, 50-центових і доларових монет.

Користувач повинен окремо ввести кількість монет кожного номіналу, наприклад "Скільки у вас одвоцентових монет?"

Потім програма повинна вивести результат приблизно такого вигляду.

У вас 23 одноцентові монети.
У вас 17 п'ятицентових монет.
У вас 14 десятицентових монет.
У вас 7 двадцатіпятіцентових монет.
У вас 3 п'ятидесятицентові монети.
Загальна вартість ваших монет дорівнює 573 центам.

Вдосконалить програму: якщо у користувача тільки одна монета, виведіть відповідь в граматично правильній формі. Наприклад, "14 десятицентових монет" і "1 одноцентову монету" (а не "1 одноцевтових монет"). Крім того, виведіть результат в доларах і центах, тобто 5 доларів 73 центи. а не 573 цента.

Практичне заняття 2. Обчислення. Вирази. Інструкції.

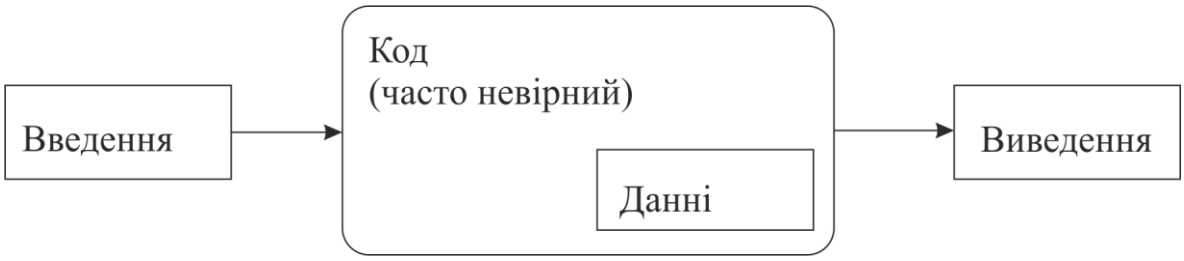
2.1. Мета практичного заняття

Вивчити основні технології використання обчислень, виразів та інструкцій, що дають змогу побудувати правильну і оптимальну програму.

2.2. Теоретичний матеріал

2.2.1. Обчислення

Всі програми виконують обчислення; інакше кажучи, вони отримують на вхід якісь дані і виводять якісь результати.



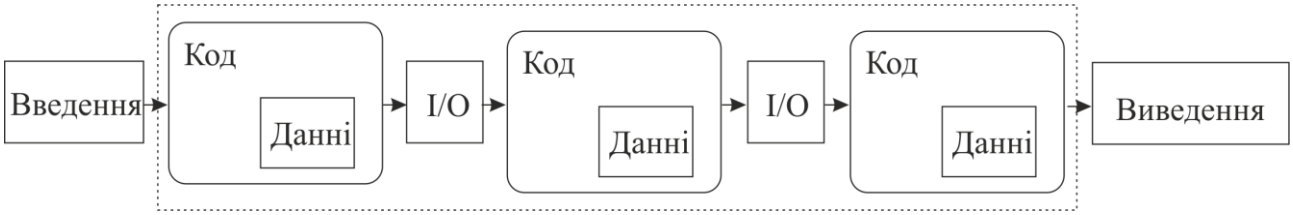
Вхідна інформація може надходити з клавіатури, від миші, з сенсорного екрану, з файлів, від інших пристроїв введення та інших частин програми. До категорії "інші пристрої введення" відносяться більшість цікавих джерел даних: музичні клавішні пульти, пристрої відеозапису, датчики температури, сенсори цифрових відеокамер, тощо. Різноманітність цих пристроїв нескінченно.

Для обробки вхідної інформації програми зазвичай використовують спеціальні дані, які називають структурами даних (data stru-ctures) або станами (states). Наприклад, програма, що імітує календар, може містити списки святкових днів в різних країнах і список ваших ділових зустрічей. Одні з цих даних є частиною програми з самого початку, а інші з'являються, коли програма зчитує вхідні дані і витягує з них корисну інформацію. Наприклад, програма-календар може створювати список ваших ділових зустрічей в міру їх введення в неї. У цьому випадку основний вхідний інформацією є місяці і дні зустрічей (можливо, що вводяться за допомогою клацання мишею) і дані про самих зустрічах (ймовірно, що вводяться за допомогою клавіатури).

Пристроєм виведення цієї програми є екран, на якому виводиться календар і дані про призначені зустрічі, а також кнопки і запрошення для введення, які програма може виводити на екран.

Вхідна інформація може надходити від самих різних джерел. Аналогічно результати можуть виводитися на самі різні пристрої: на екран, у файл, в з'єднання з мережею, в інші пристрої виведення. в інші програми або частини програми. До прикладів пристроїв виведення відносяться також мережеві інтерфейси, музичні синтезатори. електричні мотори. лампочки і світлодіоди, обігрівачі тощо

З програмістської точки зору найбільш важливими і цікавими категоріями введення-виведення є "в іншу програму" і "в інші частини програми". Велика частина цієї книги присвячена останньої категорії: як представити програму у вигляді взаємодіючих частин і як забезпечити спільний доступ до даних і обмін інформацією. Це ключові питання програмування. Проілюструємо їх графічно.



Абревіатура "I/O" означає "введення-виведення". В даному випадку висновок з однієї частини програми є введенням в наступну частину. Ці частини програми мають доступ до даних, що зберігаються в основній пам'яті, на постійному пристрої зберігання даних (наприклад, на диску) або передаються через мережеві з'єднання. Під частинами програми ми розуміємо сутності, такі як функція, що обчислює результат на основі отриманих аргументів (наприклад, витягує корінь квадратний з числа з плаваючою точкою). Функція, що виконує дії над фізичними об'єктами (наприклад, змальовує лінію на екрані), або функція, що модифікує якусь таблицю в програмі (наприклад, додає ім'я в таблицю клієнтів).

Коли ми говоримо "виклик" і "висновок", то зазвичай маємо на увазі, що в комп'ютер вводиться або з комп'ютера виводиться якась інформація, але, як ви незабаром побачите, ми можемо використовувати ці терміни і для ін-формації, переданої іншій частині програми або отриманої від неї. Інформацію, яка є введенням в частину програми, часто називають аргументом, а дані, що надходять від частини програми, - результатом.

Обчисленням ми називаємо якусь дію, що створює певні результати і засноване на певних вхідних даних. наприклад породження результату (виведення), рівного 49, на основі аргументу (введення), рівного 7. За допомогою обчислення (функції) зведення в квадрат square. Як курйозний факт нагадаємо, що до 1950-х років комп’ютер (обчислювачем) в США називався людина. виконував обчислення, наприклад бухгалтер, навігатор або фізик. В даний час ми просто передоручили більшість обчислень комп'ютерів (машинам), серед яких найпростішими є кишенькові калькулятори.

2.2.2. Вирази

Основними будівельними конструкціями програм є | j вираження. Вираз обчислює певне значення на основі деякої кількості операндів. Найпростіше вираз являє собою звичайну літерально константу, наприклад 'a', 3.14 або "Norah".

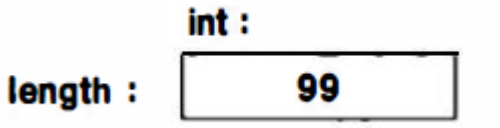
Імена змінних також є виразами. Змінна - це об'єкт, що має ім'я. Розглянемо приклад.

```
// Обчислення площі:
int length = 20;      // літерально ціле значення (використовується
// для ініціалізації змінної)
int width = 40;
int area = length * width; // Множення
```

Тут літерали 20 і 40 використовуються для ініціалізації змінних length і width, відповідних довжині і ширині. Після цього довжина і ширина перемножуються; інакше кажучи, ми перемножуємо значення length і width. Тут вираз "значення length" є скорочення виразу "значення, що зберігається в об'єкті з ім'ям length". Розглянемо ще один приклад.

```
length = 99; // Надаємо length значення 99
```

Тут слово length, що позначає лівий операнд оператора присвоювання, означає "об'єкт з ім'ям length", тому цей вислів читається так: "записати число 99 в об'єкт з ім'ям length". Слід розрізняти ім'я length, що стоїть в лівій частині оператора присвоювання або ініціалізації (воно називається "lvalue змінної length" або "об'єкт з ім'ям length") і в правій частині цих операторів (в цьому випадку воно називається "rvalue змінної length", "значення об'єкта з ім'ям length "або просто" значення length "). У цьому контексті корисно уявити змінну у вигляді ящика. позначеного іменем.



Інакше кажучи, length - це ім'я об'єкта типу int, що містить значення 99. Іноді (як lvalue) ім'я length відноситься до ящика (об'єкту), а іноді (як rvalue) - до самого значенням, що зберігається в цьому ящику.

Комбінуючи вираження за допомогою операторів, таких як + і *, ми можемо створювати більш складні вирази, так, як показано нижче. При необхідності для угруповання вираження можна використовувати дужки.

```
int perimeter = (length + width) * 2; // Скласти і помножити
```

Без дужок цей вислів довелося б записати наступним чином:

```
int perimeter = length * 2 + width * 2;
```

що занадто громіздко і провокує помилки на кшталт такої:

```
int perimeter = length + width * 2; // Скласти width * 2 з length
```

Остання помилка є логічною, і компілятор не може її виявити. Компілятор просто бачить змінну з ім'ям perimeter, ініціалізувала коректним виразом. Якщо результат виразу не має сенсу, то це ваші проблеми. Ви знаєте математичне визначення периметра, а компілятор - немає.

У програмах застосовуються звичайні математичні правила, що регламентують порядок виконання операторів, тому length + width * 2 означає length + (width * 2). Аналогічно вираз a * b + c / d означає (a * b) + + (c / d), а не a * (b + c) / d.

Перше правило використання дужок говорить: "Якщо сумніваєшся, використовуй дужки". І все ж програміст повинен навчитися правильно формувати вираження, щоб не сумніватися в значенні формули a * b + c / d. Занадто часте використання дужок, наприклад (a * b) + (c / d), знижує читабельність програми.

Чому ми піклуємося про читабельність? Тому що ваш код будете читати не тільки ви, але і, можливо, інші програмісти, а заплутаний код уповільнює читання і перешкоджає його аналізу. Незграбний код не просто складно читати, а й важко виправляти. Погано написаний код часто приховує логічні помилки. Чим більше зусиль потрібно при його читанні, тим складніше переконати себе та інших, що він є правильним. Не пишіть занадто складних виразів на кшталт

```
a * b + c / d * (ef / g) / h + 7 // Занадто складно
```

і завжди намагайтеся вибирати осмислені імена.

Вирази зі сталими.

У програмах, як правило, використовується безліч констант. Наприклад, в програмі для геометричних обчислень може використовуватися число "пі", а в програмі для перерахунку дюймів в сантиметри - множник 2.54. Очевидно, що цим константам слід призначати осмислені імена (наприклад, pi, а не 3.14159). Аналогічно константи не повинні змінюватися випадковим чином. З цієї причини в мові C ++ передбачено поняття символічної константи, тобто іменованого об'єкта, якому після його ініціалізації неможливо присвоїти нове значення. Розглянемо приклад.

```
const expr double pi = 3.14159;
pi = 7;// Помилка: присвоювання значення константі
int v = 2 * pi / r; // ОК: ми читаємо, але не змінюємо значення pi
```

Такі константи корисні для підвищення зручності читання програм. Побачивши фрагмент коду. ви. звичайно, зможете здогадатися про те, що константа 3.14159 є наближенням числа "пі", але що ви скажете про кількість 299792458? Крім того, якщо вас попросять змінити програму так. щоб число "пі" було записано з точністю до 12 десяткових знаків, то. можливо, ви станете

шукати в програмі число 3.14, але якщо хто-небудь несподівано вирішив апроксимувати число "пі" дробом 22/7, то, швидше за все, ви її не знайдете. Набагато краще змінити визначення константи pi, вказавши необхідну кількість знаків.

```
const expr double pi = 3.14159265359;
```

Отже, в програмах краще використовувати не літерали (за винятком самих очевидних, таких як 0 і 1). Замість них слід застосовувати константи з інформативними іменами. Неочевидні літерали в програмі (за рамками визначення констант) глузливо називають магічними.

У деяких місцях, наприклад в мітках оператора case, мова C ++ вимагає використовувати цілочисельні вирази зі сталими. тобто вирази, що мають цілочисельні значення і складаються виключно з констант. Розглянемо приклад.

```
constexpr int max = 17;    // Літерал є
int val = 19;              // константним виразом
шах + 2                    // Константне вираз
                           // (константа плюс літерал)
va1 + 2                          // Неконстантний вираз:
                           // використовується змінна.
```

До речі, число 299792458 - одна з фундаментальних констант всесвіту, що означає швидкість світла у вакуумі, виміряну в метрах в секунду. Якщо ви її відразу не впізнали. то, цілком можливо, будете відчувати труднощі при розпізнаванні інших літералів в програмі. Уникайте "магічних" констант!

Символічна константа const expr повинна мати значення, відоме під час компіляції. наприклад:

```
constexpr int шах = 100;
void use (int n)
{
constexpr int c1 = max + 7; // ОК: c1 одно 107
constexpr int c2 - n + 7; // Помилка: значення c2 невідомо
}
```

Для ситуацій. коли значення "змінної" ініціалізується значенням, яке невідомо під час компіляції, але після ініціалізації залишається незмінним. C ++ пропонує другу різновид констант (const):

```
constexpr int max = 100;
void use (int n)
{
```

```
constexpr int cl = max + 7; // ОК: cl одно 107
const int c2 = n + 7;         // ОК, але не намагайтеся
// змінювати значення c2
// ...
c2 • 7; // Помилка: c2 є const
}
```

Такі "константні змінні" дуже поширені з двох причин.

- В C ++ 98 не було constexpr, так що програмісти використовували ключове слово const.
- "Змінні", які не є константними виразами (їх значення невідомі під час компіляції), але значення яких не змінюються після ініціалізації, широко використовуються самі по собі.

Перетворення.

Типи в виразах можна "змішувати". Наприклад. вираз 2.5 / 2 означає розподіл значення типу double на значення типу int. Що це означає? Яке розподіл виконується: цілих чисел або з плаваючою точкою? Цілочисельне ділення відкидає залишок; наприклад, 5/2 дорівнює 2. Розподіл чисел з плаваючою точкою відрізняється тим, що залишок в його результаті не відкидали; наприклад, 5.0 / 2. Про одно 2.5. Отже, відповідь на питання "Які числа діляться в вираженні 2.5 / 2: цілі або з плаваючою точкою?" абсолютно очевидний: "Зрозуміло, з плаваючою точкою; в іншому випадку ми втратили б інформацію". Ми хотіли б отримати відповідь 1.25, а не 1, і саме 1.25 ми і отримаємо. Правило (для розглянутих нами типів) говорить: якщо оператор має операнд типу double, то використовується арифметика чисел з плаваючою точкою і результат має тип double; в іншому випадку використовується целочисленная арифметика, і результат має тип int, наприклад:

```
5/2 дорівнює 2 (а не 2.5)
2.5 / 2 дорівнює 2.5 / double (2), тобто 1.25
'A' + 1 означає int { "a '} + 1
```

Записи type (value) і type {value} означають "перетворити value в тип type. Як якщо б ви ініціалізували змінну типу type значенням value". Іншими словами, при необхідності компілятор перетворює ("підвищує") операнд типу int в операнд типу double, а операнд типу char - в операнд типу int. Обчисливши результат. компілятор може перетворити його знову для використання в якості ініціалізатор або в правій частині оператора присвоювання, наприклад:

```
double d = 2.5; int i = 2;
double d2 = d / i; // d2 == 1.25 int i2 = d / i; // i2 == 1
int i3 {d / i}; // Помилка: перетворення double -> int
d2 = d / i;     // d2 - 1.25
```

```
i2 - d / i; // i2 == 1
```

Будьте обережні: якщо вираз містить числа з плаваючою точкою, можна легко забути про правила цілочисельного ділення. Розглянемо звичайну формулу для перетворення температури за Цельсієм в температуру за Фаренгейтом: / = 9/5 c + 32. Її можна записати так:

```
double dc; cin >> dc;
double df = 9/5 * dc + 32; // Обережно!
```

На жаль, незважаючи на цілком логічну запис, цей вислів не дає точного перетворення температури: значення 9/5 дорівнює 1, а не 1.8, як ми сподівалися. Для того щоб формула стала правильною, або 9, або 5 (або обидва числа) слід перетворити в значення типу double.

```
double dc; cin >> dc;
double d £ = 9.0 / S * dc + 32; // Краще
```

2.2.3. Інструкції

Вираз обчислює значення по набору операндів. А що робити, якщо потрібно обчислити кілька значень? А що якщо щось необхідно зробити багато разів? А що робити, якщо треба зробити вибір з кількох альтернатив? А якщо нам потрібно вважати вхідну інформацію або вивести результат? У мові C ++, як і в багатьох мовах програмування, для створення таких виразів існують спеціальні конструкції, іменовані інструкціями (statement).

До сих пір ми стикалися з двома видами інструкцій: виразами і оголошеннями. Інструкції першого типу є вираження, які завершуються крапкою з комою, наприклад:

```
a = b;
++ b
```

Перед вами дві інструкції, що представляють собою вирази. Наприклад, присвоювання = - це оператор, тому a = b - це вираз, і для його завершення необхідно поставити крапку з комою a = b; в результаті виникає інструкція. Навіщо потрібна крапка з комою? Причина носить скоріше технічний характер. Розглянемо приклад.

```
a = b ++ b;    // Синтаксична помилка: пропущена крапка з комою
```

Без крапки з комою компілятор не знає, що означає цей вислів: a = b ++; b; або a = b; ++ b ;. Проблеми такого роду не обмежуються мовами програмування. Наприклад, розглянемо вираз "Стратити не можна помилувати!" Стратити чи

помилувати? Для того щоб усунути неоднозначність, використовуються знаки пунктуації. Так, поставивши кому, ми повністю вирішуємо проблему: "Стратити не можна, помилувати!" Коли інструкції йдуть одна за одною, комп'ютер виконує їх в порядку запису. Розглянемо приклад.

```
int a = 7;
cout << a << '\ n';
```

Тут оголошення з ініціалізацією виконується до оператора виведення. В цілому ми хочемо. щоб інструкція давала якийсь результат (не обов'язково має вигляд значення). Без цього інструкції, як правило, не приносять користі. Розглянемо приклад.

```
1 + 2; // Виконується додавання, але суму використовувати неможливо
a * b; // Виконується множення, але твір не використовується
```

Такі безрезультатні інструкції зазвичай є логічними помилками, і компілятори часто попереджають програмістів про це. Таким чином, інструкції, що представляють собою вирази, зазвичай є інструкціями привласнення, введення-виведення або виклику функції.

Згадаємо ще про один різновид: порожній інструкції. Розглянемо наступний код:

```
if (x == 5);
{y = 3; }
```

Це виглядає, як помилка, і це майже правда. Крапка з комою в першому рядку, взагалі-то, не повинна стояти на цьому місці. Але, на жаль, ця конструкція в мові C ++ вважається цілком допустимою. Вона називається порожній інструкцією, тобто інструкцією, яка нічого не робить. Порожня інструкція, що стоїть перед крапкою з комою, рідко буває корисною. У нашому випадку компілятор не видасть ніякого попередження про помилку, і вам буде важко зрозуміти причину неправильної роботи програми.

Що станеться, коли ця програма почне виконуватися? Компілятор перевірить, чи рівне значення змінної x числу 5. Якщо ця умова істинно, то буде виконана така інструкція (порожня). Потім програма перейде до виконання наступної інструкції, присвоївши змінної y значення 3. Якщо ж значення змінної x не дорівнює 5, то компілятор не виконуватиме порожню інструкцію (що також не породжує ніякого ефекту) і присвоїть змінній y значення 3 (навряд чи це те, чого ви домагалися, якщо значення x не рівне 5). Іншими словами, інструкція if не має ніякого значення: змінної y буде присвоєно значення 3 незалежно від значення змінної x. Ця ситуація є типовою для програм, написаних новачками, причому такі помилки буває дуже важко виявити.

Наступний розділ присвячений інструкцій. що дозволяє змінити порядок обчислень і висловити більш складні обчислення, ніж ті, які зводяться до послідовного виконання ряду інструкцій.

Інструкції вибору.

У програмах, як і в житті, ми часто робимо вибір з кількох альтернатив. У мові C ++ для цього використовуються інструкції if і switch.

Інструкція if

Найпростіша форма вибору в мові C ++ реалізується за допомогою інструкції if, що дозволяє вибрати одну з двох альтернатив. Розглянемо приклад.

```
int main ()
{
int a = 0;
int b = 0;
cout << "Будь ласка, введіть два цілих vncna \ n";
cin >> a >> b;
if (a <b)       // Умова
                // 1-я альтернатива
        // (вибирається, якщо умова істинна):
cout << "max (" << a << "," << b
        << ") одно" << b << "\ n";
else
        // 2-я альтернатива
        // (вибирається, коли умова помилкова):
        cout << "max (" << a << "," << b
        << ") одно" << a << "\ n";
}
```

Інструкція if здійснює вибір з двох альтернатив. Якщо його умова є істинним. то виконується перша інструкція; в іншому випадку виконується друга. Це проста конструкція. Вона існує в більшості мов програмування. Фактично більшість базових конструкцій в мовах програмування є просто новий запис понять, відомих всім ще зі шкільної лави або навіть з дитячого садка. Наприклад, вам, ймовірно, говорили в дитячому саду, що для того, щоб перейти вулицю, ви повинні дочекатися, поки на світлофорі загориться зелене світло: "якщо горить зелене світло, то можна переходити, а якщо горить червоне світло, то необхідно почекати" . У мові C ++ це можна записати як-то так:

```
if (traffic_light == green) go ();
if (traffic_light == red) wait ();
```

Отже, базові поняття прості, але і просте поняття можна використовувати занадто спрощено. Розглянемо неправильну програму (як правило, не звертаємо увагу на відсутність директив #include).

```
// Перетворення дюймів в сантиметри і навпаки
// Суфікс 'i' або 'з' озна чає одиницю виміру на вході
int main ()
{
constexpr double cm_per_inch = 2.54; // Сантиметрів в дюймі
double length = 1; // Довжина (дюйми або см)
char unit = 0;
cout << "Будь ласка, введіть довжину"
        << "і одиницю виміру (з або і): \ n";
cin >> length >> unit;
if (unit 'i')
        cout << length << "in == ''
        << cm_per_inch * length << "cm \ n";
else
        cout << length << "cm =="
        << length / cm_per_inch << "in \ n";
}
```

Насправді ця програма працює приблизно так, як і передбачено: введіть 1i, і ви отримаєте повідомлення 1in = 2.54cm; введіть 2.54cm, і ви отримаєте повідомлення 2.54cm == 1in. Експериментуйте - це корисно.

Проблема полягає в тому, що ви не можете запобігти введення невірної інформації. Програма передбачає. що користувач завжди вводить правильні дані. Умова unit = 'i' відрізняє одиницю виміру 'i' від будь-яких інших варіантів. Вона ніколи не перевіряє його для одиниці виміру 'з'.

Що станеться. якщо користувач введе 15f (футів) "просто, щоб подивитися. що буде"? Умова (uni t - 'i') стане хибним, і програма виконає частину інструкції else (другу альтернативу). перетворюючи сантиметри в дюйми. Ймовірно. це не те, чого ви хотіли, вводячи символ 'f'.

Ми повинні завжди перевіряти вхідні дані програми, оскільки - вільно чи мимоволі - хто-небудь коли-небудь та введе невірні дані. Програма повинна працювати розумно. навіть якщо користувач так не надходить.

Наведемо поліпшену версію програми.

```
// Перетворення дюймів в сантиметри і навпаки
// Суфікс 'i' або 'з' означає одиницю виміру на вході,
// будь-який інший суфікс вважається помилкою

int main ()
```

```
{
constexpr double cm_per_inch ° 2.54; // кількість см в дюймі
double length = 1;          // довжина (дюйми або см)
char unit = '';         // пробіл - не одиниці виміру
cout << "Будь ласка, введіть довжину"
        << "і одиницю виміру (з або і): \ n";
cin >> length >> unit;
if (unit = 'i')
        cout << length << "in =="
        << cm_per_inch * length << "cm \ n";
else if (unit == 'з')
        cout << length << '' cm == "
        << length / cm_per_inch << "in \ n";
else
cout << "Вибачте, я не знаю, що таке '"
<< unit << '' '\ n' ';
}
```

Спочатку ми перевіряємо умова unit = 'i', а потім умова unit == 'c'. Якщо жодне з цих умов не виконується, виводиться повідомлення "Вибачте,. ..". Це виглядає так, ніби ви використовували інструкцію "else-if", але такої інструкції в мові C ++ немає. Замість цього ми використовували комбінацію двох інструкцій if. Загальний вигляд інструкції if виглядає так:

```
if (вираз) інструкція else інструкція
```

Інакше кажучи, за ключовим словом if слід вираз в дужках, а за ним - інструкція, ключове слово else і така інструкція. Ось як можна використовувати інструкцію if в частині else інструкції if:

```
if (вираз) інструкція else
if (вираз) інструкція else інструкція
У нашій програмі цей прийом використаний так:
if (unit == 'i')
. . . // 1-я альтернатива else if (unit == 'c')
. . . // 2-я альтернатива else
. . . // 3-тя альтернатива
```

Таким чином можна записати як завгодно складну перевірку і з | кожної альтернативою зв'язати свою інструкцію. Однак слід пам'ятати, що програма повинна бути простою, а не складною. Не варто демонструвати свою винахідливість, створював дуже складні програми. Краще доведіть свою

компетентність, написавши найпростішу програму, вирішальну поставлену задачу.

Інструкція switch

Порівняння значення unit з символами 'і' і 'з' являє собою найбільш поширену форму вибору: вибір, заснований на порівнянні значення з декількома константами. Такий вибір настільки часто зустрічається на практиці, що в мові C ++ для нього передбачена окрема інструкція: switch. Перепишемо наш приклад в іншому вигляді

```
int main ()
{
constexpr double cm_per_inch = 2.54; // Кількість см в дюймі
double length = 1; // довжина (дюйми або см)
char unit = 'a';
cout << "Будь ласка, введіть довжину"
<< "і одиницю виміру (з або і): \ n"; cin >> length >> unit; switch (unit) {case 'і':
cout << length << "in ="
<< cm_per_inch * length << "cm \ n"; break; case 'з':
cout << length << "cm ="
<< length / cm_per_inch << "in \ n"; break; default:
cout << "Вибачте, я не знаю, що таке '"
<< unit << " '\ n"; break;
}
}
```

Синтаксис оператора switch архаїчний, але він набагато ясніше вкладених інструкцій if, особливо якщо необхідно порівняти значення з багатьма константами. Значення, вказане в дужках після ключового слова switch, порівнюється з набором констант. Кожна константа представлена як частина мітки case. Якщо значення дорівнює константі в мітці case, то вибирається інструкція з даного розділу case. Кожен розділ case завершується ключовим словом break. Якщо значення не відповідає ні одній мітці case, то вибирається оператор, зазначений в розділі default. Цей розділ не обов'язковий, але бажаний, щоб гарантувати перебір всіх альтернатив. Якщо ви ще цього не знали, то врахуйте, що програмування привчає людину сумніватися практично у всьому.

Технічні подробиці інструкції switch:

1. Значення. яке визначає вибір варіанту. повинно мати цілочисельний тип, тип char або бути перерахуванням. Зокрема, вибір за значенням string зробити неможливо.
2. Значення міток розділів case повинні бути константними виразами. Зокрема. в мітках розділу case можна використовувати змінні.
3. Мітки двох розділів case не можуть мати однакові значення.

4. Один вибір може описуватися декількома мітками case.

5. Не забувайте завершувати кожен розділ case ключовим словом break. На жаль, компілятор попередить вас, якщо ви забудете про це.

Розглянемо приклад.

```
int main () // Вибір можна робити тільки
{       // по цілих чисел і т.п. типам даних
cout << "Ви любите рибу? \ n";
atrinq a;
cin >> a;
switch (a) {// Помилка: значення має бути цілим,
case "немає":       ! / Символом або перерахуванням
break; case "так":
break;
}
}
```

Інструкція switch генерує оптимізований код для порівняння значення з набором констант. Для великих множин констант він зазвичай створює більш ефективний код в порівнянні з набором інструкцій if. Однак це означає, що значення міток розділів case повинні бути константами і відрізнятися одна від одної. Розглянемо приклад.

```
int main ()                 // Мітки розділів case повинні бути константами
{                           // Визначаємо альтернативи:
int y = 'y';                // Це може привести до проблем
constexpr char n = 'n';
constexpr char m = '?';
cout << "Ви любите рибу? \ n";
char a;
cin >> a;
switch (a) {
        case n:
        // ...
        break;
case y:                     // Помилка: змінна в мітці case
        break;
case m:
        // ...
        break;
case 'n':       // Помилка: дублікат мітки case
                // (значення мітки n одно 'n')
//. . .
```

```
break;
default:
// ...
break;
}
}
```

Часто для різних значень інструкції switch доцільно виконати одну і ту саму дію. Було б утомливо повторювати цю дію для кожної мітки з такого набору. Розглянемо приклад.

```
int main ()    // Одна інструкція може мати кілька міток
{
cout << "Будь ласка, введіть цифру \ n";
char a;
cin >> a;
switch (a) {
case '0': case '2': case '4': case '6': case '8':
        cout << "парна \ n";
        break;
case '1': case '3': case '5': case '7': case '9':
cout << "непарна \ n";
break;
default:
cout << «не цифра \ n";
break;
}
}
```

Найчастіше, використовуючи інструкцію switch, програмісти забувають завершити розділ case ключовим словом break. Розглянемо приклад.

```
int main () // Приклад поганої програми (відсутній break)
{
constexpr double cm_per_inch = 2.54; // Кількість см в дюймі
double length = 1;                   // Довжина в дюймах або см
char unit = 'a';
cout << "Будь ласка, введіть довжину і"
        << "одиницю виміру (з або і): \ n";
cin >> length >> unit;

switch (unit) {
```

```
case 'i':
cout << length << "in =="
<< cm_per_inch * length << "cm \ n";
case 'з':
cout << length << "cm =="
<< length / cm_per_inch << "in \ n";
}
}
```

На жаль, компілятор прийме цей текст, і коли ви закінчите ви-конання розділу case з міткою 'i', просто "провалитеся" в розділ case з міткою 'c', так що при введенні рядка 2in програма виведе на екран наступні результати:

```
2in == 5.08cm
2 cm == 0.787402in
Ми вас попередили!
```

2.2.4. Ітерація

Ми рідко робимо що-небудь тільки один раз. З цієї причини в мовах програмування передбачені зручні засоби для багаторазового повторення дій. Ця процедура називається повторенням або (особливо, коли дії виконуються над послідовністю елементів в структурі даних) ітерацією.

Як приклад ітерації розглянемо першу програму, виконану на машині з програмою, що зберігається (EDSAC). Вона була написана Девідом Вілером (David Wheeler) в комп'ютерній лабораторії Кембриджського університету (Cambridge University, England) 6. травня 1949 року. Ця програма обчислює і роздруковує простий список квадратів.

```
0       0
1       1
2       4
3       9
4       16

98      9604
99      9801
```

Тут у кожному рядку міститься число, за яким слідують знак табуляції ('\ t') і квадрат цього числа. Версія цієї програми на мові C ++ виглядає так:

```
// Обчислюємо і роздруковуємо таблицю квадратів чисел 0-99
int main ()
```

```
(
int i = 0; // Починаємо з нуля
while (i <100) {
        cout << i << '\ t' << square (i) << '\ n';
        ++ i // Інкремент i (i стає рівним i + l)
}
}
```

Позначення square (i) означає квадрат числа i.

Ні, насправді перша сучасна програма не була написана на мові C ++, але її логіка була такою ж.

- Обчислення починаються з нуля.
- Перевіряємо, чи не досягли ми числа 100, і якщо досягли. то завершуємо обчислення.
- В іншому випадку виводимо числоі його квадрат, розділені символом табуляції ('\ t'). збільшуємо число і повторюємо обчислення.

Очевидно, що для цього нам необхідно наступне.

- Спосіб повторного виконання інструкції (цикл).
- Змінна, за допомогою якої можна було б відслідковувати кількість повторень інструкції в циклі (лічильник циклу. або керуюча змінна). У даній програмі вона має тип int і називається i.
- Початкове значення лічильника циклу (в даному випадку - 0).
- Критерій припинення обчислений (в даному випадку ми хочемо виконати зведення в квадрат 100 раз).
- Щось, що виконується в циклі (співало циклу).

У даній програмі ми використовували інструкцію while. Відразу за ключовим словом while слід умова, а потім тіло циклу.

```
while (i <100) // Умова циклу, що перевіряє значення лічильника i
{
cout << i << '\ t' << aquaxe (i) << '\ n';
// Інкремент i (1 стає рівним i + l)
}
```

Тіло циклу - це блок (укладений у фігурні дужки). який роздруковує таблицю і збільшує лічильник циклу і на одиницю. Кожне повторення циклу починається з перевірки умови i <100. Якщо ця умова істинно, то ми не закінчуємо обчислення і продовжуємо виконувати тіло циклу. Якщо ж ми досягли кінця, тобто змінна і дорівнює 100, то виходимо з інструкції while і виконуємо інструкцію, наступну за нею. У цій програмі після виходу з циклу програма закінчує роботу, тому ми з неї виходимо.

Лічильник циклу для інструкції while повинен бути визначений і проініцівлізовано заздалегідь, поза інструкції while. Якщо ми забудемо це

зробити, то компілятор видасть повідомлення про помилку. Якщо ми визначимо лічильник циклу. Але забудемо його проініціалізувати, то більшість компіляторів попередять про це (повідомивши щось на зразок "локальна змінна і не встановлена"), але не стануть перешкоджати виконанню програми. Чи не пропускайте такі попередження! Компілятори практично ніколи не помиляються, якщо справа стосується неініціалізованих змінних. Такі змінні часто стають джерелом помилок. В цьому випадку слід написати

```
int i = 0; // Починаємо обчислення з нуля
і все буде добре.
```

Як правило, написання циклів не викликає ускладнень. Проте при вирішенні реальних завдань ця задача може виявитися складним. Зокрема, іноді складно правильно висловити умова і проініціалізувати всі змінні так, щоб цикл був коректним.

Блоки.

Зверніть увагу на те, як ми згрупували дві інструкції, мають бути виконанні.

```
while (i <100) {
cout << i << '\ t' << square (i) << '\ n';
++ i;            // Інкремент i (i стає рівним i +1)
}
```

Послідовність інструкцій, укладених у фігурні дужки ({i}). називається блоком або складовою інструкцією. Блок - це різновид інструкції. Порожній блок {} іноді виявляється корисним для вираження того, що в даному місці програми не слід нічого робити. Розглянемо приклад.

```
if (a <= b) {// Нічого не робимо}
else {  // Міняємо місцями a й b
int t = a;
a = b;
b = t;
}
```

Інструкція for

Ітерація над послідовностями чисел настільки часто використовується в мові С ++. як і в інших мовах програмування. що для цієї операції передбачена спеціальна синтаксична конструкція. Інструкція for схожа на інструкцію while. за винятком того. що управління лічильником циклу зосереджено в його початку, де за ним легко стежити і розпізнавати. "Першопрограму" можна переписати так:

```
// Обчислюємо і роздруковуємо таблицю квадратів чисел 0-99
int main ()
{
for (int i = 0; i <1;
cout << i << '\ t' << square (i) << '\ n';
}
```

Це означає "Виконати тіло циклу, починаючи зі змінною і, рівною нулю, і збільшувати її на одиницю при кожному виконанні тіла циклу. Поки змінна і не досягне значення 100". Інструкція for завжди еквівалентна якоїсь інструкції while. В даному випадку конструкція

```
for (int i = 0; i <100; ++ i)
cout << i << '\ t' << square (i) << '\ n';
```

еквівалентна коду

```
{
int = 0; // Ініціалізатор інструкції for
while (i <100) {// Умова інструкції for
                        // Тіло інструкції for
cout << i << '\ t' << square (i) << '\ n';
++ i // Інкремент інструкції for
}
}
```

Деякі новачки вважають за краще використовувати інструкцію while, а не інструкцію for. Однак за допомогою інструкції for можна створити набагато більш ясний код. оскільки цикл for містить прості операції ініціалізації, перевірки умови і інкремента лічильника. Використовуйте інструкцію while тільки тоді, коли немає іншого виходу.

Ніколи не змінюйте лічильник циклу в тілі інструкції for. Це порушить всі розумні припущення читача програми про зміст циклу. Розглянемо приклад.

```
int main () {
for (int i = 0; i <100;        {// Для і з діапазону [0,100)
cout << i << '\ t' << square (i) << '\ n';
++ i; // Що це? Схоже на помилку. '
}
}
```

Будь-який читач. побачив цей цикл. розумно припустить, що його тіло буде виконано 100 раз. Однак це не так. Інструкція ++ i в його тілі забезпечує

подвійний інкремент лічильника і на кожній ітерації, так що висновок буде здійснений тільки для 50 парних чисел. Побачивши такий код, ви можете припустити. що це помилка. викликана некоректним перетворенням інструкції for з інструкції while. Якщо ви дійсно хочете, щоб лічильник збільшувався на 2, напишіть наступне:

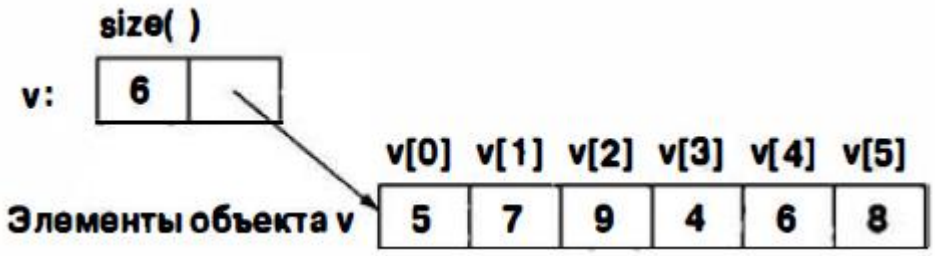
```
// Обчислюємо і виводимо на друк таблицю квадратів
// парних чисел з діапазону [0,100)
int main () {
for (int i = 0; i <100; i + = 2)
cout << i << '\ t' << square (i) << '\ n';
}
```

Будь ласка, не забувайте, що ясна і проста програма коротше заплутаною. Це загальне правило.

2.2.5. Вектор

Для того щоб програма робила щось корисне, необхідно зберігати колекцію даних, з якими вона працює. Наприклад. нам може знадобитися список телефонних номерів, список гравців футбольної команди. список книг. прочитаних в минулому році. список курсів, графік платежів за автомобіль. список прогнозів погоди на наступний тиждень, список цін на фотокамеру в інтернет-магазині, тощо. Цей перелік можна продовжувати до нескінченності, а тому і в програмах ці списки зустрічаються дуже часто. Поки почнемо з найпростішого і, мабуть. найбільш корисного способу зберігання даних: за допомогою вектора vector.

Вектор - це просто послідовність елементів, до яких можна звертатися за індексом. Наприклад, розглянемо об'єкт типу vector з ім'ям v.



Перший елемент вектора має індекс, що дорівнює 0; другий елемент - індекс 1, тощо. Ми звертаємося до елементу, вказуючи ім'я вектора і індекс елемента в квадратних дужках, так що значення v [0] дорівнює 5, значення v [1] дорівнює 7, тощо. Індекси вектора завжди починаються з нуля і збільшуються на одиницю. Це вам повинно бути знайоме: вектор зі стандартної бібліотеки C ++ - це просто новий варіант старої і добре відомої ідеї. Уявімо вектор так, як

показано на малюнку, щоб підкреслити, що вектор "знає свій розмір", тобто завжди зберігає його в одній з комірок.

Такий вектор можна створити, наприклад, так:

```
vector <int> v = {5, 7, 9, 4, 6, 8}; // Вектор з 6 цілих чисел
```

Як бачимо, для того щоб створити вектор, необхідно вказати тип його елементів і їх початкові значення. Тип елементів вектора вказується після слова vector в кутових дужках (<>). в даному випадку це тип int. Ось ще один приклад.

```
vector <string> philosopher // Вектор з чотирьох рядків
= { "Kant", "Plato", "Hume", "Kierkegaard"};
```

Природно, в векторі можна зберігати елементи тільки одного, оголошеного типу.

```
philoaopher [2] = 99; // Помилка: присвоювання цілого числа рядку
v [2] = "Hume";      // Помилка: присвоювання рядки цілому числу
```

Ми можемо також визначити vector заданого розміру, які не вказуючи значення його елементів. У цьому випадку використовується запис (n), де n - кількість елементів, а елементи отримують значення за замовчуванням для даного типу. наприклад:

```
vector <int> vi (6); // vector з 6 int, ініціалізованих 0
vector <string> vs (4); // vector з 4 рядків, ініціалізованих ""
```

Рядок без символів ("") називається символом нового рядка.

Зверніть увагу, що ми не можемо просто звернутися до неіснуючого елементу вектора:

```
vd [20000] = 4.7; // Помилка часу виконання програми
```

Обхід вектора.

vector "знає" свій розмір, так що ми можемо вивести елементи вектора наступним чином:

```
vector <int> v = {5, 7, 9, 4, 6, 8};
for (int i = 0; i <v.size (); cout << v [i] << '\ n';
```

Виклик v.size () дає кількість елементів у векторі v. У загальному випадку v.size() дає нам можливість звернення до елементів vector без ризику випадкового звернення до елементу за межами вектора. Діапазон індексів елементів вектора v є [0, v.size ()). Це математична запис напіввідкритій послідовності елементів.

Першим елементом v є v [0], а останнім - v [v.size () - 1]. Якщо v.size () = 0, v не має елементів, тобто являє собою порожній вектор.

Мова програмування використовує перевагу напіввідкритій послідовності для простого циклу по всіх елементах послідовності, такий, як елементи вектора, наприклад:

```
vector <int> v = {5, 7, 9, 4, 6, 8};
for (int x: v)          // Для всіх x з v
cout << x << '\ n';
```

Такий цикл називається циклом for за діапазоном, так як слово діапазон (range) часто використовується для позначення послідовності елементів. Ми читаємо for (int x: v) як "для кожного int x в v", і цей цикл в точності представляє собою цикл по всіх індексах з діапазону [0, v.size ()). Такий цикл за діапазоном використовується для простого проходу по всіх елементах послідовності по одному елементу за раз. Більш складні завдання на зразок проходу по кожному третьому елементу вектора або тільки по другій половині діапазону або порівняння елементів двох векторів вирішуються за допомогою більш складного і більш загального традиційного циклу for.

Збільшення вектора.

Часто ми починаємо роботу з порожнім вектором і збільшуємо його розмір у міру зчитування або обчислення даних. Ключовий операцією є push_back (), що додає в вектор новий елемент, який стає останнім елементом вектора. Розглянемо приклад

.

```
vector <double> v;// Починаємо з пустого вектора,
                          // V не містить жодного елемента

v.push_back (2.7);        // Додаємо в кінець v елемент 2.7
                          // Тепер v містить один елемент v [0] == 2. 7

v.push_back (5.6);        // Додаємо в кінець v елемент 5.6
                          // Тепер v містить два елементи і v [1] == 5.6

v.push_back (7.9)         // Додаємо в кінець v елемент 7.9
                          // Тепер v містить три елементи і v [2] == 7.9
```

Зверніть увагу на синтаксис виклику push_back (). Він називається визовом функції-члена; push_back () є функцією-членом об'єкта типу vector, і тому для її виклику використовується форма виклику з точкою.

виклик функції-члена:

ім'я_об'єкта. ім'я.функції.члена (список_аргументов)

Розмір вектора можна визначити, викликавши іншу функцію-член об'єкта типу vector - size (). Спочатку значення, що повертається v.size (), дорівнює 0, а після третього виклику функції push_back () це значення дорівнює 3.

Якщо ви маєте досвід програмування, то можете помітити, що тип vector схожий на масив в мові C та іншими мовами програмування. Однак вам не потрібно заздалегідь вказувати розмір (довжину) вектора, і ви можете додавати в нього елементи в міру необхідності. Надалі ми переконаємося, що тип vector зі стандартної бібліотеки C ++ володіє і іншими корисними властивостями.

Числовий приклад.

Розглянемо більш реалістичний приклад. Часто у нас є ряд значень, які ми зчитуємо в програму. щоб потім щось з ними зробити. Це "щось" може означати побудова графіка, обчислення середнього і медіани, пошук найбільшого значення, сортування, змішування з іншими даними, пошук цікавлять нас значень, порівняння з іншими даними і т.п. Перераховувати операції з даними можна нескінченно, але спочатку дані необхідно вважати в пам'ять комп'ютера.

Розглянемо основний спосіб введення невідомого - можливо, великого - обсягу даних. Як конкретний приклад спробуємо вважати числа з плаваючою точкою, що представляють собою значення температури.

```
// Прочитуємо значення температури в вектор
int main ()
{
vector <dovble> temps;                  // Температури
for (double temp; cin >> temp;}         // Читання в temp
temps.push_back (temp);                 // Вносимо temp в vector

// ... Якісь дії. . .
}
```

Отже, що ж відбувається в цьому фрагменті програми? Спочатку ми оголошуємо вектор для зберігання даних:

```
vector <double> temps;  // температури
```

Тут вказано тип вхідних даних. Ми зчитуємо і зберігаємо значення типу double.

Тепер виконується сам цикл зчитування.

```
for (double temp; cin >> temp; // Читання в temp
temps.push_back (temp); // Вносимо temp в vector
```

Ми визначили змінну temp типу double для зчитування значень. Інструкція cin >> temp зчитує значення типу double, а потім це значення вноситься в вектор (записується в його кінець). Ці операції вже були продемонстровані вище. Новизна полягає в тому, що в якості умови виходу з циклу for ми використовуємо операцію введення cin >> temp. В основному умова cin >>temp є істинним, якщо значення лічено коректно, в іншому випадку воно є хибним, так що в циклі for зчитуються всі числа типу double, поки на вхід не надійде щось інше. Наприклад. якщо ми подамо на вхід дані

1.2 3.4 5.6 7.8 9.0

то в вектор temps будуть занесені п'ять елементів: 1.2. 3.4. 5.6, 7.8, 9.0 (саме в зазначеному порядку, наприклад temps [0] = 1.2). Для припинення введення використовується символ "1", тобто значення. яка не є double.

Щоб обмежити область видимості вхідної змінної temp циклом, ми використовуємо цикл for, а не цикл while:

```
double temp;
while (cin >> temp)                 // Читання
temps.push_back (temp); // Внесення в вектор
// Змінна temp може бути використана після циклу
```

Як завжди, в циклі for все, що повинно відбуватися, показано в заголовку, так що такий код легше зрозуміти, і важче допустити в ньому випадкову помилку.

Записавши дані в вектор, ми можемо легко з ними працювати. Як приклад обчислимо середнє і медіану значень температур.

```
// Обчислюємо середнє і медіану значень температур
int main ()
{
vector <double> temps;                  // Температури
for (double temp; cin »temp;)           // Читання в temp
temps.push_back (temp);                 // Внесення temp в вектор

// Обчислення середньої температури:
double sum = 0; for (int x: temps) sum + = x;
cout << "Середня температура:" << sum / temps.size () << i \ n ';

// Обчислення медіани температури:
sort (temps);          // Сортування вектора температур
cout << "Медианная температура:"
<< temps [temps.size () / 2] << '\ n';
```

```
}
```

Ми обчислюємо середнє значення, просто підсумовуючи все елементи і ділячи суму на кількість елементів (тобто на значення temps. Size ()).

```
// Обчислення середньої температури:
double sum = O;
for (int x: temps)
sum + = x;
cout << "Середня температура:" << sum / temps.size () << '\ n';
```

Зверніть увагу, наскільки зручним виявився оператор + =.

Для обчислення медіани (значення, щодо якого половина всіх значень виявляється менше, в інша половина - більше) елементи слід упорядкувати. Для цієї мети використовується варіант алгоритму сортування із стандартної бібліотеки sort ().

```
// Обчислення медіани температури:
sort (temps);                    // Сортування вектора температур
cout << "Медианная температура:"
<< temps [temps.size () / 2] << '\ n';
```

Цей алгоритм сортування буде розглянуто набагато пізніше. Після сортування значень температури знайти медіану дуже легко: ми просто знаходимо середній елемент, тобто елемент з індексом temps. size () / 2. Якщо проявити певну прискіпливість (характерну для програмістів), то можна помітити, що знайдене нами значення може не виявитися медіаною в відповідності із суворим визначенням.

Текстовий приклад.

Наведений вище приклад цікавий нам з загальної точки зору. Зрозуміло, середнє значення і медіана температури цікавлять багатьох людей - метеорологів, аграріїв і океанографів, - але нам, програмістам, важлива узагальненість прикладу - то, що вектор і прості операції можуть застосовуватися у величезній кількості найрізноманітніших додатків. Можна сказати, що будь-які дані нас не цікавили, при необхідності їх аналізу нам необхідний вектор. Як приклад створимо простий словник.

```
// Простий словник: список упорядкованих слів
int main ()
{
vector <string> words;
for (string temp; cin >> temp;) // Читання слів, розділених
                                // пробільними символами
```

```
words.push_hack (temp); // Внесення їх у вектор
cout << "Кількість слів:" << words.size () << '\ n';
sort (words);          // Сортування слів
for (int i = 0; i <words.size (); ++ i)
if (i = 0 | | words [i-1]! = words [i]) // Це нове слово?
cout << words [i] << "\ n";
}
```

Якщо в цю програму ввести кілька слів, то вона виведе їх в алфавітному порядку без повторів. Наприклад, припустимо, що в програму вводяться слова

```
a man a plan a canal panama
```

У відповідь програма виведе на екран наступні слова:

```
a
canal
man
panama
plan
```

Як зупинити читання рядки? Інакше кажучи, як припинити цикл введення?

```
for (string temp; cin >> temp;} // Читання слів
words.push_pa      ck (temp); // Внесення їх у вектор
```

Коли ми зчитували числа для припинення введення просто вводили якийсь символ, який не був числом. Однак для рядків цей прийом не працює, так як в рядок може бути лічений будь (звичайний) символ. На щастя, існують символи, які не є "звичайними".

Велика частина даної програми дивно схожа на програму для роботи з температурою. Фактично ми написали "словникову програму" методом вирізання і вставки (копіювання фрагментів коду) з "температурної програми". Єдиною новою інструкцією є перевірка

```
if (i == 0 | | words [il]! = words [i-1]) // Це нове слово?
```

Якщо видалити цю перевірку з програми. то висновок стане іншим:

```
a
a
a
canal
```

man
panama
plan

Ми не любимо повторень, тому видаляємо їх з допомогою даної перевірки. Що вона робить? Вона з'ясовує, чи відрізняється попереднє слово від знову виїденого (words [i-1]! = words [i]), і якщо відрізняється, то слово виводиться на екран, а якщо немає, то не виводиться. Очевидно, що у першого слова попередника немає (i == 0), тому спочатку слід перевірити номер слова і об'єднати ці перевірки за допомогою оператора 1 + 1 (або).

if (i = = 0 || words [i-1]! = words [i]) // Це нове слово?

Зверніть увагу, що ми можемо порівнювати рядки. Для цього ми використовуємо оператори ! = (Не дорівнює), == (дорівнює), < (менше), <= (менше або дорівнює), > (більше) і > = (більше або дорівнює), які можна застосовувати і до рядків. Оператори <. > Та інші використовують звичайний лексикографічний порядок, так що рядок "Apі" передує рядках "Apple" і "Chimpanzee".

2.3. Задачі для розв'язання

Виконайте завдання крок за кроком. Не слід поспішати і пропускати етапи. На кожному етапі перевірте програму, ввівши принаймні три пари значень - чим більше, тим краще.

1. Напишіть програму, що містить цикл while, в якому зчитуються і виводяться на екран два числа типу int. Для виходу з програми використовуйте символ "1".
2. Змініть програму так, щоб вона виводила на екран рядок "Найменше значення рівне:" з подальшим найменшим значенням, а потім - рядок "Найбільше значення рівне:" з подальшим максимальним значенням.
3. Виправите програму так, щоб вона виводила рядок "Числа рівні", але тільки за однакової кількості введених чисел.
4. Змініть програму так, щоб вона працювала з числами типу double, а не int.
5. Змініть програму так. щоб вона виводила рядок "Числа майже рівні", якщо числа відрізняються одне від одного менше ніж на 1.0 / 100.
6. Тепер змініть тіло циклу так. щоб він зчитував тільки по одному числу типу double в кожній ітерації. Визначте дві змінні, щоб відстежувати найменше та найбільше серед усіх раніше введених значень. У кожній ітерації циклу виводите тільки що введене число. Якщо воно виявиться найменшим серед всіх введених, виводите на екран рядок "Найменше серед введених". Якщо

ж воно виявиться найбільшим серед введених, виводите на екран рядок "Найбільше серед введених".

7. Додайте до кожного введеного числа типу double одиницю виміру; інакше кажучи, вводите такі значення, як 10cm, 2.5in, 5ft або 3.33m. Допустимими є чотири одиниці виміру: cm, m, in, ft. Прийміть наступні коефіцієнти перетворення: 1m = 10cm, 1in = 2.54cm, 1ft == 12in. Індикатор одиниці виміру вводите в рядок. Можна вважати введення 12 m (з пропуском між числом і одиницею виміру) еквівалентним введенню 12m (без пробілу).

8. Якщо введена неправильна одиниця виміру, наприклад yard, meter, km або gallons, то таке значення слід відхилити.

9. Відстежуйте суму введених значень (крім найменшого і найбільшого) і їх кількість. Коли цикл завершиться, виведіть на екран найменше введене значення, найбільше введене значення, кількість введених значень і їх суму. Зверніть увагу на те, що накопичуючи суму, ви повинні вибрати для неї одиницю виміру (використовуйте метри).

10. Зберігайте всі введені значення (перетворені в метри) у векторі, а по завершенні роботи циклу виведіть їх на екран.

11. Перед тим як вивести значення з вектора, відсортуйте їх в порядку зростання.

12. Припустимо, ми визначаємо медіану послідовності як "число, щодо якого рівно половина елементів менше, а інша половина - більше". Підказка: медіана не зобов'язана бути елементом послідовності.

14. Прочитайте послідовності чисел типу double в вектор. Будемо вважати, що кожне значення є відстань між двома містами, розташованими на певному маршруті. Обчисліть і виведіть на друк загальна відстань (суму всіх відстаней). Знайдіть і виведіть на друк найменше та найбільше відстані між двома сусідніми містами. Знайдіть і виведіть на друк середня відстань між двома сусідніми містами.

15. Напишіть програму, вгадувати число. Користувач повинен задумати число від 1 до 100, а програма повинна задавати питання, щоб з'ясувати, яка кількість він задумав (наприклад, "Задумане число менше 50"). Ваша програма повинна вміти ідентифікувати за допомогою не більше ніж семи спроб. Примітка: використовуйте оператори <i <=. а також конструкцію if-else.

16. Напишіть програму, що виконує найпростіші функції калькулятора. Ваш калькулятор повинен виконувати чотири основні арифметичні операції - додавання, віднімання, множення і ділення. Програма повинна пропонувати користувачеві ввести три аргументи: два значення типу double і символ операції. Якщо вхідні аргументи рівні 35.6, 24.1 і '+', то програма повинна вивести на екран рядок "Сума 35.6 і 24.1 дорівнює 59.7".

17. Створіть вектор, який зберігає десять строкових значень "zero", "one",…,"nine". Використовуйте його в програмі, перетворюючої цифру в відповідне строкове представлення: наприклад, при введенні цифри 7 на екран повинна бути виведена рядок seven. За допомогою цієї ж програми, використовуючи той же цикл введення, перетворіть строкове представлення

цифри в числове: наприклад, при введенні рядка seven на екран повинна бути виведена цифра 7.

18. Модифікуйте міні-калькулятор, описаний в задачі 5, так. щоб він приймав на вхід цифри, що записані як в числовому, так і в строковому форматі.

19. Легенда свідчить, що якийсь цар захотів подякувати винахідника шахів і запропонував йому попросити будь-яку нагороду. Винахідник попросив покласти на першу клітку одне зерно рису. на другу - два. на третю - чотири і так далі, подвоюючи кількість зерен на кожній з 64 клітин. На перший погляд. це бажання виглядає цілком скромним. але насправді в царстві не було такої кількості рису! Напишіть програму, що обчислює, скільки клітин треба заповнити, щоб збретатель отримав не менше 1 000 зерен рису, не менше 1 000 000 зерен рису і не менше 1 000 000 000 зерен рису. Вам, зрозуміло. знадобиться цикл і. ймовірно, кілька змінних типу int. для того щоб відстежувати поточний номер клітини. кількість зерен в поточній клітці і кількість зерен у всіх попередніх клітинах. Ми пропонуємо на кожній ітерації циклу виводити на екран значення всіх цих змінних, щоб бачити проміжні результати.

20. Спробуйте обчислити кількість зерен рису, запитаних винахідником шахів в задачі 8. Виявляється. що це число настільки велике. що для його точного уявлення не підходить ні тип int, ні тип double. Чому дорівнює найбільшу кількість клітин. для яких ще можна обчислити точну кількість зерен рису (з використанням змінної типу int). Визначте найбільшу кількість клітин. для яких ще можна обчислити наближене кількість зерен (з використанням змінної типу double)?

21. Напишіть програму для гри "Камінь. Папір. Ножиці ". Якщо ви не знаєте правил цієї гри. Спробуйте з'ясувати їх у друзів або за допомогою Google (пошук інформації - звичайне заняття програмістів). Для вирішення поставленого завдання використовуйте інструкцію switch. Крім того, машина повинна давати випадкові відповіді (тобто вибирати камінь, папір або ножиці на наступному ходу випадковим чином). Написати справжній генератор випадкових чисел прямо зараз вам буде досить важко. тому заздалегідь заповніть вектор послідовністю значень. Якщо вмонтувати цей вектор в програму. то вона завжди буде грати одну і ту ж гру. тому доцільно дозволити користувачеві самому вводити деякі значення. Спробуйте перешкодити користувачеві легко вгадувати наступний хід машини.

22. Напишіть програму. знаходить все прості числа між 1 і 100. Для цього можна написати функцію. перевіряє. чи є число простим (тобто ділиться воно на просте число, яке не перевищує дане). використовуючи вектор простих чисел, записаний в порядку зростання (наприклад, якщо вектор називається primes, то primes [0] = 2, primes [1] = 3, primes [2] = 5 і т.д.). Напишіть цикл перебору чисел від 1 до 100. перевірте кожне з них на простоту і збережіть знайдені прості числа у векторі. Напишіть інший цикл. в якому всі знайдені прості числа виводяться на екран. Порівняйте отримані результати з вектором primes. Першим простим числом вважається число 2.

23. Змініть програму з попередньої вправи так, щоб в неї вводилося число max, а потім знайдіть все прості числа від 1 до max.

24. Напишіть програму. знаходить все прості числа між 1 і 100. Для вирішення цього завдання існує класичний метод "Решето Ератосфена". Якщо цей метод вам невідомий, пошукайте його опис в Інтернеті. Напишіть програму. використовує цей метод.

25. Змініть програму. описану в попередній вправі, так, щоб в неї вводилося число max, а потім знайдіть все прості числа від 1 до max.

26. Напишіть програму, приймаючу на вхід число n і знаходить перші n простих чисел.

27. У завданнях вам було запропоновано написати програму. яка для певного ряду чисел визначала б найбільше та найменше числа в ньому. Число, яке повторюється в послідовності найбільшу кількість разів, називається модою. Напишіть програму. визначальну моду безлічі позитивних чисел.

28. Напишіть програму, яка визначить найменше та найбільше значення, а також моду послідовності рядків типу string.

29. Напишіть програму для вирішення квадратних рівнянь. Квадратне рівняння має вигляд $ax^2 + bx + c = 0$. Якщо ви не знаєте формул для вирішення такого рівняння, проведіть додаткові дослідження. Нагадаємо, що програмісти часто виконують такі дослідження, перш ніж приступити до вирішення завдання. Для введення чисел a, b і c використовуйте змінні типу double. Оскільки квадратне рівняння має два рішення, виведіть обидва значення, x1 і x2.

30. Напишіть програму, в яку спочатку вводиться набір пар, що складаються з імені та значення, наприклад Joe 17 і Barbara 22. Для кожної пари занесіть ім'я в вектор names, а число - в вектор scores (в відповідні позиції, так що якщо names [7] == "Joe", то scores [7] = 17). Припиніть введення, зустрівши рядок введення NoName 0. Переконайтеся, що кожне ім'я єдине, і виведіть повідомлення про помилку, якщо є ім'я, введене двічі. Виведіть на друк всі пари (ім’я, бали) по одному в рядку.

Практичне заняття 3. Вказівники, масиви та посилання.

3.1. Мета практичного заняття

Вивчити технології програмування вказівників, масивів та посилань, їх особливості використання при вирішенні практичних задач та побудові оптимального коду програми.

3.2. Теоретичний матеріал

3.2.1. Вказівники

Вказівник - змінна, значенням якої є адреса іншої змінної. Якщо звичайна змінна дозволяє отримати доступ до області пам'яті на ім'я змінної, то вказівник дозволяє отримати доступ до області пам'яті за адресою. Ці два способи звернення до однієї і тієї ж області пам'яті можуть комбінуватися і використовуватися одночасно. У розглянутому далі прикладі створюються дві змінні, але доступ до них для присвоювання значень виконується за допомогою показників.

Вказівник оголошується так: задається тип значення, на яке він може вказувати, а після ідентифікатора типу розміщується зірочка *. Далі традиційно слідує ім'я вказівника. Щоб отримати адресу змінної (звичайну, не вказівника), перед ім'ям змінної розміщують &. Щоб отримати значення змінної, адреса якої записаний у вказівнику, перед ім'ям вказівника ставлять *.

Розглянемо невеликий програмний код, в якому використовуються вказівники.

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main () {
// Зміна кодування консолі:
system ( "chcp тисяча двісті п'ятьдесят один> nul");
// Символьна змінна:
char symb;
// Цілочисельна змінна:
int num;
// Вказівник на символьне значення:
char * p;
// Вказівник на цілочисельне значення:
int * q;
// Значення вкзівника p - адреса змінної symb:
p = & symb;
```

```
// Значення вказівника q - адреса змінної num: q = Snum;
// Значення змінної symb присвоюється
// через вказівник на неї:
* P = 'A'
// Значення змінної num присвоюється
// через вказівник на неї:
* Q = 100;
// Перевіряється значення змінної symb:
cout << "symb =" << symb << endl;
// Перевіряється значення змінної num:
cout << "num =" << num << endl;
// Затримка консольного вікна:
system ( "pause> nul");
return 0;
}
```

Командами char symb і int num в програмі відповідно оголошуються символьна змінна symb (значенням може бути символ - одиночний символ полягає в одинарні лапки) і целочисельна змінна num.

Вказівник p на символьне значення оголошується командою char * p. Вказівник p, таким чином, може містити значенням адреса змінної типу char. Значення вказівником p присвоюється командою p = & symb, так що в результаті значенням вказівника p є адреса змінної symb. Аналогічні справи з вказівником q на цілочисельне значення. Вказівник оголошується командою int * q, а значення вказівником присвоюється командою q = & num, в результаті чого значенням вказівника q є адреса змінної num.

Значення змінним symb і num присвоюються через вказівники. Значення змінної symb присвоюється командою *p = 'A'. Значення змінної num присвоюється командою *q = 100. Ми врахували, що для доступу до області пам'яті, на яку встановлено вказівник, перед ім'ям вказівника слід поставити зірочку *. Після присвоювання значень змінним symb і num ми перевіряємо значення змінних командами cout << "symb =" << symb << endl і cout << "num =" << num << endl;, що містять явне звернення до змінних symb і num.

3.2.2. Масиви та вказівники

Є кілька важливих фактів з приводу масивів (ми поки обговорюємо одномірні статичні масиви) і вказівників. Коротенько ось вони:

- При оголошенні масиву місце під егпро елементи в пам'яті виділяється так, що елементи розташовані один за іншим.
- Ім'я масиву є вказівником на його перший елемент.
- До вказівниками можна додавати і віднімати від них цілі числа. Результатом є вказівник, зміщений від поточного вказівника на відповідне кількість осередків пам'яті (в напрямку збільшення або зменшення адреси).

- Різницею двох вказівників (одного типу) є ціле число, яке визначає кількість позицій (осередків) між осередками пам'яті, на які посилаються вказівники.
- Вказівники можна індексувати (індекс вказується в квадратних дужках після імені вказівника і формально може бути від'ємним). Значення індексованого вказівника - це значення в осередку, яка відстоїть від даного вказівника на кількість позицій, яке визначається індексом.

Резюме досить просте: звернення до елементу масиву представляє собою операцію індексування вказівника.

Далі буде представлена програма, в якій створюється символьний масив і заповнюється значеннями. При цьому в значній мірі використовуються вказівники.

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main () {
// Зміна кодування консолі:
system ( "chcp тисяча двісті п'ятьдесят один> nul");
// Розмір масиву:
const int size = 12;
// Індексний змінна:
int k;
// Створення символьного масиву:
char symbs [size];
// Вказівники на символьні значення:
char * p; char * q;
// Вказівник на перший елемент масиву:
p = symbs;
// Значення першого елемента масиву:
p [0] = 'A'
// Вказівник на останній елемент масиву:
q = & symbs [size-l];
// Кількість позицій між першим
// і останнім елементами масиву:
cout << "Між першим і останнім елементами" << qp << "позицій \ n";
// Заповнення масиву значеннями:
while (p! = q) {
// Вказівник на наступний елемент:
p ++;
// Обчислення значення елемента масиву:
* P = p [-1] +1;
}
```

```
cout << "Елементи масиву \ n |";
// Відображення вмісту масиву:
for (k = 0; k <size; k ++) {
cout << symbs [k] «" | ";
}
Cout << "\ n Елементи в зворотному порядку \ n |";
// Відображення вмісту масиву в зворотному порядку:
 for (k = 0; k <size; k ++) {cout << q [-k] << "|";
}
cout << endl;
// Затримка консольного вікна:
system ( "pause> nul");
return 0;
}
```

У програмі оголошується цілочисельна константа size, яка визначає розмір символьного масиву, який створюється командою char symbs [size]. Також в програмі оголошуються два символьних вказівника p і q (команди char * p і char * q). Вказівником p значення присвоюється командою p = symbs. Оскільки ім'я масиву є посиланням на перший елемент масиву, то після виконання даної команди вказівник p посилається на перший елемент масиву symbs. Командою p [0] = 'A' першому елементу масиву symbs присвоюється значення 'A'. Так відбувається в силу правила індексування вказівників: p [0] є значення області пам'яті, яка відстоїть на 0 позицій від осередку, на яку посилається вказівник p. Оскільки на момент виконання програми вказівник p містить значенням адресу першого елемента масиву, то саме цей елемент отримує значення 'A'.

Значення вказівником q присвоюється командою q = ssymbs [size-l]. Оскільки symbs [size-1] є ні що інше, як останній елемент масиву symbs, то & symbs [size-1] - адреса останнього елемента масиву. Таким чином, значенням вказівника q є адреса останнього елемента масиву. Якщо обчислити різницю вказівників qp, то результатом отримаємо кількість позицій між останнім і першим елементами масиву. Дане значення на одиницю менше довжини масиву (і в даному конкретному випадку дорівнює 11 оскільки в масиві 12 елементів).

Для заповнення масиву значеннями запускається оператор циклу while. В операторі перевіряється умова p! = Q, яке полягає в тому, що значення вказівників p і q різні. У тілі оператора циклу командою p ++ операція інкремента застосовується до вказівника p. Дана команда еквівалентна команді p = p + 1. Додаток до вказівника одиниці дає в результаті вказівник, який встановлений на сусідню клітинку. Оскільки в масиві елементи в пам'яті розташовані підряд один за іншим, то після виконання команди p ++ вказівник p посилатиметься на наступний елемент в масиві. Командою *p = p [-1] +1 обчислюється значення елемента масиву. У лівій частині команди варто вираз *p, яке є елементом

масиву, на який встановлено вказівник p. Вираз p [-1] +1 в правій частині містить інструкцію p [-1]. Це значення елемента, який знаходиться на одну позицію назад (в напрямку зменшення адреси) по відношенню до елементу, на який встановлено вказівник p. Простіше кажучи, для визначення значення елемента, на який встановлено вказівник p, береться значення попереднього елемента і до нього додається одиниця.

3.2.3. Посилання

Для ілюстрації особливостей використання посилань розглянемо зовсім невеличкий приклад, наведений нижче.

```
#nclude <iostream>
#include <cstdlib>
using namespace std;
int main () {
// Зміна кодування консолі:
system (chcp тисяча двісті п'ятьдесят один> nul ");
// целочисленная змінна:
int num;
// Посилання на змінну:
int & ref = num;
// Присвоєння значення змінної:
num = 100;
// Перевірка значення змінної і посилання:
cout << "num =" << num << endl;
cout << "ref =" << ref << endl;
// Присвоєння значення посиланням:
ref = 200;
// Перевірка значення змінної і посилання:
cout << "num =" << num << endl;
cout << "ref =" << ref << endl;
// Затримка консольного вікна:
system ( "pause> nul");
return 0;
}
```

У програмі оголошується цілочисльна змінна num, а також ще одна змінна, яка називається ref. Оголошення і ініціалізація цієї змінної (команда int & ref = num) має деякі особливості. А саме:

- при оголошенні змінної ref перед ім'ям змінної вказано амперсанд &;
- змінної ref привласнюється як значення змінна num, причому самої змінної num значення ще не присвоєно.

Дві ці особливості характерні для оголошення посилання.

Посилання є змінною, яка посилається (дозволяє отримати доступ) на ту ж область пам'яті, що і деяка інша змінна. З деяким спрощенням посилання можна вважати синонімом певної змінної. Справа в тому, що при оголошенні звичайної змінної для неї в пам'яті виділяється місце. При створенні посилання пам'ять під посилання не виділяється, а використовується область пам'яті, виділена під іншу змінну. В результаті отримуємо дві змінні, які мають доступ до однієї і тієї ж області пам'яті. Змінна, область пам'яті якої використовується посиланням, присвоюється посиланням при оголошенні. Перед ім'ям посилання при оголошенні використовується символ &. Посилання створюється для однієї змінної і не може бути «перекинута» на іншу змінну.

У програмі виконуються такі операції зі змінною num і посиланням ref:

- присвоюється значення зміною num, після чого перевіряються значення змінної num і посилання ref;
- присвоюється значення посиланням ref і знову перевіряються значення змінної num і посилання ref.

3.2.4. Динамічне виділення пам'яті

Масиви, які ми розглядали до цього, були статичними - пам'ять під них виділялася в процесі компіляції програми. Існують динамічні масиви, пам'ять під які виділяється в процесі виконання програми.

Динамічне виділення пам'яті виконується за допомогою оператора new. Після оператора new вказується ідентифікатор типу, що визначає, для зберігання змінної якого типу виділяється пам'ять. Якщо пам'ять виділяється не під окрему змінну, а під масив, то після ідентифікатора типу в квадратних дужках вказується розмір динамічного масиву. Результатом інструкції по виділенню динамічної пам'яті повертається вказівник на виділену область пам'яті (для окремої змінної) або вказівник на перший елемент масиву (якщо створюється масив).

Після того, як в динамічно виділеної пам'яті необхідність відпала, її звільняють за допомогою оператора delete. Після оператора delete розміщується вказівник на звільнену область пам'яті. Якщо видаляється динамічний масив, то між оператором delete і вказівником на перший елемент видаляється масиву разме частішають порожні квадратні дужки [].

Далі представлена програма, в якій створюється динамічний масив, заповнюється значеннями, після чого масив видаляється.

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main () {
// Зміна кодування консолі:
System ( "chcp тисяча двісті п'ятьдесят один> nul");
// Вказівник на ціле число:
int * size;
// Динамічне виділення пам'яті під змінну:
```

```
size = new int;
cout << "Введіте розмір масиву:";
// Зчитування значення з клавіатури:
 cin >> * size;
// Вказівник на символьне значення:
char * symbs;
// Створення символьного масиву:
symbs = new char [* size];
// Заповнення масиву:
for (int k = 0; k <* size; k ++) {
symbs [k] = 'a' + k;
cout << symbs [k] << "";
}
// Видалення масиву:
delete [] symbs;
// Видалення змінної:
delete size;
cout << "\ nМассів і змінна видалені \ n";
// Затримка консольного вікна:
system (npause> nuln);
return 0;
}
```

У програмі створюється два вказівника: вказівник size на цілочисельне значення і вказівник symbs на символьне значення! Значення вказівником size присвоюється командою size = new int. В результаті виконання даної команди в пам'яті динамічно виділяється місце під цілочисльну змінну, а адреса виділеної області пам'яті записується значенням у вказівник size. Тут слід ще раз підкреслити, що size являє собою вказівник. Після динамічного виділення пам'яті цей вказівник отримує значення. Але область пам'яті, на яку він посилається, значення не отримує. У програмі воно зчитується з клавіатури, для чого використовується команда cin >> * size. Оскільки лічений значення заноситься не у вказівник size, а в область пам'яті, на яку він посилається, то праворуч від оператора введення >> використовуємо вираз * size.

Динамічний символьний масив створюється командою symbs = new char [* size]. У квадратних дужках зазначено вираз * size, яке визначає розмір масиву і є лічених з клавіатури значенням динамічно створеної змінної.

В результаті виконання команди symbs = new char [* size] створюється масив розміру *size, а адреса цього масиву (адреса його першого елемента) записується у вказівник symbs. У статичному масиві ім'я масиву є вказівником на перший елемент масиву. В даному випадку ідентифікатор symbs також є вказівником на перший елемент масиву. Тому ми цілком можемо ототожнювати

symbs з ім'ям динамічного масиву. Ми це, власне, і робимо, коли з допомогою оператора циклу заповнюємо масив значеннями і відображаємо значення елементів масиву в командному вікні. Значення елементу масиву присвоюється командою symbs [k] = 'a' + k. У правій частині вираження 'a' + k обчислюється так: до коду символу 'a' додається значення k і отримане число служить кодом, за яким визначається символ, який є результатом виразу. В результаті масив symbs заполоняють послідовністю букв, починаючи з 'a'.

Для видалення динамічного масиву symbs використовується команда delete [] symbs, а динамічна змінна видаляється командою delete size.

Головна особливість символьного масиву, що містить текст - це те, що розмір масиву в загальному випадку не збігається з розміром тексту, який в нього записується (масив повинен бути досить великий, щоб він міг утримувати текст різної довжини). Як індикатор завершення тексту в символьному масиві використовують нуль-символ \ 0, у якого нульовий код.

Оператор виводу <<, якщо праворуч від нього зазначено ім'я символьного масиву, виконує виведення всіх елементів символьного масиву, аж до нуль-символу.

Нижче буде представлена програма, в якій виконуються певні маніпуляції з текстовими значеннями. Основний для реалізації тексту служить символьний масив.

В C ++ існує два способи реалізації тексту: у вигляді символьного масиву і через об'єкт класу string. Однак базовим є спосіб реалізації тексту за допомогою символьного масиву - у всякому разі, текстові літерали реалізуються саме так.

Розглянемо наведений нижче програмний код:

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main () {
// Зміна кодування консолі:
system ( "chcp тисяча двісті п'ятьдесят один> nul");
// Символьний масив:
char str [100] = "Програмуємо на C ++";
// Відображення вмісту символьного масиву:
cout << str << endl;
// Поелементне відображення вмісту
// символьного масиву:
for (int k = 0; str [k]; k ++) {
cout << str [k] << "_";
}
cout << endl;
// Відображення вмісту починаючи
// з певної позиції: for (char * p = str; * p; p ++) {
```

```
cout << p << endl;
}
// Зміна символу в масиві:
str [13] = '\ 0';
// Відображення вмісту масиву:
cout << str << endl;
// Відображення, починаючи з вказаної позиції:
 cout << str + 14 << endl;
// Відображення тексту, до якого додається число:
cout << "РАЗ два три" +4 << end1;
// Вказівник на символьне значення:
const char * q = "РАЗ два три" +8;
// Значення символу, на який посилається вказівник:
 cout << q [0] << endl;
// Відображення значення вказівника:
cout << q << endl;
// Затримка консольного вікна:
system ( "pause> nul");
return 0;
```

Обговоримо основні і найбільш цікаві фрагменти програми. Отже, командою char str [100] = "Програмуємо на С ++" створюється символьний масив str і в цей масив автоматично записується (посимвольний) текст "Програмуємо на С ++". Безпосередньо масив складається з 100 елементів. Текст, який записується в масив, очевидно, містить набагато менше букв. Тому частина елементів масиву залишаються невикористаними. Щоб при обробці символьного масиву було зрозуміло, де закінчується текст і починаються «невикористовувані» елементи, в масив автоматично додається нуль-символ '\ 0'.

Для відображення тексту з масиву str в командному вікні використовуємо команду cout << str << endl. Формально в даному випадку Ви зможете бачити значення вказівника (оскільки ім'я масиву є вказівником на перший елемент масиву). Якби мова йшла, наприклад, про числовому масиві, то був би відображений числовий код адреси, що є значенням вказівника. Але символьні масиви (а якщо точніше, то вказівники на символьні значення) обробляються особливим чином: при спробі вивести в консольне вікно за допомогою оператора виведення значення вказівника на символ насправді відображається символ з осередки, на яку встановлено вказівник, а також символи з наступних осередків. Процес триває до тих пір, поки в черговий осередку не буде прочитаний нуль-символ. Підсумок такий: щоб відобразити вміст символьного масиву в консолі, слід праворуч від оператора виведення вказати ім'я масиву.

Разом з тим ніхто нам не забороняє обробляти і відображати вміст символьного масиву поелементно. Ситуацію ілюструє оператор циклу, в якому

текст з масиву str відображається символ за символом, при цьому після кожного символу додається символ підкреслення.

3.2.5. Двовимірні масиви

У двовимірному масиві доступ до елементу виконується за допомогою двох індексів. Нижче представлена програма, в якій створюється двовимірним масив. Масив заповнюється випадковими символами (буквами) і його вміст виводиться з консольне вікно.

При створенні двовимірного масиву вказується тип його елементів, назва масиву і розмір по кожному з індексів. Для кожного індексу використовуються окремі квадратні дужки.

Двовимірний масив зручно представляти у вигляді таблиці або матриці. Розмір по першому індексу визначає кількість рядків в матриці, а розмір по другому індексу визначає кількість стовпців. Звернення до елементу двовимірного масиву виконується зазначенням імені масиву і пари індексів. Індексація (по кожному з двох індексів) починається з нуля.

Розглянемо наведений нижче програмний код:

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main () {
// Зміна кодування консолі:
system ( "chcp тисяча двісті п'ятьдесят один> nul");
// Ініціалізація генератора випадкових чисел:
srand (2);
// Кількість "стовпців" в масиві:
const int width = 9;
// Кількість "рядків" в масиві: const int height = 5;
// Створення двовимірного масиву:
char Lts [height] [width];
// Заповнення двовимірного масиву:
for (int i = 0; i <height; i ++) {
        for (int j = 0; j <width; j ++) {
        // Випадкова буква від 'A' до 'Z':
        Lts [i] [j] = 'A' + rand ()% 25;
        // Відображення значення елемента масиву:
        cout << Lts [i] [j] << "";
}
// Перехід до нового рядка:
cout << endl;
}
```

```
// Затримка консольного вікна:
system ( "pause> nul");
 return 0;
}
```

Що стосується безпосередньо програмного коду, то там з'являються дві цілочисельні константи width і height, які визначають розміри масиву. Сам масив створюємо командою char Lts[height][width]. Для заповнення масиву та виведення значень його елементів в консоль використовуються вкладені оператори циклу. Індексна змінна і в зовнішньому операторі циклу перебирає рядки масиву, а індексна змінна j у внутрішньому операторі циклу перебирає стовпці масиву. Значення елементів масиву привласнюються командою Lts [i] [j] = 'A' + rand ()% 25. Значення елемента масиву виходить додатком до коду символу 'A' цілого випадкового числа в діапазоні значень від 0 до 24, і по отриманому таким чином коду відновлюється символ. Відображається значення елемента двовимірного масиву командою cout << Lts [i] [j] << "". Роздільником між символами в одному рядку є пробіл. Після завершення відображення рядка командою cout << endl виконується перехід до нового рядка.

3.2.6. Масиви вказівників

Елементами масиву, крім іншого, можуть бути вказівниками. Іншими словами, ніхто не забороняє нам створити масив з вказівників. Більш конкретно, уявімо, що створюється масив з вказівників на цілочисельні значення. Елементи такого масиву є вказівниками. Ім'я цього масиву є вказівником на вказівник на ціле число.

Щоб оголосить вказівник на вказівник, в команді оголошення використовують дві зірочки **. Наприклад, інструкція int **p оголошує вказівник p, значенням якого може бути адреса змінної, яка сама є вказівником на цілочисельне значення.

Далі чинимо так: створюємо динамічні числові масиви і адреси цих масивів (їх перших елементів) присвоюємо значеннями елементів масиву вказівників. Виходить специфічна конструкція, яка за всіма зовнішніми ознаками нагадує двовимірний масив, тільки в такому масиві в різних рядках може міститися різна кількість елементів. Розглянемо такий програмний код:

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main () {
// Зміна кодування консолі:
system ( "chcp I251> nul");
// Ініціалізація генератора випадкових чисел:
srand (2);
// Індексні змінні:
```

```
int i, j;
// Розмір масиву вказівників:
const int size = 5;
// Масив зі значеннями, визначальними розміри
// числових масивів:
const int cols [size] = (3,7,6,4,2);
// Динамічний масив вказівників:
int ** nums = new int * [size];
// Створення динамічних числових масивів
// і заповнення їх значеннями:
for (i = 0; i <size; i ++) {
// Динамічний числовий масив:
nums [i] = new int [cols [i]];
cout << "|";
// Заповнення числового масиву: for (j = 0; j <cols [i]; j ++) {
// Випадкове число від 0 до 9:
nums [i] [j] = rand ()% lO;
// Відображення значення елемента масиву:
cout << nums [i] [j] << "";
}
cout << endl;
}
// Видалення числових динамічних масивів;
for (i = 0; i <size; i ++) {;
// Видалення динамічного числового масиву:
delete [] nums [i];
}
// Видалення динамічного масиву вказівників:
delete [] nums;
// Затримка консольного вікна:
system ( "pause> nul");
return 0;
}
```

У програмі, крім іншого, створюється цілочисельна константа size, яка визначає масив покажчиків. Масив створюємо командою int ** nums = new int * [size]. Це динамічний масив. Тип його елементів визначається інструкцією int * в правій частині виразу. Зірочка * після int-інструкції свідчить про те, що мова йде про покажчики на цілі числа. Підкреслимо, що покажчики на цілі числа - елементи створеного масиву. Ім'я масиву є покажчиком на перший елемент, який в даному випадку сам є покажчиком. Тому ім'я масиву nums є покажчиком на

покажчик на ціле число. Тому змінна nums оголошена з ідентифікатором типу int **.

Ще в програмі командою const int cols [size] = {3,7,6,4,2} оголошується і ініціалізується константний (масив з елементами-константами) масив cols. Елементи даного масиву визначають розмір динамічних числових масивів, які створюються далі в програмі і покажчики на які записуються в масив nums.

Для створення динамічних числових масивів і заповнення їх значеннями запускається оператор циклу, в якому індексна і змінна пробігає значення від 0 до size-1. У тілі оператора циклу командою nums [i] = new int [cols [i]] створюється динамічний числовий масив. Розмір масиву визначається елементом cols[i] масиву cols. Покажчик на масив присвоюється значенням елементу nums [i] масиву покажчиків. Після того, як числовий динамічний масив створений, він заповнюється випадковими числами, а значення його елементів виводяться в консольне вікно. Реалізується все перераховане за допомогою оператора циклу, в якому індексна змінна j послідовно приймає значення в діапазоні від 0 до cols [i] -1. Значення елементу присвоюється командою nums [i] [j] = rand ()% 10, а відображається значення елемента в консолі командою cout << nums [i] [j] << "|".

Після того, як динамічні масиви створені і заповнені, потреба в них відпадає, і вони видаляються з пам'яті. Спочатку видаляються числові масиви, для чого запускається оператор циклу і в тілі циклу командою delete [] nums [i] видаляється кожен з числових динамічних масивів. Після цього командою delete [] nums видаляється динамічний масив покажчиків.

3.3. Задачі для розв'язання

1. Напишіть програму, в якій створюється два числових масива однакового розміру. Необхідно обчислити суму попарних добутків елементів цих масивів. Так, якщо через a_k і b_k позначити елементи масивів (індекс 0 <= k <n), то необхідно обчислити суму $\sum_{k=0}^{n-1} a_k b_k$

2. Напишіть програму, в якій створюється числовий масив і для цього масиву обчислюється сума квадратів елементів масиву.

3. Напишіть програму, в якій створюється двовимірний числовий масив і для цього масиву обчислюється сума квадратів його елементів.

4. Напишіть програму, в якій створюється квадратна матриця (реалізується через двовимірний масив). Матрицю необхідно заповнити числовими значеннями (спосіб заповнення вибрати самостійно), а потім виконати транспонування матриці: матриця симетрично «відбивається» щодо головної діагоналі, в результаті чого елемент матриці a_{ij} стає елементом a_{ji} і навпаки.

5. Напишіть програму, в якій створюється квадратна матриця (реалізується через двовимірний масив). Матриця заповнюється випадковими числами, а потім робиться «поворот за годинниковою стрілкою»: перший стовпець стає першим рядком, другий стовпець стає другим рядком, і так далі.

6. Напишіть програму, в якій для двовимірної квадратної числовий матриці обчислюється слід - сума діагональних елементів матриці.

7. Напишіть програму, в якій для одновимірного числового масиву обчислюється найбільший (найменший) елемент і позиція, на якій елемент знаходиться в масиві.

8. Напишіть програму, в якій створюється одновимірний числовий масив. Після заповнення значеннями (наприклад, випадковими числами) масиву в ньому потрібно виконати циклічну перестановку елементів. Кількість позицій для циклічної перестановки вводиться користувачем з клавіатури.

9. Напишіть програму, в якій створюється і заповнюється натуральними числами двовимірний масив. Заповнення починається з лівого верхнього елемента зліва направо, зверху вниз (тобто заповнюється спершу перший рядок, потім друга, і так далі).

10. Напишіть програму, в якій створюється і заповнюється натуральними числами двовимірний масив. Заповнення починається з лівого верхнього елемента зверху вниз, зліва направо (тобто заповнюється спочатку перший стовпчик, потім другий і так далі).

11. Напишіть програму, в якій створюється квадратна матриця. Елементам на обох діагоналях присвоюються поодинокі значення, всі інші елементи нульові

12. Напишіть програму для обчислення добутку прямокутних матриць (кількість рядків і стовпців в матрицях різне). Перемножуються матриці A (розмірами m на n) і B (розмірами n на l). Результатом є матриця C (розмірами m на l). Робоча формула для обчислення значень матриці C має вигляд $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ $1 \le i \le m, 1 \le j \le l$.

13. Напишіть програму, в якій створюється символьний масив для запису тексту. У масив записується текст, після чого необхідно змінити порядок проходження символів в тексті на протилежний: останній символ стає першим, передостанній символ стає другим і так далі.

14. Напишіть програму, в якій створюється двовимірний числовий масив і заповнюється випадковими числами. Для кожного рядка (або стовпця) двовимірного масиву визначається найбільший (або найменший) елемент, і з таких елементів створюється одновимірний числовий масив.

15. Напишіть програму, в якій створюється символьний масив для запису тексту. Підрахувати для записаного в масиві тексту кількість слів і довжину кожного слова. Словами вважати набір символів між пробілами.

16. Напишіть програму, в якій створюється динамічний масив. Розмір масиву - випадкове число (тобто генерується випадкове число, що визначає розмір масиву). Заповнити масив «симетричними» значеннями: перший і останній елемент отримують значення 1, другий і передостанній елемент отримують значення 2, і так далі.

17. Напишіть програму, в якій створюється три одновимірних числових масиву однакового розміру. Перші два масиви заповнюються випадковими

числами. Третій масив заповнюється так: порівнюються елементи з такими ж індексами в першому і другому масиві і в третій масив на таку ж позицію записується більше (або менше) з порівнюваних значень.

18. Напишіть програму, в якій створюється два динамічних масиву (різної довжини). Масиви заповнюються випадковими числами. Потім створюється третій динамічний масив, і його розмір дорівнює сумі розмірів перших двох масивів. Третій масив заповнюється так: спочатку в нього записуються значення елементів першого масиву, а потім значення елементів другого масиву.

19. Напишіть програму, в якій створюється два динамічних масиву однакового розміру. Масиви заповнюються випадковими числами. Потім створюється третій динамічний масив, і його розмір в два рази більше розміру кожного з перших двох масивів. Третій масив заповнюється почергової записом елементів з перших двох масивів: спочатку записується значення елемента першого масиву, потім значення елемента другого масиву, потім знову першого і знову другого, і так далі.

20. Напишіть програму, в якій створюється двовимірний числовий масив. Масив заповнюється випадковими числами. На його основі створюється новий масив, який виходить «викреслюванням» зі старого одного рядка і одного стовпця. Номери «викреслювати» шпальти і рядки визначаються введенням з клавіатури або за допомогою генератора випадкових чисел.

Практичне заняття 4. Технології роботи з функціями.

4.1. Мета практичного заняття

Вивчити технології створення функції. Типи функцій та їх реалізація. Особливості застосування функцій для створення оптимізованого коду програми.

4.2. Теоретичний матеріал

4.2.1. Оголошення та опис функції

Дуже простий приклад використання функцій представлений в програмі нижче. Там описані математичні функції для обчислення синуса і косинуса. У програмі дані функції використовуються для обчислення значень при заданому аргументі, а аналогічні вбудовані функції викликаються для перевірки результату обчислень.

Розглянемо наведений нижче програмний код:

```
#include <iostream>
#include <cstdlib> #include <cmath>
using namespace std;
// Оголошення функцій
double myCos (double); // Обчислення косинуса
double mySin (double); // Обчислення синуса
void show (double); // Допоміжна функція
// Головна функція програми:
int main () {
// Зміна кодування консолі:
system ( "chcp тисяча двісті п'ятьдесят один> nul");
// Константа для значення числа "пі":
const double pi = 3.141592;
// Обчислення синуса і косинуса при різних
// значеннях аргументу:
show (pi / 6);
show (pi / 4);
show (pi / 3);
show (pi / 2);
// Затримка консольного вікна: system ( "pause> nul");
return 0;

// Опис функції для обчислення косинуса:
double myCos (double x) {
// Верхня межа суми:
```

```
int n = 100;
// Змінні для запису значення суми
// і добавки до суми:
double s = 0, q = l;
// Обчислення суми:
for (int k = 0; k <= n; k ++) {
s + = q; // Добавка до суми
// Добавка для наступної ітерації:
q * = (- 1) * x * x / (2 * до + 1) / (2 * до + 2);
}
// Результат функції: return s;
}
// Опис функції для обчислення синуса:
double mySin (double x) {
// Верхня межа суми:
int n = 100;
// Змінні для запису значення суми
// і добавки до суми:
double s = 0, q = x;
// Обчислення суми:
for (int k = 0; k <= n; k ++) {
s + = q; // Добавка до суми
// Добавка для наступної ітерації:
q * = (- 1) * x * x / (2 * до + 2) / (2 * до + 3);
}
// Результат функції: return s;
}

// Опис допоміжної функції: void show (double x) {
cout << "Значення аргументу:" << x << endl;
// Обчислення косинуса:
cout << "Косинус:" «myCos (x) <<" vs. "<< cos (x) << endl;
// Обчислення синуса:
cout << "Синус:" «mySin (x) <<" vs. "<< sin (x) << endl;
// Додатковий відступ:
cout << endl;
}
```

У програмі описуються функції myCos () для обчислення косинуса, mySin() для обчислення синуса і допоміжна функція show (). Причому функції спочатку оголошуються, а безпосередньо опис функцій виконується після опису головної функції main ().

Функція myCos () результатом повертає значення типу double і у неї один аргумент (позначений як x), теж типу double. Локальна целочисельна змінна n оголошена і ініціалізувати зі значенням 100 в тілі функції, визначає верхню межу суми, що підлягає поверненню результатом функції. Ще дві змінні типу double призначені для запису значення суми (змінна s з початковим значенням 0) і доданка, яке на кожній ітерації додається до суми (змінна q з початковим значенням 1). Обчислюється сума за допомогою оператора циклу, в якому індексна змінна до пробігає значення від 0 до n включно, і за кожен цикл виконуються команди s+=q (додавання нового доданка до суми) і q*=(-1)*x*x/(2*k+1)/(2*k+2) (обчислення добавки для наступної ітерації). По завершенні обчислень значення змінної s повертається результатом функції.

Функція для обчислення mySin() описана аналогічно до функції для обчислення косинуса myCos(), хіба що з поправкою на змінене початкове значення для добавки (змінна q) і спосіб її обробки (команда q*=(-1)*x*x/(2*k+2)/(2*k+3) в тілі оператора циклу).

Функція show() не повертає результат, тому ідентифікатором типу результату для неї вказано ключове слово void. У функції є аргумент типу double, позначений як x. Це значення використовується при обчисленні синуса і косинуса.

Функцією show() відображається значення аргументу, для якого обчислюється синус і косинус, значення синуса і косинуса, обчислені за допомогою функцій mySin() і myCos(), а також для порівняння наводяться значення для синуса і косинуса, обчислені за допомогою вбудованих функцій sin() і cos ().

4.2.2. Перевантаження функцій

Вирішимо програмними методами задачу про обчислення прибутку, одержуваної вкладником банку. Вхідними параметрами задачі є:

- вихідна грошова сума, яку вкладник розміщує в банку на депозитному рахунку;
- річна процентна ставка, за якою нараховується дохід на вкладену суму;
- кількість років, на які розміщується депозит;
- кількість періодів в році, коли виконується нарахування відсотків.

Ми хочемо передбачити в програмі наступні можливості:

- якщо при обчисленні доходу вказується тільки початкова сума і процентна ставка, то автоматично вважається, що депозит розміщується на один рік і відсотки нараховуються теж один раз на рік;
- якщо при обчисленні доходу вказується початкова сума, про-процентним ставка і кількість років, то за замовчуванням вважаємо, що відсотки нараховуються раз на рік;
- також можна при обчисленні доходу вказати початкову суму, річну процентну ставку, кількість років і кількість періодів нарахування відсотків.

Для всіх перерахованих випадків ми опишемо окрему функцію, ви-яка значиться підсумкову суму, причому скористаємося таким механізмом, як перевантаження функцій.

Перевантаження функцій полягає в тому, що в програмі створюється кілька функцій з однаковими назвами. Такі функції повинні відрізнятися прототипом (тип результату, назва функції, кількість і тип аргументів) так, щоб при виконанні функції можна було однозначно визначити, про яку саме версію функції йдеться. Про функції з однаковою назвою зазвичай говорять як про різних версіях однієї функції.

Програмний код з рішенням даної задачі представлений нижче.

```
#include <iostream>
#include <cstdlib>
using namespace std;
// Функція з двома аргументами:
double getMoney (double m, double r) {
return m * (l + r / 100);
}
// Функція з трьома аргументами:
double getMoney (double m, double r, int y) {
double s = m;
for (int k = l; k <= y; k ++) {
s * = (l + r / 100);
}
return s;
}
// Функція з чотирма аргументами:
double getMoney (double m, double r, int y, int n) {
return getMoney (m, r / n, y * n);
}
// Головна функція програми:
int main () {
// Зміна кодування консолі:
system ( "chcp тисяча двісті п'ятьдесят один> nul");
// Початкова сума вкладу:
double money = 1000;
// Процентна ставка:
double rate = 5;
cout << "Початкова сума:" << money << endl;
cout << Річна ставка: "<< rate <<"% \ n ";
// Обчислення доходу за різні проміжки часу:
cout << "Внесок на один рік:" << getMoney (money, rate) << endl;
cout << "Внесок на 7 років:" << getMoney (money, rate, 7) << endl;
```

```
cout << "Внесок на 7 років \ n (нарахування 3 рази в рік):";
cout << getMoney (money, rate, 7,3) << endl;
// Затримка консольного вікна:
system ( "paus> nul");
 return 0;
}
```

Проаналізуємо програмний код. Нас в основному буде цікавити опис функції getMoney (). У програмі описані три версії цієї функції, які відрізняються лише кількістю аргументів, які передаються функції. Розглянемо кожен з них окремо.

Найбільш простий код у функції getMoney() з двома аргументами. В цьому випадку double-аргумент m визначає початкову суму вкладу, а ще одні double-аргумент r дає значення річної відсоткової ставки. Результатом функції повертається значення виразу m * (l + r / 100) в повній відповідності з формулою для обчислення підсумкової суми для річного депозиту.

У версії функції з трьома аргументами перші два аргументи m та r мають таке ж призначення, як і в попередньому випадку, а третій цілочисельний аргумент y визначає кількість років, на які розміщується депозит. Результат в тілі функції обчислюється за допомогою оператора циклу. Локальна змінна s з початковим значенням m (перший аргумент функції), яка повертається результатом функції, за кожен цикл множиться на множник (1 + r / 100), і таких циклів рівно y. В результаті отримуємо значення, відповідне формулою обчислення підсумкової суми депозиту, розміщеного на кілька років (з одноразовим нарахуванням відсотків в кінці року).

У версії функції getMoney() з чотирма аргументами останній цілочисельний аргумент n визначає кількість періодів в році для нарахування відсотків. Результатом функції повертається значення вираз getMoney(m,r/n/,y*n). Тут ми фактично в тілі функції з чотирма аргументами викликаємо версію цієї ж самої функції з трьома аргументами.

У головній функції програми представлені приклади виклику функції getMoney() з різною кількістю аргументів. Кожен раз, коли викликається функція, що має кілька версій, рішення про задіювання тієї чи іншої версії функції приймається на основі команди виклику: в основному за кількістю і типом аргументів, а також за типом результату.

4.2.3. Значення аргументів за замовчуванням

Розглянуте вище завдання можна було вирішити і трохи інакше. Зокрема, в даному конкретному випадку замість перевантаження функцій ми цілком могли обмежитися створенням всього однієї функції, але з аргументами, що мають значення за замовчуванням.

При описі функції для деяких або всіх її аргументів можна вказати значення за замовчуванням. Дані значення використовуються, якщо при виконанні функції відповідний аргумент не вказано. Аргументи, які мають

значення за замовчуванням, описуються в списку аргументів функції останніми. Значення за замовчуванням для аргументу вказується в прототипі функції після імені аргументу через знак рівності.

У лістингу нижче представлена програма, в якій вирішується поставлене раніше завдання, але тільки тепер ми створюємо всього одну функцію getMoney(), а два її останніх аргументу мають значення за замовчуванням.

```
#include <iostream>
#include <cstdlib>
using namespace std;
// Функція з аргументами, що мають
// значення за замовчуванням:
double getMoney (double m, double r, int y = l, int n = l) {
double s = m;
double z = n * y;
double q = r / n;
for (int k = l; k <= z; k ++) {
s * = (l + q / 100);
}
return s;
}
// Головна функція програми:
 int main () {
// Зміна кодування консолі:
System ( "chcp тисяча двісті п'ятьдесят один> nul");
// Початкова сума вкладу:
double money = 1000;
// Процентна ставка:
 double rate = 5;
cout << "Початкова сума:" << money << endl;
cout << Річна ставка: "<< rate <<"% \ n ";
// Обчислення доходу за різні проміжки часу:
cout << "Внесок на один рік:" << getMoney (money, rate) << endl;
cout << "Внесок на 7 років:" << getMoney (money, rate, 7) << endl;
cout << "Внесок на 7 років \ n (нарахування 3 рази в рік):";
cout << getMoney (money, rate, 7,3) << endl;
// Затримка консольного вікна:
system ( "pause> nul");
return 0;
}
```

Коли функція getMoney () викликається з трьома аргументами, це все одно, як якщо б четвертий її аргумент був дорівнює одиниці (значення за

замовчуванням). Якщо функція викликається тільки з двома аргументами, то третій і четвертий її аргументи автоматично вважаються одиничними (значення за замовчуванням для даних аргументів). Що стосується коду, що визначає функцію, то локальна змінна s з початковим значенням m визначає стартовий внесок, змінна z зі значенням n*y визначає загальна кількість періодів нарахування відсотка, змінна q зі значенням r / n визначає процентну ставку відсотка для кожного періоду, за який нараховуються відсотки. Результат обчислюється за допомогою оператора циклу, в якому z циклів і за кожен цикл змінна s множиться на (l + q / 100). По завершенні обчислень значення змінної s повертається результатом функції. Тут, за великим рахунком, ми повторили алгоритм обчислень, використаний в попередньому прикладі для випадку функції з трьома аргументами. Єдине, ми зробили «поправку» на ефективну процентну ставку і кількість періодів нарахування процентного доходу.

4.2.4. Рекурсія

Якщо в описі функції викликається ця сама функція (але, як правило, з іншим аргументом або аргументами), то така ситуація називається рекурсією. Приклад використання рекурсії представлений в лістингу нижче. Мова йде про все ту ж задачі про обчислення підсумкової суми депозиту, але тільки тепер для простоти ми розглядаємо не три, а один спосіб нарахування відсотків: депозит розміщується на вказану кількість років під фіксовану річну ставку відсотка, а нарахування доходу за відсотками виробляються раз на рік . У програмі описана функція getMoney() з трьома аргументами, і при описі функції використана рекурсія. Розглянемо наведений нижче програмний код:

```
#include <iostream>
#include <cstdlib>
using namespace std;
// У функції використовується рекурсивний виклик:
double getMoney (double m, double r, int y) {
if (y == 0) {
return m;
}
else {
return (l + r / 100) * getMoney (m, r, yl);
}
}
// Головна функція програми:
int main () {
// Зміна кодування консолі:
system ( "chcp тисяча двісті п'ятьдесят один> nul");
// Початкова сума вкладу:
double money = 1000;
// Процентна ставка:
```

```
double rate = 5;
cout << "Початкова сума:" << money << endl;
cout << "Річна ставка:" << rate << "% \ n";
// Обчислення доходу за різні проміжки часу:
cout << Внесок на 1 рік: "<< getMoney (money, rate, l) << endl;
cout << "Внесок на 7 років:" << getMoney (money, rate, 7) << endl;
cout << "Внесок на 10 років:" << getMoney (money, rate, 10) << endl;
// Затримка консольного вікна:
system ( "pause> nul");
return 0;
}
```

У програмі описується функція getMoney () з трьома аргументами, після чого в головній функції програми функція getMoney () викликається з різними значеннями третього аргументу.

Інтерес представляє спосіб опису функції getMoney (). Основу коду функції становить умовний оператор, в якому перевіряється умова y == 0 (через y позначений третій аргумент функції getMoney ()). Якщо умова істинно, результатом функції повертається значення m (перший аргумент функції getMoney ()). Тут ми виходимо з обставини, що на вклад, розміщений в банку на 0 років, відсотки не нараховуються. Тому підсумкова сума в такій ситуації дорівнює початковій сумі вкладу. Якщо ж умова y == 0 помилково, то результатом функції повертається значення виразу (1 + r/100)*getMoney(m,r,y-1). Це і є рекурсивний виклик функції getMoney ().

Значення функції getMoney () з заданими аргументами m, r і y (значення виразу getMoney (m, r, y)) обчислюється таким чином. Перевіряється значення аргументу y, і якщо воно відмінно від нуля, обчислюється значення виразу (1+r/100)*getMoney(m,r,y-1). При обчисленні значення виразу знову викликається функція getMoney(), але на цей раз третій аргумент на одиницю менше, і так далі, поки не буде викликана функція getMoney () з нульовим третім аргументом. Тоді при виконанні функції повертається значення ш (перший аргумент функції). Це дозволяє обчислити значення попереднього виразу, потім того, що було перед ним, і так далі, аж до самого першого виразу, з якого все починалося.

4.2.5. Механізми передачі аргументів функцій

Щоб зрозуміти природу проблеми, яка буде обговорюватися, розглянемо невеликий приклад, наведений нижче. Суть програми дуже проста:

- Використання функцій swap() з двома символьними (тип char) аргументами;
- при виконанні функції значення переданих аргументів виводяться в консольне вікно;

• аргументи обмінюються значеннями, і їх нові значення також відображаються в командному вікні.

У програмі оголошується дві змінні, вони передаються аргументами функції swap () при її виклику, а потім перевіряються значення змінних.

Програмний код має вигляд:

```
#include <iostream>
#include <cstdlib>
using namespace std;
// Передача аргументів за значенням:
void swap (char a, char b) {
cout << "Викликається функція swap ()" << endl;
// Перевіряються значення аргументів функції:
cout << "Перший аргумент:" << a << endl;
cout << "Другий аргумент:" << b << endl;
// Зміна значень аргументів функції:
char t = b;
b = a;
a = t;
for (int i = l; i <= 20; i ++) {
cout << "-";
cout << endl;
// Перевіряються значення аргументів функції:
cout «" Перший аргумент: "<< a << endl;
cout «" Второй аргумент: "<< b << endl;
cout << "Виконання функції swap () завершено" << end1;
}
// Головна функція програми: int main () {
// Зміна кодування консолі:
system ( "chcp тисяча двісті п'ятьдесят один> nul");
// Змінні для передачі аргументами функції:
char x = 'A', y = 'B';
// Перевіряються значення змінних:
cout << "Перша змінна:" << x << endl;
cout << "Друга змінна:" << y << endl;
// Виклик функції: swap (x, y);
// Перевіряються значення змінних:
cout << "Перша змінна:" << x << endl;
cout << "Друга змінна:" << y << endl;
// Затримка консольного вікна:
system ( "pause> nul");
return 0;
```

```
}
```

Наші очікування від результатів виконання програми могли бути наступними: після виклику функції swap() змінні, які пере даються їй аргументами, обміняються значеннями. В реальності резуль тат виконання програми такий, як наведено нижче:

Як бачимо, при перевірці значень аргументів на тілі функції все йде за планом: аргументи обмінялися значеннями. Але от коли ми перевіряємо значення змінних після виклику функції swap () виявляється, що їх значення не змінилися. Причина криється в способі передачі аргументів функції. За замовчуванням використовується режим передачі аргументів за значенням: передаються не самі змінні, а їх копії (після завершення виконання функції копії аргументів видаляються з пам'яті). Тому при виконанні функції swap () всі операції виконуються не зі змінними x і y, які передавалися аргументами функції, а з їх копіями. Саме копії обмінюються своїми значеннями. Значення змінних x і y залишаються незмінними.

4.2.6. Передача вказівника аргументом функції

Ще один спосіб опису функції swap (), але на цей раз аргументами функції передаються вказівники. Розглянемо програму, представлену в лістингу нижче.

```
#include <iostream>
#include <cstdlib>
using namespace std;
// Аргументами функції передаються вказівники:
void swap (char * a, char * b) {
cout << "Викликається функція swap ()" << endl;
// Перевіряються значення аргументів функції:
cout << "Перша змінна:" << * a << endl;
cout << "Друга змінна:" << * b << endl;
// Зміна значень змінних:
char t = * b;
* B = * a;
* A = t;
for (int i = l; i <= 20; i ++) {
cout << "-";
}
cout << endl;
// Перевіряються значення змінних:
cout << "Перша змінна:" << * a << endl;
cout << "Друга змінна:" << * b << endl;
cout << "Виконання функції swap () завершено" «endl;
}
```

```
// Головна функція програми:
int main () {
// Зміна кодування консолі:
system ( "chcp тисяча двісті п'ятьдесят один> nul");
// Змінні для передачі аргументами функції:
char x = 'A', y = 'B';
// Перевіряються значення змінних:
cout << "Перша змінна :," << x << endl;
cout << "Друга змінна:" << y << endl;
// Виклик функції: swap (& x, & y);
// Перевіряються значення змінних:
cout << "Перша змінна:" << x «endl;
cout «" Друга змінна: "<< y << endl;
// Затримка консольного вікна:
system ( "pause> nul");
return 0;
}
```

У програмі в описі функції swap () тип обох аргументів вказаний як char *. Це означає, що аргументами функції передаються вказівники на символьні значення. Тому коли функція swap () викликається в головній функції програми, команди виклику має вигляд swap (&x, &y). Оскільки змінні x і y оголошені як відносяться до типу char, то &x і &y (адреси змінних) - це вказівники на значення типу char. Іншими словами, якщо в попередніх прикладах в функцію передавалися змінні (або їх копії), то на цей раз аргументами функції передаються вказівники на змінні.

Отже, аргументами функції swap() передаються вказівники. Але перевіряються значення, які записані за відповідними адресами. І обмін значеннями відбувається не для вказівників, а для значень, які записані за адресами, що є значеннями вказівників. Так, командою char t = * b оголошується локальна змінна t і їй присвоюється значення, записане за адресою з вказівника b (другий аргумент функції). Далі командою * b = * a за адресою з вказівника b записується значення, розміщене за адресою з вказівника a (перший аргумент функції). Після цього командою * a = t за адресою, що міститься у вказівнику a, записується значення змінної t (саме це значення на самому початку було записано за адресою з вказівника b).

Важливо розуміти, що, коли ми передаємо аргументами функції вказівники, вони передаються за значенням. Тобто насправді створюється копія аргументу, який передається функції. Але специфіка ситуації в тому, що копія вказівника містить ті ж самі значення, що і вихідний вказівник, а тому копія вказівника встановлена на ту ж комірку пам'яті, що і оригінальний вказівник. У тілі функції, в свою чергу, операції (по зміні значення) виконуються не з вказівниками, а з осередками, адреси яких записані у вказівниках. Так що

механізм передачі аргументів за замовчуванням в даному прикладі принципової ролі не грає.

4.2.7. Вказівник як результат функції

Функція результатом може повертати практично все, що завгодно - за винятком статичного масиву. Зокрема, результатом функції може бути вказівник. У лістингу нижче представлена програма, в якій описується функція getMax (). У функцію передається числовий одновимірний масив, а результатом функція повертає вказівник на елемент з найбільшим значенням в масиві. Розглянемо наведений нижче програмний код:

```
#include <iostream>
#include <cstdlib>
using namespace std;
// Функція результатом повертає
// вказівник на елемент масиву:
 int * cretMax (int * nums, int n) {
int i = 0, k;
// Визначення індексу найбільшого елемента:
for (k = 0; k <n; k ++) {
if (nums [k]> nums [i]) {i = k;
}
}
// Результат функції - вказівник на елемент:
 return nums + i;
}
// Функція для відображення вмісту масиву:
 void show (int * nums, int n) {
for (int i = 0; i <n; i ++) {
cout << nums [i] << "";
}
cout << endl;
}
// Головна функція програми:
int main () {
// Зміна кодування консолі:
System ( "chcp тисяча двісті п'ятьдесят один> nul");
// Розмір масиву:
const int size = 10;
// Створення масиву:
int numbers [size] = {1,5,8,2,4,9,11,9,12,3};
// Відображення вмісту масиву:
show (numbers, size);
```

```
// Запис результата функції у вказівник:
int * maxPnt = getMax (numbers, size);
// Відображення максимального значення:
cout << "Максимальне значення:" << * maxPnt << endl;
// Присвоєння значення елементу:
* MaxPnt = -100;
 // Відображення вмісту масиву:
show (numbers, size);
// Значення присвоюється змінної:
int maxNum = * getMax (numbers, size);
// Перевірка максимального значення:
cout << "Максимальне значення:" << maxNum << endl;
// Присвоєння значення змінної:
maxNum = -200;
// Перевірка вмісту масиву:
show (numbers, size);
cout «" Максимальне значення: ";
// Обчислення нового максимального значення:
cout << * getMax (numbers, size) << endl;
cout << "Індекс елемента:";
// Обчислення індексу елемента з найбільшим значенням:
cout << getMax (numbers, size) -numbers << endl;
// Затримка консольного вікна: system ( "pause> nul");
return 0;
}
```

Ідентифікатором типу результату для функції getMax () вказана інструкція int *, що означає, що результатом функції є вказівник на цілочисельне значення. У тілі функції цілочисельна змінна і ініціалізована з початковим нульовим значенням. Далі запускається оператор циклу (індексна змінна до пробігає всі можливі значення для індексів масиву), в якому послідовно порівнюються значення елементів масиву зі значенням елемента, що має індекс i. Якщо виконана умова nums [k]> nums [i], то значення індексу до присвоюється змінної i. В результаті після перебору всіх елементів масиву в змінну і буде записаний індекс елемента з найбільшим значенням (або індекс першого стрічного елемента з найбільшим значенням, якщо таких елементів декілька). Результатом функції повертається значення nums + i. Тут ми скористалися правилами адресної арифметики і тим, що ім'я масиву є вказівником на його перший елемент.

У функції main () створюється цілочисельна константа size, яка визначає розмір масиву, а сам масив створюється командою int numbers [size] = {1,5,8,2,4,9, 11,9,12,3}. Відображення вмісту масиву виконується за допомогою спеціально описаної для такої мети функції show ().

При виконанні команди int * maxPnt = getMax (numbers, size) у вказівник maxPnt записується адреса того елемента в масиві number, який має найбільше значення. Дізнатися дане значення можемо через вираз *maxPnt. При цьому maxPnt є адресою (але не індексом!) Даного елемента Щоб дізнатися індекс елементу, необхідно відняти від вказівника на цей елемент вказівник на перший елемент масиву (ім'я масиву).

Після виконання команди *maxPnt = -100 елемент масиву, що мав до цього найбільше значення, отримає нове значення -100.

Якщо нас цікавить значення елемента, а не його адреса, то можна при виконанні функції getMax () вказати перед її ім'ям зірочку * (що означає отримання значення за адресою), як робиться в інструкції int maxNum = * getMax (numbers, size). В результаті в змінну maxNum записується значення найбільшого елементу, але при цьому інформація про адресу елемента не запам'ятовується. Тому якщо ми згодом дамо змінної maxNum нове значення, на вихідному масиві це ніяк не позначиться. Нарешті, для обчислення значення індексу елемента з найбільшим значенням обчислюємо вираз getMax (numbers, size) -numbers (різниця вказівників дає ціле число, яке визначає кількість осередків між відповідними областями пам'яті - для елементів масиву отримане таким чином значення збігається з індексом елемента).

4.2.8. Посилання як результат функції

Функція результатом може повертати посилання. Якщо функція повертає посилання, то її результатом є не просто значення деякого виразу або змінної, а фактично сама змінна.

Невелика варіація на тему попереднього прикладу, але на цей раз з поверненням функцією посилання замість вказівника, приведена в лістингу нижче.

```
#include <iostream>
#include <cstdlib>
using namespace std;
// Функція результатом повертає
// посилання на елемент масиву:
int & getMax (int * nums, int n) {
int i = 0, k;
// Визначення індексу найбільшого елемента:
for (k = 0; k <n; k ++) {
if (nums [k]> nums [i]) {i = k;
}
}
// Результат функції - посилання на елемент: return nums [i];
}
// Функція для відображення вмісту масиву:
```

```
void show (int * nums, int n) {
for (int i = 0; i <n; i ++) {
cout «nums [i]« "";
}
cout << endl;
}
// Головна функція програми:
int main () {
// Зміна кодування консолі:
system ( "chcp тисяча двісті п'ятьдесят один> nul");
// Розмір масиву:
const int size = 10;
// Створення масиву:
int numbers [size] = {1,5,8,2,4,9,11,9,12,3};
// Відображення вмісту масиву:
show (numbers, size);
// Запис результат функції в змінну:
int maxNum = getMax (numbers, size);
// Відображення максимального значення:
cout << "Максимальне значення:" << maxNum << endl;
// Присвоєння значення змінної:
maxNum = -100;
// Відображення вмісту масиву: show (numbers, size);
// Результат функції записується на заслання:
int SmaxRef = getMax (numbers, size):
// Перевірка максимального значення:
cout << "Максимальне значення:" «maxRef << endl;
// Присвоєння значення посиланням: maxRef = -200;
// Перевірка вмісту масиву:
show (numbers, size);
cout << "Максимальне значення:";
// Обчислення нового максимального значення:
Cout << getMax (numbers, size) << endl
// Затримка консольного вікна:
system ( "pause> nul");
return 0;
}
```

Подивимося, що принципово змінилося в програмному коді (в порівнянні з попереднім лістингом). В першу чергу, звичайно, це результат функції getMax(): тепер функція повертатися не вказівник, а посилання, про що свідчить інструкція & перед ім'ям функції в її описі. Оскільки функція описана з

ідентифікатором int, то мова йде про посиланням на цілочисельне значення. У тілі функції, для числового масиву nums, переданого в функцію, повертається посилання на елемент з найбільшим значенням. Формально команда повернення функцією результату виглядає як return nums [i], що в принципі означає повернення результатом значення елемента з індексом і з масиву nums. Але оскільки в даному випадку в описі функції перед її ім'ям стоїть інструкція &, то повертається не просто значення елемента, а посилання на цей елемент. З прикладної точки зору дане твердження означає, що через результат функції getMax () ми можемо не тільки прочитати значення найбільшого елементу, але і привласнити йому нового значення.

У функції main () створюється цілочисельним масив number розміру size, і саме з цим масивом виконуються операції з залученням функції getMax (). Так, при виконанні команди int maxNum = getMax (numbers, size) в змінну maxNum записується значення найбільшого елементу масиву. Подальша зміна значення змінної maxNum (як, наприклад, при виконанні команди maxNum = -100) на масиві numbers не позначаються - значення його елементів залишаються незмінними. Але якщо ми результат функції запишемо не в звичайну змінну, а на заслання (прикладом може служити команда int & maxRef = getMax (numbers, size)), то ця посилання стане альтернативним назвою для відповідного елемента масиву (того елемента, на який функція повертає посилання) . Тому, скажімо, виконання команди maxRef = -200 означає, що відповідний елемент масиву отримує нове значення.

4.2.9. Вказівник на функцію

Ім'я функції є вказівником на функцію. Це просте обставина можна використовувати при вирішенні значного класу прикладних задач. Далі розглядаються деякі приклади використання вказівників на функції.

Змінна, що є вказівником на функцію, описується наступним чином:

- вказується ключове слово, яке визначає тип результату, який повертає функція, на яку може посилатися вказівник;
- вказується назва для змінної-вказівника;
- перед ім'ям змінної-вказівника ставиться зірочка *, а вся конструкція з зірочки та імені вказівника ставиться в круглі дужки;
- після імені вказівника в круглих дужках перераховуються ідентифікатори типів аргументів, які передаються функції, на яку може посилатися вказівник.

У лістингу нижче представлений дуже простий приклад використання вказівників на функцію.

```
#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;
// Функції з двома аргументами (тип double і int),
```

```
// повертають результат типу double:
double f (double x, int n) {
double s = l;
for (int k = l; k <= n; k ++) {
s * = (l + x);
}
return s;
}
double g (double x, int n) {
double s = l;
for (int k = l; k <= n; k ++) {
s * = x / k;
}
return s;
}
// Функції з одним аргументом (тип int),
// повертають результат типу char:
char h (int n) {
return 'A' + n;
}
char u (int n) {
return 'Z'-n;
}
// Головна функція програми:
 int main () {
// Зміна кодування консолі:
system ( "chcp тисяча двісті п'ятьдесят один> nul");
// Змінні для передачі аргументами:
double x = 2;
int n = 3;
// Покажчики на функції:
double (* p) (double, int);
char (* q) (int);
double (* r) (double);
// Використання покажчиків на функції:
P = f;
cout << "|" << p (x, n) << "|";
p = g;
cout << p (x, n) << "|"
q = h;
cout << q (n) << "|"
q = u;
```

```
cout << q (n) << "|";
r = exp;
cout << r (x / 2) << "|";
 r = log;
cout << r (x) << "| \ n";
// Затримка консольного вікна:
system ( "pause> nul");
return 0;
}
```

У програмі описані чотири допоміжні функції:

- функцією f() для аргументів x і n повертається значення $(1 + x)^n$;
- функцією g() для аргументів x і n повертається значення $x^n / n !$;
- функцією h() для аргументу n повертається символ, зміщений в кодовій таблиці символів від символу 'A' на n позицій вперед;
- функцією u() для аргументу n повертається символ, зміщений в кодовій таблиці символів від символу 'Z' на n позицій назад.

У головній функції програми оголошується три покажчика на функції:

- покажчик p оголошений командою double (* p) (double, int). Значним цього вказівником може присвоюватися ім'я функції, у якій два аргументи (перший типу double і другої типу int) і яка результатом повертає значення типу double. Наприклад, після виконання команди p = f покажчик p є альтернативним способом звернення до функції f (). Тому команда p (x, n) означає виклик функції f () з аргументами x і n. Після виконання команди p = g, вираз p (x, n) означає виклик функції g() з аргументами x і n.
- Покажчик q оголошений командою char (* q) (int). Значним цього вказівником може присвоюватися ім'я функції, у якій один аргумент типу int і яка результатом повертає значення типу char. Наприклад, після виконання команди q = h покажчик q є альтернативним способом звернення до функції h (). Команда q (n) означає виклик функції h () з аргументом n. Після виконання команди q = u, вираз q (n) означає виклик функції u() з аргументом n.
- покажчик rоголошений командою double (* r) (double). Значним цього вказівником може присвоюватися ім'я функції, у якій один аргумент типу double і яка результатом повертає значення типу double. Після виконання команди r = exp (функція для обчислення експоненти) покажчик r є альтернативним способом звернення до вбудованої математичної функції exp (). Команда r (x / 2) означає виклик функції exp () з аргументом x / 2. Після виконання команди r = log (функція для обчислення натурального логарифма) вираз r (x) означає виклик функції log () з аргументом x.

4.3. Задачі для розв'язання

1. Напишіть програму з функцією для обчислення факторіала числа (факторіал числа n! = 1 2 3 ... n - добуток натуральних чисел від 1 до цього числа). Запропонуйте різні способи організації коду (наприклад, з рекурсією і без рекурсії).

2. Напишіть програму з функцією для обчислення подвійного факторіала числа (подвійний факторіал числа n !! = n (n - 2) (n - 4) ... - добуток натуральних чисел «через одне число» - останній множник 2 для парних чисел і 1 для непарних чисел). Запропонуйте різні способи організації коду (наприклад, з рекурсією і без рекурсії).

3. Напишіть програму функцією для, обчислення чисел Фібоначчі. Аргументом функції передається номер числа в послідовності, а результатом повертається саме число. Розглянути спосіб опису функції з використанням рекурсії і без використання рекурсії.

4. Напишіть програму з функцією для обчислення біноміальних коефіцієнтів $C_n^k = \frac{n!}{k!(n-k)!}$. Параметри k і n передаються аргументами функції.

5. Напишіть програму з функцією для обчислення суми натуральних чисел. У функції використовувати рекурсію. Перевантажити функцію так, щоб другий аргумент визначав ступінь, в яку зводяться складові при підсумовуванні. Наприклад, якщо аргументами функції передані значення і та до, то результатом повертається сума $1^k + 2^k + 3^k + ... + n^k$ (при k = 1 виходить сума натуральних чисел).

6. Напишіть програму з перевантаженої функцією. Якщо функції передається один числовий аргумент, то вона повертає результатом значення цього аргументу. Якщо функції передається два числових аргументу, то вона повертає результатом суму квадратів їх значень. Якщо функції передається три числових аргументу, то вона повертає результатом суму кубів їх значень.

7. Напишіть програму з функцією, аргументами якої передаються: покажчик на функцію і масив числових значень. При виконанні функції до кожного з елементів масиву застосовується функція, передана першим аргументом (через покажчик). Передбачити перевантаження функції таким чином, щоб першим аргументом (замість покажчика на функцію) можна було передавати число (в такому випадку всі елементи масиву-аргументу множаться на дане число).

8. Напишіть програму з функцією, призначеної для обчислення похідної від деякої функції. Першим аргументом передається покажчик на диференційовану функцію, а другий аргумент - точка, в якій обчислюється похідна. Для обчислення похідною від некоторой функції f (x) використовувати наближену формулу f '(x) = (f (x + h) -f (x)) / h, де через h позначено малий приріст по аргументу (вибирається довільно). Передбачити перевантаження функції таку, щоб замість другого числового аргументу можна було передавати масив значень. В такому випадку аргументом функції передається ще один додатковий

масив, в який записуються значення похідної в кожній з точок, що визначаються елементами першого масиву.

9. Напишіть програму з функцією для обчислення значення полінома в точці. Аргументами функції передаються масив з числовими значеннями і точка, в якій обчислюється значення полінома. Якщо через ак позначити елементи масиву, переданого аргументом функції (індекс $k = 0,1,2,K,n$), А через х позначити точку, в якій обчислюється значення полінома, то результатом функції повинна повертатися сума $a_0 + a_1x + a_2x^2 + K + a_nx^n$. Перевантажити функцію так, щоб при відсутності аргументу, що визначає точку для обчислення значення полінома, функцією в консольне вікно виводилися коефіцієнти полінома (значення елементів масиву).

10. Напишіть програму з функцією, аргументом якої передається числовий масив. При виконанні функції елементи масиву сортуються в порядку зростання, а результатом функція повертає покажчик на останній елемент масиву.

11. Напишіть програму з функцією, аргументом якої передається числовий масив. При виконанні функції елементи масиву сортуються в порядку убування, а результатом функція повертає посилання на останній елемент масиву.

12. Напишіть програму з функцією, аргументом якої передається символьний масив з текстом. Результатом функції повертається довжина найдовшого слова в тексті, що міститься в масиві.

13. Напишіть програму з функцією, аргументом якої передається символьний масив з текстом. При виконанні функції текст, записаний в масив, відображається в зворотному порядку. Спробуйте реалізувати таку функцію через рекурсію.

14. Напишіть програму з функцією, аргументами якої передаються символьний масив і окремий символ. Результатом функцією повертається кількість букв, які визначаються символьним аргументом, в тексті, що міститься в символьному масиві.

15. Напишіть програму з функцією, аргументом якої передається двовимірний числовий масив. Результатом функцією повертається посилання на елемент масиву з найбільшим значенням.

16. Напишіть програму з функцією, аргументом якої передається двовимірний числовий масив. Результатом функцією повертається покажчик на елемент масиву з найменшим значенням.

17. Напишіть програму з функцією, аргументом якої передається двовимірний числовий масив. Результатом функцією повертається середнє арифметичне для елементів масиву (сума всіх елементів, поділена на їх кількість).

18. Напишіть програму з функцією, аргументом якої передається символ і двовимірний символьний масив (попередньо заповнений випадковими символами). Результатом функцією вертається кількість входжень символу в двовимірний масив.

19. Напишіть програму з функцією, аргументом якої передається символ і двовимірний символьний масив, в кожен рядок якого записаний текст. Результатом функцією повертається кількість входжень символу в двовимірний масив (на відміну від попередньої задачі, в даному випадку перевіряти потрібно в повному обсязі елементи двовимірного символьного масиву, а тільки ті, що містять текст).

Література

1. Пелешко Д.Д., Теслюк В.М. Об’єктні технології С++11: навч. Посібник. – Львів: Видавництво Львівської політехніки, 2013. – 360 с.
2. Страуструп, Б. Программирование: принципы и практика с использованием C++ (C++11 та C++14), 2-е изд.: Пер. с англ. - М.: ООО "И.Д. Вильямс", 2016. - 1328 с.
3. Павловская, Т.А. C/C++. Программирование на языке высокого уровня. — СПб.: Питер, 2003. - 461 с.
4. Павловская, Т.А., Щупак, Ю.А. C/C++. Структурное и объектно-ориентированное программирование: Практикум. — СПб.: Питер, 2011. - 352 с.
5. A Repository for Libraries Meant to Work Well with the C++ Standard Library. Електронний ресурс: www.boost.org.
6. ISO International Standard ISO/IEC 14882:2014(E) – Programming Language C++ Електронний ресурс: https://isocpp.org/std/the-standard
7. ISO /IEC 9899:2011. Programming Languages - C. The C standard.
8. ISO/IEC 1 4882:2011. Programming Languages - C++. The C++ standard.
9. Кёниг, Э., Му, Б.Э. Эффективное программирование на C++. Серия C++ In-Depth, т.2.: Пер. с англ. — М.: Издательский дом "Вильямс", 2002. — 384 с.
10. Саттер, Г. Решение сложных задач на C++. Серия C++ In-Depth: Пер. с англ. — М.: Издательский дом “Вильямс”, 2008. - 400 с.
11. Страуструп, Б. Дизайн и эволюция C++: Пер. с англ. - М.: ДМК Пресс. - 448 с.
12. Уоррен, Г.С. Алгоритмические трюки для программистов. – 2-е изд.: Пер. с англ. – М.: Издательский дом "Вильямс", 2014. – 512 с.
13. Конспект лекцій з дисципліни «Вступ до програмування ч.2». ДВНЗ «Комп’ютерна академія шаг», Львів, 2017.