Лекція 3

Тема лекції: Використання динамічних рядків та масивів. Перевантаження функцій. Аргументи функцій за замовчуванням

Багатовимірні динамічні масиви

Для оголошення покажчика на дво- або тривимірний масив (або більшої розмірності) потрібно зазначити кількість елементів у другій і наступних позиціях. Наприклад, для виділення простору для масиву *10x20* цілих чисел, потрібно почати з оголошення:

```
int (*matrix)[20];
```

Воно визначає *matrix* як покажчик на масив із 20 цілих. Далі, виділяємо пам'ять для матриці розміром 10x20 і привласнюємо їй адресу змінної *matrix*:

```
matrix = new int [10][20];
```

Для створення куба 8*8*8 треба написати:

```
int (*matrix)[8][8];
matrix = new int [8][8][8];
```

При виділенні простору для багатовимірного масиву перший індекс може бути змінною, а другий і наступні мають бути константами.

Видалення багатовимірних масивів будь-якої розмірності відбувається так, начебто вони мають лише один вимір:

```
delete[] matrix;
```

Перевантаження функцій

У C++ функції можуть мати однакове ім'я доти, доки вони відрізняються хоча б одним параметром. Функції “перевантажені”, оскільки їхні імена однакові, а виконувана ними робота різна.

```
#include <iostream. h>
int square(int a);
double square(double a);
long double square(long double a);
```

```
main()
{
 int x=10;
 double y=15.7;
 long double z=30.75;
 cout << square(x) << '\n';
 cout << square(y) << '\n';
 cout << square(z) << '\n';
 return 0;
}
int square(int a)
{
 return a*a;
}
double square(double a)
{
 return a*a;
}
long double square(long double a)
{
 return a*a;
}
```

Аргументи функцій за замовчуванням

Важлива властивість C++ - можливість передачі аргументів у функцію за замовчуванням. Вона використовується, якщо при виклику функції необхідна тільки частина параметрів, а не всі.

Припустимо, нам необхідна функція, що повертає суму чотирьох цілих значень, подібна такій:

```
int sum(int a,int b,int c,int d)
{
  return a+b+c+d;
}
```

У даному випадку для виклику функції тільки з двома аргументами

необхідно доповнити нулями два відсутні параметри:

```
cout << sum(v1,v2,0,0);
```

Але щоб зробити такий запис необхідно розібратися в документації по функції *sum()* для того, щоб визначити, якими значеннями доповнити невикористані параметри. Аргументи функцій за замовчуванням створені для захисту від використання помилкових значень параметрів при виклику функцій. Вони доповнюють необхідні значення для незаданих аргументів.

Для оголошення значення аргументу за замовчуванням в прототипі функції потрібно завершити параметр функції знаком рівності та необхідним значенням.

```
int sum(int a,int b,int c=0,int d=0);
```

Значення за замовчуванням повинні йти останніми у списку параметрів функції і можуть з'являтися тільки в прототипі функції. Тепер доступні такі оператори:

```
cout << sum(1,2);         //a=1;b=2;c=0;d=0
cout << sum(1,2,3);
cout << sum(1,2,3,4);
```

Посилки

У наступних рядках:

```
int i=12;
int *p=&i;
```

оголошується ціла змінна *i*, ініціалізована значенням 12, і покажчик на ціле *p*, що адресує і. Відповідно до цих оголошень такі два оператори виводять значення і*:*

```
cout << i;
cout << *p;
```

Вираз **p* вказує компілятору, що треба використовувати значення, що адресується покажчиком р. Використання покажчиків у якості доступу до змінних містить у собі небезпеку - якщо оператор змінив адресу, що зберігається в покажчику, то цей покажчик більш не посилається на те ж саме значення. Якщо така зміна була внесена ненавмисно, то результатом цього може

бути серйозна помилка. C++ пропонує альтернативу для більш безпечного доступу до змінних через посилки. Операція & означає посилку. Посилка на значення завжди посилається на це значення.

Посилки в якості змінних. Як змінна посилка посилається на інше значення.

Приклад: чотири засоби адресації і виведення цілого значення.

```
#include <iostream. h>
main()
{
 int ivar=1234;       //Змінній привласнюється значення
 int *iptr=&ivar;    //Покажчику привласнюється адреса ivar
 int &iref=ivar;     //Посилка асоціюється з ivar
 int *p=&iref;       //Покажчику привласнюється адреса iref
 cout << ivar;
 cout << *iptr;
 cout << iref;
 cout << *p;
 return 0;
}
```

Тут оголошуються чотири змінні. Перша - просте число *ivar*, що ініціалізує значення 1234. Наступна - покажчик на ціле з ім'ям *iptr*, якому привласнена адреса *ivar*. Третя оголошена як змінна-посилка. Рядок

```
int &iref=ivar;
```

оголошує *iref* як посилку на ціле значення. Змінній *iref* привласнюється *ivar* не як значення змінної, а як адреса в пам'яті. *iref* вказує на *ivar* подібно покажчику. Оператор

```
cout<<iref;
```

виводить значення *ivar*. Це відбувається завдяки використанню *iref* у якості посилки на місце розташування *ivar* у пам'яті. Четверте оголошення створює ще один покажчик - *p*, якому привласнюється адреса, що зберігається в *iref*. Вживання покажчика у виразі **p* дає доступ до значення *ivar*.

При використанні посилок треба пам'ятати правило: одного разу ініціалізувавши посилку, їй не можна привласнити інше значення.

Приклад: додамо в попередню програму оголошення:

```
int anotherInt;
iref = anotherInt;
```

Це привласнення не змусить *iref* посилатися на *anotherInt*. Цей вираз привласнить значення *anotherInt* об'єкту на який посилається *iref*, тобто *ivar*. Посилки повинні бути ініціалізовані при їхньому оголошенні. Наприклад, не можна написати так:

```
int &iref;
```

На відміну від покажчиків, що можуть бути оголошені в неініціалізованому стані або встановлені в нуль, посилки завжди посилаються на об'єкт. Для посилок не існує аналогу нульового покажчика.

Посилки не можна ініціалізувати значенням у чотирьох особливих випадках:

- при оголошенні їх із ключовим словом *extern*;
- при використанні в якості параметрів функції;
- при використанні в якості типу, що повертається значенням функції;
- в оголошеннях класів.

Не існує операторів, що безпосередньо роблять дії над посилками. Всі операції відбуваються тільки над відповідними об’єктами. Припустимо, додамо оператор

```
iref++;
```

Компілятор згенерує код, що інкрементує *ivar* - змінну, на яку посилається *iref*. Оператор не вчинить ніяких дій безпосередньо над *iref*.

Посилки також можуть посилатися на константи. Привласнити значення безпосередньо посилкам не можна. Оголошення

```
int &iref=1234;
```

є помилкою, тому що 1234 не є об'єктом, на котрий *iref* може посилатися.

Можна оголосити константну посилку на об'єкт так:

```
int x = 1234;
const int &iref = x;
```

Цей прийом ефективно перетворює змінну на константу, яку не можна піддати змінам. Значення змінної *x* можна змінити явно, але привласнити *x* нове значення через посилку не можна, тому що *iref* - константа.

Посилки в якості параметрів функцій. Самі по собі змінні-посилки використовуються рідко. Замість того, щоб використовувати посилки для доступу до іншої змінної, легше використовувати саму цю змінну В якості параметрів функції посилки мають більш широке застосування. Посилки корисні у функціях, що повертають два або більше значень, що вводяться.

Приклад використання посилок у якості параметрів функції: обчислення сумарного податку і роздрібної ціни по відомій загальній вартості покупки і податкового тарифу.

```
#include<iostream.h>
#include<stdlib.h>
#include<stdio.h>
double GetDouble(const char *prompt);
void GetData(double &paid,double &rate);
void  Calculate(const  double  paid,const  double  rate,double
&list,double &tax);
main()
{
 double paid,rate,list,tax;
 GetData(paid,rate);
 Calculate(paid,rate,list,tax);
 printf("List prise=$%8.2f\n",tax);
 return 0;
}
double GetDouble(const char *prompt)
{
 char s[20];          //Рядок для введення
 printf(prompt);      //Відображення рядка запиту
 scanf("%20s",s);     //вивести дані як рядок
 return atof(s);      //повернути введені дані
}
//Змінюються значення paid і rate
void GetData(double &paid, double &rate)
{
 paid = GetDouble("Price paid? ");
 rate = GetDouble("Tax rate (ex:. 06)? ");
```

```
}
//Змінюються значення list і tax
void Calculate(const double paid,const double rate,double
&list,double &tax)
{
 list = paid/(1+rate);
 tax = paid - list;
}
```

У програмі визначені три функції.

Перша - *GetDouble(const char *prompt)* запитує і повертає дійсне значення подвійної точності.

Друга - *GetData(double &paid,double &rate)* демонструє використання параметрів-посилок для повернення двох значень. До функції *GetData()* передаються адреси параметрів замість їхніх значень.

Розглянемо тіло функції *GetData()*. У ньому двічі викликається *GetDouble()* з аргументом-рядком, який повинен відображуватись, і отримані значення привласнюються параметрам-посилкам *paid* та *rate*. Оскільки *paid* та *rate* - посилки, значення привласнюються об’єктам, переданим функції *GetDouble()*.

Третя функція обчислює ціну і податок до сплати, базуючись на дійсній ціні ставки податку. Тут *paid* і *rate* оголошені константами і тому функція їх не змінює. Передаються їхні значення. Параметри *list* і *tax* - вхідні параметри функції. Вони змінюються функцією і, отже, передаються за посилкою. І оскільки параметри - це посилки, у дійсності використовуються змінні, передані функції *Calculate()*. Операція привласнення відбувається за посилкою, внаслідок чого змінюються дійсні змінні, коли у функції *main()* відбувається виклик функції *Calculate()*.

Аргументи, що передаються за значенням, неможливо відрізнити від аргументів, що передаються за посилками. При використанні параметрів-посилок необхідно гарне документування програм.

Посилки і покажчики в якості параметрів тісно пов'язані. Посилки можуть інтерпретуватись компілятором C++ як покажчики. Розглянемо таку

функцію:

```
void f(int *ip)
{
  *ip = 12;
}
```

Всередині цієї функції здійснюється доступ до переданого аргументу, адреса якого зберігається в покажчику *ip*. Вираз *f(&ivar)* передає адресу *ivar*. Значення 12 привласнюється змінній *ivar*. Аналогічна функція, що використовує параметри-посилки виглядає так:

```
void f(int &ir)
{
 ir = 12;
}
```

Покажчик **ip* замінений посилкою *ir*, якій функція привласнює значення 12. Вираз *f(ivar);*

привласнює значення об'єкту-посилці: передає *ivar* за посилкою до функції *f()*. І оскільки *ir* посилається на *ivar*, насправді значення 12 привласнюється *ivar*.

Посилки в якості результатів функцій. Функції можуть повертати посилки на об'єкти за умови, що ці об'єкти існують. Наприклад, оголосимо функцію:

```
double &ref(double d);
```

Їй необхідний один дійсний аргумент подвійної точності. Повертає вона посилку на об'єкт того ж типу, оголошений десь в іншому місці. Це можна використовувати при приховуванні деталей реалізації внутрішньої структури даних, що дозволяє проводити в ній зміни без відповідних змін у коді.