Лекція 7. Додаткові можливості функцій в С++

1. Вказівники на функції та їх використання.

2. Перегрузка функцій.

3. Створення аналогів функцій за допомогою макропідстановки.

4. Inline – функції.

1. Вказівники на функції та їх використання

Часто доводиться програмувати функції, які можуть за вибором звертатися до однієї із цілої множини функцій. Наприклад, в програмі для обчислення означеного інтеграла можна задати за допомогою параметрів межі інтегрування, але можна отримати набагато універсальніший варіант, якщо дозволити користувачеві самому обирати підінтегральну функцію. В таких ситуаціях зазвичай використовують вказівник на функцію як параметр іншої функції. Наприклад, вказівник pf на функцію f(x), аргумент і результат якої мають тип double, можна оголосити так:

```
double (*pf)(double x);
```

До прикладу, розглянемо функцію, що обчислює інтеграл методом прямокутників для довільної заданої підінтегральної функції :

```
double int_rect(double a, double b,
                        double (*f)(double x))
{ int i, n=100;
  double s=0,h=(b-a)/n;
  for(i=0; i<=n; i++) s += f(a+i*h);
  return s*h;}
```

В якості параметра ***f*** можна використати ім'я будь-якої стандартної функції з бібліотеки math.h:

```
cout << int_rect(0,M_PI,sin) << endl;
     //результат= 1.99984
cout << int_rect(0,M_PI,cos) << endl;
     //результат=-4.18544e-17
```

Як правило, функції, що повертають значення, використовуються в правій частині оператора присвоєння. Однак функції в якості свого значення

можуть повертати вказівники і посилання. А за вказівниками і посиланнях можливий запис. Такі функції називають "лівими" (ще один представник left-value). Наведемо як приклад функцію, що повертає посилання на максимум з двох своїх аргументів:

```
double& max(double &x, double &y)

{ return (x>y)? x : y; }
```

Звичайне використання:

```
double r=max(a,b);
```

«Ліве» використання:

```
double a=5,b=6;

max(a,b)=10;  //еквівалентно b=10;
```

Аналогічний варіант з вказівниками:

```
double* max(double *x, double *y)

{ return (*x>*y)?*x:*y; }

.......................

  double a=5,b=6;

  *max(&a,&b)=10;
```

Можна заборонити «ліве використання» функції, що повертає вказівник або посилання, для цього достатньо розмістити напочатку її заголовка специфікатор ***const***:

```
const double& max(double &x, double &y)

{ return (x>y)? x : y); }
```

В пізніх стандартах С++, за аналогією до популярних сучасних технологій, реалізовано можливість використання так званих лямбда-виразів. Вони надають можливість реалізовувати функції безпосередньо в місці їх виклику, що в поєднання з вказівниками на функції надає програмісту досить цікавий і динамічний інструмент розробки. До прикладу, використати розглянуту вище функцію наближеного обчислення інтеграла до знаходження $\int_0^1 (x^2 - x^3)dx$ з використанням «лямбди» можна в такий спосіб:

```
cout << int_rect(0, 1, [](double x)->double

                  {return x * x - x * x * x; }) << endl;

   //результат=0.083325
```

На жаль, більш детальне вивчення цього цікавого механізму виходить за межі нашого курсу.

2. Перегрузка функцій

В ранніх версіях алгоритмічної мови ФОРТРАН було занадто багато функцій які дублювали одна одну для різних типів аргументу. Наприклад:

• sin (x) - обчислює синус для дійсного аргументу x;

• dsin (DX) - обчислює синус для дійсного аргументу з подвоєною точністю;

• csin (CX) - обчислює синус для комплексного аргументу cx;

• cdsin (CDX) - обчислює синус для комплексного аргументу з подвоєною точністю.

Це зумовлено тим, що для кожної з цих функцій в системній бібліотеці існує свій алгоритм і своя програма. Значно простіше для програміста користуватися загальноприйнятим в математиці позначенням sin(x) а компілятор повинен сам, залежності від типу аргументу, визначити яким алгоритмом скористатися і яка точність потрібна. Ця ідея була реалізована в наступних версіях Фортран і для користувача кількість математичних функцій "зменшилася" майже в 4 рази.

У мові С спостерігається приблизно така ж картина, як і в ранніх версіях Фортран. Уявіть собі, що ми часто використовуємо в програмі форматний вивід числових скалярних величин різного типу. Було б зручно виділити такі операції в окремі функції. Ось як це мало б виглядати на мові С:

```
void print_int(char *nx,int x)

{ printf("\n%s=%d",nx,x); }

void print_float(char *nx,float x)

{ printf("\n%s=%f",nx,x); }

void print_double(char *nx,double x)

{ printf("\n%s=%lf",nx,x); }
```

В мові С++ це можна зробити так:

```
void print (char *nx,int x)

{ printf("\n%s=%d",nx,x); }

void print (char *nx,float x)

{ printf("\n%s=%f",nx,x); }

void print (char *nx,double x)

{ printf("\n%s=%lf",nx,x); }
```

Тобто в мові С++ дозволяється створювати декілька функцій з однаковими іменами, які відрізняються одна від одної, наприклад, типом аргументів, їх кількістю, типом значення, що повертається. Проте, якщо функції будуть відрізнятися лише типом значення, що повертається, компілятор не зможе зробити правильного вибору функції.

3. Створення аналогів функцій за допомогою макропідстановки

Заміна однієї послідовності символів в тексті програми на іншу реалізується за допомогою макропідстановки ***#define*** (від англ. - Визначити):

```
#define s1s2s3...sn q1q2...qm
```

При цьому послідовність символів ***s1s2s3...sn*** в коді початкової програми буде замінено на послідовність ***q1q2...qm***. Пропуски перед послідовністю заміни та після ігноруються. Заміні не піддаються значення текстових стрічок і коментарі. Фрагмент заміни може бути і багаторядковим. У цьому випадку в кінці кожного рядка поміщається символ переносу "\". У наведеній нижче програмі міститься декілька найбільш характерних прикладів використання директиви #define:

```
#include <stdio.h>
#include <conio.h>
#define Nmax 100
#define max(a,b) ((a)>(b))?(a):(b)
#define print(a) printf("\n%s = %d \n", #a, a); \
void main ()
{int x = 5, y = 8;
int z = Nmax;
int w = max (x * y, z);
print (x); print (y); print (z); print (w);
}
/ / === Результат роботи ===
x = 5
y = 8
z = 100
w = 100
```

Звернемо увагу на деякі тонкощі в наведених підстановках. По-перше, аргументи макровизначення функції max у виразі заміни взято в круглі дужки. Це дозволяє при виклику такої функції передавати цілі вирази. Нехай в програмі досить часто доводиться використовувати операцію піднесення до квадрату. Якщо з цією метою використати макропідстановку ***Square(x)*** і визначити її без використання дужок (***#define Square(x) x*x***), то для звертання ***Square(1+z)*** компілятор замінить ***1+z*1+z***, тобто при обчисленні отримаємо значення ***2*z+1***, а не ***(z+1)*(z+1)***.

4. Inline – функції

Вбудовані функції - це дуже короткі функції, реалізовані невеликим числом машинних команд. До них невигідно звертатися з використанням стандартного механізму, що вимагає обов'язкової запису переданих аргументів у стек, отримання даних з стека, запису результату в стандартний регістр і т.п. Набагато простіше на місце виклику інлайн-функції вставити і налаштувати саме тіло функції. Це набагато ефективніше, особливо в тих випадках, коли робота функції зводиться до кількох машинних команд. Така техніка компіляції нагадує процедуру макропідстановки за допомогою директив ***#define***, розглянуту вище.

Типовими прикладами вбудованих функцій є процедури визначення абсолютної величини ***abs(x)***, вибору максимального або мінімального значення з двох аргументів і т.п. Іноді, з метою оптимізації вузьких місць в програмі, корисно попросити компілятор застосувати техніку вбудовування до часто вживаних функцій користувача. Прямою вказівкою про те, що функція повинна бути вбудованою, є службове слово у заголовку функції:

```
inline int even(int x)
{  return !(x%2); }
inline double min(double a,double b)
{ return a < b ? a : b; }
```

Використання вбудованих функцій пришвидшує роботу самої програми з одного боку, а з іншого надає програмістові можливість писати програмний код у звичному стилі. Це доволі зручно, наприклад, в об'єктно-орієнтованій частині C++ діє домовленість, за якою функції реалізовані всередині оголошення класу вважаються inline-функціями за замочуванням, тобто без вживання позначки ***inline***.

Не кожну функцію компілятору вдається вбудувати в місце її виклику. Наприклад, якщо у функції присутня ***рекурсія*** (виклик самої себе). В таких випадках компілятору дозволено ігнорувати вказівку про вбудування функції і задіювати стандартний механізм виклику з використанням стеку.