Лекции по C/C++: функции

Функция – это независимая часть программы, решающая некоторую частную задачу.

Сложная программа *всегда* строится как совокупность функций. Это позволяет решить следующие проблемы:

- лёгкое повторное использование однажды написанного кода;
- лучшее структурирование кода за счет разбиения задачи на независимые подзадачи;
- оптимизация и уменьшение размеров кода за счет неоднократного использования (вызова) в нём функций.

Общий вид описания функции следующий:

```
тип имя (тип1 параметр1, ..., типN параметрN)
```

Первый тип показывает, какого типа значение вычисляет (*возвращает*) функция. Если функция не должна вычислять "одного ответа", для неё может быть указан пустой тип `void`. Такую функцию нельзя будет вызвать из выражения (справа от знака `=`). Имя функции назначается программистом и строится по тем же правилам, что любой другой идентификатор. Каждый параметр (аргумент) функции может быть любого из разрешённых в языке или добавленных программистом типов данных.

Пример. `int count (int a, int b)`

Функция с именем `count` вычисляет целое значение и имеет 2 целых входных величины, обозначенных `a` и `b`.

Функция обменивается данными с программой только через свои параметры.

Если параметр *выходной* (его значение должно вычисляться внутри функции) перед его именем ставится знак "`&`":

```
void equation (float a, float b, float c,
               float &x1, float &x2)
```

Функция с именем `equation` здесь имеет 3 входных вещественных величины, обозначенных `a`, `b`, `c` и 2 выходных - `x1`, `x2`.

Тело функции, как и тело оператора цикла, всегда заключено в операторные скобки `{ … }`

Например, показанная ниже функция умеет вычислять расстояние от точки на плоскости с координатами `(x,y)` до начала координат `(0,0)`

```
float distance (float x, float y) {
 float r=sqrt(pow(x,2)+pow(y,2));
 return r; //Вернуть из функции ответ
}
```

"Ответ" должен быть именно того типа данных, который указан перед именем функции! После ключевого слова `return` можно писать также выражение того типа, что указан перед именем функции. Тело нашей функции могло бы выглядеть и так:

```
 return sqrt(pow(x,2)+pow(y,2));
```

Конечно, чаще функции состоят не из одного оператора, а из нескольких.

Написание функции ещё не означает её выполнения. Для выполнения функцию надо *вызвать*, указав её имя и список *фактических параметров*, количество и типы которых соответствуют *формальным параметрам* в заголовке функции. Например, нашу функцию мы могли бы вызвать так:

```
float x1=3.5,y1=-1;
float d1 = distance (x1,y1);
```

Здесь формальные параметры - величины `x` и `y`, указанные в заголовке функции, а фактические параметры `x1` и `y1` будут подставлены вместо `x` и `y` при вызове функции.

Вот такой способ вызвать нашу функцию тоже был бы верным:

```
float d2 = distance (1.,1.5);
```

Здесь фактические параметры - вещественные константы `1.` и `1.5`, они будут использованы функцией в качестве значений `x` и `y` при очередном выполнении функции.

А вот следующий оператор ошибочен:

```
float d3 = distance (x1,y1,1.);
```

Здесь передается 3 фактических параметра, но функция готова принять только два.

Если функция реализована (записана) выше по тексту программы, чем она вызвана, никаких дополнительных действий для правильной работы этого вызова предпринимать не нужно. Если это не так, следует указать *прототип* (заголовок) функции выше по тексту программы, чем она вызвана:

```
int summa (int a, int b) { return a+b; }
int main () {
 int s1=summa (3,7);
}
```

В этой программе функция `summa` будет успешно вызвана из главной функции `main`. А вот во второй программе, где тело функции `raznost` записано после тела `main`, при отсутствии прототипа возникнет ошибка компиляции:

```
int raznost(int,int); //это прототип
int main(){
 //Если функцию исползует только main,
 //прототип можно бы было записать и здесь
 int r1=raznost (5,1);
}
int raznost (int a,int b) { return a-b; }
```

Обычно прототипы всех функций программного модуля собраны в файл с типом `.h` и подключены к модулю оператором вида

```
#include "unit1.h"
```

Если перед прототипом указано ключевое слово `extern`, это означает, что функция подключается из другого файла проекта.

В качестве примера напишем функцию для проверки номера координатной четверти точки на плоскости. После этого выведем номера четверти для 2 заданных точек.

Сначала определимся с аргументами и типом функции:

- 2 входных параметра, координаты точки `x` и `y` (вещественные);
- 1 выходной параметр, целый номер четверти `n`, принимает значения от 1 до 4 или 0, если хотя бы одна из координат равна 0.

Так как ответ, вычисляемый функцией, всего один, он может быть возвращён непосредственно через тип функции.

```
#include <stdio.h>
int quadrant (float x, float y) {
 int n=0;
 if (x>0) {
  if (y>0) n=1;
  else if (y<0) n=4;
 }
 else if (x<0) {
  if (y>0) n=2;
  else if (y<0) n=3;
 }
 return n;
}
int main () {
 float x1,y1,x2,y2;
 printf ("\nВведите точку 1:");
 fflush (stdin); scanf("%f %f",&x1,&y1);
 printf ("\nВведите точку 2:");
 fflush (stdin); scanf("%f %f",&x2,&y2);
 int n1=quadrant(x1,y1);
 printf ("\nОтвет 1 = %d", n1);
 printf ("\nОтвет 2 = %d", quadrant(x2,y2));
 getchar(); return 0;
}
```

По коду видно, что в такой программе полезна была бы ещё и функция для ввода с клавиатуры координат точки:

```
void point_input (float &x, float &y) {
 printf ("\nВведите коорд X:");
 fflush (stdin); scanf("%f",&x);
 printf ("\nВведите коорд Y:");
 fflush (stdin); scanf("%f",&y);
 return; //можно не писать для void ф-ций
}
```

Её вызов из функции `main` мог бы быть таким:

```
point_input (x1,y1);
point_input (x2,y2);
```

Главная функция превратилась бы в последовательность вызова функций приложения, решающих его частные задачи. Именно так обычно устроена хорошо написанная программа.

Так как в C/C++ размерность массива нигде не хранится, при передаче вектора в функцию следует использовать 2 параметра - целочисленная размерность вектора и адрес нулевого элемента. Например, функция `func` заполняет массив нужной размерности последовательно идущими числами `1,2,3,…`:

```
void func (int n, int a[]) {
 //В функции работаем с массивом int a[n]
 for (int i=0; i<n; i++) a[i]=i+1;
}
```

Вызовем эту функцию:

```
 int a[3]; func (3, &a[0]);
 int b[5]; func (5, b);
```

Прототип функции может иметь вид:

```
void func (int, int []);
```

Решим ещё одну задачу с функциями и векторами: найти максимальную из длин векторов A = (4, 5, 6, 3) и B = (1, 2, 3, 4, 5). "Длину вектора" будем понимать в обычном декартовом смысле – как квадратный корень из суммы квадратов всех элементов вектора.

```
#include <stdio.h>
#include <math.h>

float length (int n, float v[]) {
 float d=0.;
 for (int i=0; i<n; i++) d += v[i]*v[i];
 return sqrt(d);
}

int main () {
 float a[] = { 4, 5, 6, 3};
 float b[] = { 1, 2, 3, 4, 5};
 float d1 = length (4,&a[0]);
 float d2 = length (5,b);
 printf ("\nResult=%f",(d1>d2?d1:d2));
 getchar(); return 0;
}
```

Для передачи матрицы в функцию есть 2 способа:

I. Передать 2 размерности (количество строк и столбцов) плюс двойной указатель, если матрица - динамическая:

```
void func (int n, int m, int **a) {
 //n – количество столбцов,
```

```
 //m – количество строк,
 //**a – ссылка на данные матрицы
 //внутри функции работаем с матрицей a[n][m]
}

//...
//Код в main:
int **a; //ссылка на матрицу
const int n=4, m=3; //число строк и столбцов
//Затем выделена память под a[n][m]
a = new int * [n];
for (int i=0; i<n; i++) a[i] = new int [n];
//Затем элементы в двойном цикле вводятся
//с клавиатуры или любым другим способом

//...
func (n,m,a); //вызов функции с аргументом-матрицей
```

II. Описать матрицу как вектор, исходя из следующих соображений. При заданной в задаче матрице `a[n][m]` определим вместо неё вектор `v[n*m]`. Тогда обращению к элементу матрицы `a[i][j]` соответствует обращение к элементу вектора `v[i*m+j]`.

Почитать в пособии: глава 5

Оглавление серии

теги: c++ учебное

05.11.2015, 07:54; рейтинг: 5412