Программирование и алгоритмизация

Санкт-Петербургский государственный политехнический университет

26 ноября 2014

Пример передачи параметра по ссылке

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

int main()
{
  int c = 5, d = 6;
  sum(c, d); //???
  sum(5, 6); //???

  return 0;
}
```

Выражения в качестве параметров

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

int div(int *a, int *b)
{ return *a / *b; }

int main()
{
  int c = 5, d = 6;
  int res = div(&c, &d); //???
  res = sum(c+d, div(&c, &d)); //???
  res = sum(c, (cin >>d, d));
}
```

При формировани параметра при вызове функции, как сложных выражений, они должны быть предварительно вычислены.

Примеры функций

Написать функцию:

- определения длины строки;
- копирования строки;
- сравнения строк.

Передача массивов в качестве параметров

```
//arr.h
const int N = 4, M = 5;

//arr.cpp
#include "arr.h"

int F(int arr[ ], int dArr[ ][M])
{
   sizeof(arr); //???
   sizeof(dArr); //???
}
```

Компилятор C/C++ никогда не передает массивы в функцию по значению, т.к. копирование значений массива является очень не эффективным способом передачи аргумента. Компилятор передает только указатель на массив.

Передача в функцию ссылки на массив

```
//arr.h
const int N = 4, M = 5;

//arr.cpp
#include "arr.h"

int F(int (&arr)[N], int (&dArr)[N][M])
{
   sizeof(arr); //???
   sizeof(dArr); //???
}
```

Возможно только в C++, имя ссылки — является пснвдонимом имени самого массива, а не указателем. При использовании ссылок на массивы, должны быть указаны все размерности.

Функции с параметрами по умолчанию

Применяется в языке C++. Механизм позволяет задавать значения для параметров по умолчанию и опускать при вызове такие параметры.

```
int f(int q, char c);

f(5, 'A'); //OK
f(5); //ОШИБКА

int f(int q, char c = 'X');
f(5); //OK
```

Опускать список фактических параметров при вызове функции можно только подряд с конца списка параметров (справа налево). Т.к. компилятор не будет понимать, какой из параметров был опущен.
Значения по умолчанию, должны быть указаны при объявлении функции, т.к. компилятор видит только прототип функции при формировании вызова функции.

Перегрузка имен функций

Одной из особенностей C++ является возможность перегрузки имен функций, т.е. использования одного и того же имени для нескольких разных функций.

```
int max(int a, int b); // (1)

int max(int *a, int *b); //(2)

int main()
{
  int v1 = 5, v2 = 7;
  max(v1, v2);
  max(&v1, &v2);
}
```

Рекурсивные функции

Рекурсивная функция — функция вызывающая сама себя.
Рекурсивные вычисления выполняются повторным выполнением одного и того же кода с разными наборами данных.
Достоинством рекурсивных функций является возможность создания компактного кода .
Специфика рекурсивных функций:

- программист должен обеспечить внутри рекурсивной функции не только анализ, но и обязательное выполнение условия, при котором произойдет выход из рекурсии.
- По мере возможности следует избегать использования в рекурсивной функции локальных переменных.

Вычисление факториала без рекурсии

```
int Fact(int n)
{
  int res = 1;
  for(int i = n; i > 1; i--)
     res *= i;
  return res;
}

cout << Fact(5);
```

Вычисление факториала рекурсией

```
int F(int n)
{
  if(n <= 1)
    return 1; //выход из рекурсии
  else
    return n * F(n-1);

}
```

Возвращаемые значения

Возвращамые значения могут быть:

- Адреса(указатели или ссылки);
- Базового типа;
- Пользовательского типа.

Проблемы при возвращении адреса

```
char *f()
{
  char arr[] = "ABC";
  return arr;
}

int& f2()
{
  int value = 5;
  return value;
}

int q = f2(); //???
char *c = f(); //???
```

НЕЛЬЗЯ возвращать адреса локальных переменных и других переменных время жизни которых не гарантировано.

Проблемы при возвращении адреса

Можно возвращать:

- Указатель или ссылку на объект, которая находится в вызывающей функции.
- Указатель или объект со статическим временем существования.
- Указатель на строковый литерал.
- Указатель на динамически созданный объект.

Реализация инкремента посредство функций

```
int postfixINC(int &x)
{
  int tmp = x;
  x++;
  return tmp;
}

int &prefixINC(int &x)
{
  ++x;
  return x;
}

int x = 5;
int a = prefixINC(x); //???
a = prefixINC(x)++; //???
a = postfixINC(x); //???
a = postfixINC(x)++; //???
```

Возвращение указателя на многомерный массив

```
int (*f())[20]
{
    static int arr[10][20];
    return arr;
}

int main()
{
  int (*p)[20] = f();
  p[1][1] = 5;
}
```

Функции с переменным числом параметров

Язык C/C++ допускает функции, где количество параметров в программе может изменяться.

- Признаком функции с переменным числом параметров является многоточие (...).
- Встретив монготочие в программе, компилятор прекращает проверять соответствие типов параметров при вызове функции.
- У такой функции должен быть хотя бы один обязательный параметр.
- Программист в функции с переменным числом параметров, должен иметь возможность определения точного числа передаваемых в функцию параметров.

Примеры функций с переменным числом параметров

```
int function(int a, ...);

int function2(int a, const char *, ...);

function(4, 5, 3); //???

function2(4); //???
function2(4, "Hello"); //???
```

Примеры функций с переменным числом параметров

Написать функцию с переменным числом параметров соответствующую вызову. В качестве первого параметра передается количество дополнительных аргументов. Далее следуют элементы типов: int, char, double, int, char, double и т.д.:

```
int sum(6, 7, 'A', 1.1, 4, 'B', 2.2, 5);
```

stdarg

Стандартная библиотека предоставляет несколько макросов для манипуляции с параметрами функции с переменным числом аргументов.

```
#include <cstdarg>

va_list p; //универсальный указатель
va_start(p, first); //first – имя последнего
    обязательного параметра в функции
va_arg(p, int); //формирование текущего элемента, и
     сдвиг на следующий элемент.
va_end(p); //обнуляет универсальный указатель
```

Указатели на функции

Указатель на функцию содержит стартовый адрес функции. Тип указателя зависит от количества параметров и от возвращаемого значения. При определении имя функции является ее адресом.

```
double max(double a, double b) { /* ... */ }
double min(double a, double b) { /* ... */ }

int main()
{
  double (*pf)(double, double) = max; //???
  double *pf(double, double) = max; //???

  pf = min;
  pf = max;
  pf = &min;
}
```

Вызов функции по указателю

```
int main()
{
  double (*pf)(double, double) = max; //???

  double y = pf(5.5, 4.4); //max(5.5, 4.4);
  y = (*pf)(4, 7.7);
}
```

Массивы указателей на функции

```
double (*arf[])(double) = {sin, cos, tg};
int res1 = arf[i](5.5);
res1 = (*arf[i])(7.5);
```