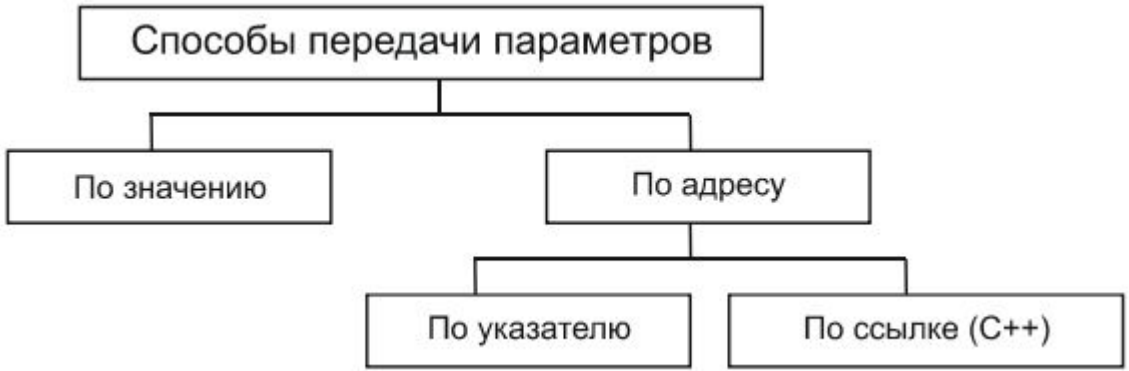
Модульное программирование (часть 2)

Функции. Способы передачи параметров

Существуют два способа передачи параметров функции:

- по значению;
- по адресу;
  - указатель;
  - ссылка; (след. семестр).



Функции. Передача параметров по значению

- **Передача параметров по значению**
  - (англоязычный эквивалент Call-By-Value)
- это простая передача копий переменных в стеке;
  - никаких изменения значений самих переменных;

Пример в коде…

Передача параметров по значению применяется:

- общий объем передаваемых аргументов невелик;
- вы намеренно предоставляете функции копию большого объекта для использования;
  - гарантируя неизменность оригинала;

Функции. Передача параметров по адресу

- Этот способ предполагает, что в качестве аргументов функция получает не копии объектов, а их адреса;
  - гораздо эффективнее
    - (и одновременно гораздо опаснее);
  - передачи копии самого объекта.
    - обладая адресом объекта, программист получает возможность изменить значение объекта по этому адресу;

Мы рассмотрим только передачу адреса посредством **указателя**.

Функции. Передача параметров по адресу

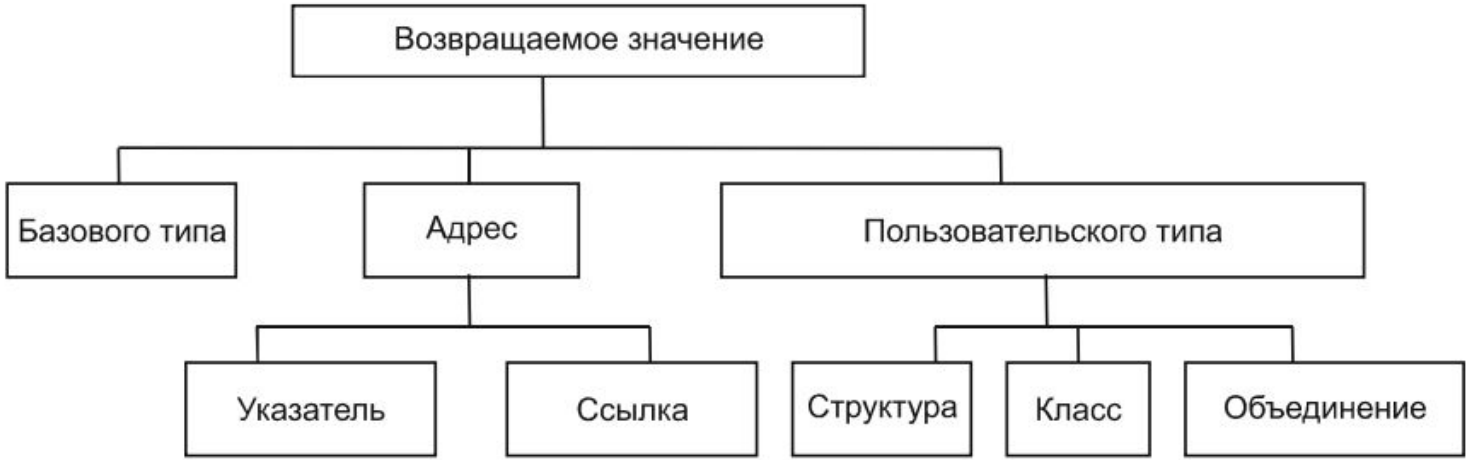
Такая передача используется для получения в функции доступа к массивам и другим большим объектам:

И пример снова в коде

Функции. Возвращаемое значение

Функция может возвращать:

- значение одного из базовых типов;
- объект пользовательского типа;
- адрес;



Функции. Возвращаемое значение

Пример 1. Возвращаемое функцией значение можно присвоить переменной подходящего типа или использовать другими способами.

```
bool Func();
int main()
{
   bool b = Func(); //возвращаемое функцией значение будет присвоено
                    переменной b
//или
       if(Func()){…}//возвращаемое значение используется для формирования
                    условия
}
```

Функции. Возвращаемое значение

Пример 2. Поиск минимального значения в массиве.

```
int* Min(int ar[ ], unsigned int n)
{
        int* p = ar; //предполагаем, что минимальным является нулевой
                     элемент массива и направляем на него указатель
        for(int i = 0; i<n; i++)
        {
                if(ar[i]<*p) //а если i-тое значение оказалось меньше
                {
                        p = &ar[i]; //перенаправляем на него указатель
                }
        return p; //в результате в переменной p будет адрес элемента
                  массива с самым маленьким значением
}
```

Функции. Возвращаемое значение. Проблемы

- При возвращении указателя из функции объект, на который указывает возвращаемое значение, должен существовать после возврата из функции;
- Предыдущий пример корректен,
  - т. к. возвращается адрес элемента массива, который, безусловно, существует после возврата из функции;

```
int* f1(int n)
{
        int nN = n*5;
        ...
        return &nN; //никогда так не делайте! Ваша функция вернет
                    указатель на область памяти в стеке, которая после
                    возвращения из функции может быть задействована
                    компилятором для других целей
}
```

Функции. Возвращаемое значение. Проблемы

Возвращать можно адреса таких объектов, которые гарантированно существуют после возврата из функции:

- указатель на объект, который располагается в стековом кадре вызвавшей функции;
- указатель на объект со статическим временем существования;
- указатель на строковый литерал (для него память будет выделена на все время выполнения программы);
- указатель на динамически созданный объект.

Функции. Перегрузка имен

- Одной из особенностей C++ является возможность перегрузки имен функций;
  - т. е. использования одного и того же имени для нескольких разных функций;
- Техника перегрузки неявно используется компилятором для базовых операций C/C++:
  - существует только одно имя для операции сложения (+), но вы используете это имя для сложения целых чисел, чисел с плавающей точкой, прибавляете целое к указателю;
    - не задумываясь о том, что компилятор при этом генерирует совершенно разные машинные инструкции;
- Идея перегрузки операций легко распространяется на функции, определяемые пользователем;
  - Цель компилятора состоит в том, чтобы использовать функцию с наиболее подходящими аргументами;

Функции. Перегрузка имен

Пример 1. Если такой возможности (печаль, конечно =(( )

```
int MaxInt(int x, int y) //функция находит максимальное значение из двух
                         параметров типа int
{ return (x>y) ? x : y; }
double MaxDouble(double x, double y) //тоже находит максимум, только
                    принимает параметры другого типа, а это означает,
                    что компилятор должен сгенерировать совершенно
                    разные низкоуровневые инструкции при действиях
                    с этими значениями (несмотря на то, что текст
                    на языке высокого уровня выглядит одинаково)
{ return (x>y) ? x : y; }
```

Функции. Перегрузка имен

Пример 2. В языке C++ программист может давать разным функциям одно и то же имя.

```
int Max(int x, int y)
{ return (x>y) ? x : y; }
double Max(double x, double y)
{ return (x>y) ? x : y; }
//Следующая функция по смыслу тоже претендует на имя Max:
int Max(int ar[], int n) //функция находит максимальный из элементов
                         массива
{
        int max = ar[0];
        for(int i=1; i<n; i++)
        {
                if(ar[i]>max) max=ar[i];
        }
        return max;
}
```

Функции. Перегрузка имен

При перегрузке имен функций действуют следующие ограничения:

- в сигнатуру функции входит только имя и число и тип аргументов;
  - но не возвращаемое значение;

```
void f(int);
double f(int);
int main()
{
  f(5); //компилятор не понимает, какую функцию хочет вызвать
        программист, т. к. возвращаемое значение принимать
        необязательно
}
```

Функции. Рекурсия

Рекурсивные функции — это функции, вызывающие сами себя.

Рекурсивные вычисления выполняются повторным выполнением одного и того же кода с разными наборами данных.

Каждое выполнение тела функции имеет **свою область** стека.

Достоинством рекурсивных функций является возможность создания компактного кода.

Недостатками рекурсивных вычислений являются: затраты времени на вызов функции и передачу ей копий параметров;

...а также затраты памяти для организации каждого вложенного вызова.

Функции. Рекурсия

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;

Без рекурсии в большинстве случаев можно обойтись.

```
{
        int n=5;
        int res=1;
        for(int i=n;i>1;i--) res*=i;
}
```

Функции. Рекурсия

Классический пример использования рекурсии для вычисления факториала:

```
            //            очевидно, что при каждом вызове функции
            компилятор должен отвести в стековом кадре память под
            локальную переменную x. Это как раз тот случай, когда
            без этой локальной переменной можно обойтись
int F(int n)
{
       if(n<=1)return 1; //обеспечили выход из рекурсии
       else
       {
              return n * F(n-1);
       }
}
```

Функции. Рекурсия