Функции используются для наведения порядка в хаосе алгоритмов.

Б. СТРАУСТРУП

ФУНКЦИИ

(продолжение)

ФУНКЦИИ

- Перегрузка функций.
- Шаблоны функций.
- Функция main().

Перегрузка функций

- Использование нескольких функций с одним и тем же именем, но с различными типами параметров, называется **перегрузкой функций.**

ПЕРЕГРУЗКА ФУНКЦИЙ.

- *Часто бывает удобно, чтобы функции, реализующие один и тот же алгоритм для различных типов данных, имели одно и то же имя.*

- *Если это имя **мнемонично**, то есть несет нужную информацию, это делает программу более понятной, поскольку для каждого действия требуется помнить только одно имя.*

ПЕРЕГРУЗКА ФУНКЦИЙ.

Компилятор определяет, какую именно функцию требуется вызвать, ***по типу фактических параметров****.*

- *Этот процесс называется разрешением перегрузки (перевод английского слова* ***resolution*** *в смысле* ***«уточнение»****).*
- *Тип возвращаемого функцией значения в разрешении не участвует.*

Механизм разрешения основан на достаточно сложном наборе правил, смысл которых сводится к тому, чтобы использовать функцию с наиболее подходящими аргументами и выдать сообщение, если такой не найдется.

Допустим, имеется четыре варианта функции, определяющей наибольшее значение:

```
int max(int, int);  // Возвращает наибольшее из двух целых
char* max(char*, char*);     /* Возвращает подстроку наибольшей
длины */
int max (int, char*);        /* Возвращает наибольшее из первого
                параметра и длины второго */
int max (char*, int);        /* Возвращает наибольшее из второго
                параметра и длины первого */

void f(int a, int b, char* c, char* d){
        cout << max (a, b) << max(c, d) << max(a, c) << max(c, b);
}
```

Перегрузка функций.

ПЕРЕГРУЗКА ФУНКЦИЙ.

При вызове функции **max** *компилятор выбирает соответствующий типу фактических параметров вариант функции.*

в приведенном примере будут последовательно вызваны все четыре варианта функции.

Если точного соответствия не найдено:

Перегрузка функций.

Если соответствие на одном и том же этапе может быть получено более чем одним способом, вызов считается неоднозначным и выдается сообщение об ошибке.

ПЕРЕГРУЗКА ФУНКЦИЙ.

Неоднозначность может появиться при:

- *преобразовании типа;*
- *использовании параметров-ссылок;*
- *использовании аргументов по умолчанию.*

Пример неоднозначности при преобразовании типа:

```
#include <iostream.h>
float f(float i){
     cout << "function float f(float i)" << endl;
     return i;
}
double f(double i){
     cout << "function double f(double i)" << endl;
     return i*2;
}
int main(){
     float    x = 10.09;
     double      y = 10.09;
     cout << f(x) << endl;     // Вызывается f(float)
     cout << f(y) << endl;     // Вызывается f(double)
     cout « f(10) « endl;
          /*Неоднозначность - как преобразовать 10: во float или double? */
return 0;
}
```

Для устранения этой неоднозначности требуется явное приведение типа для константы 10.

Пример неоднозначности при использовании параметров-ссылок :

если одна из перегружаемых функций объявлена как

int f(int a, int b),

а другая — как

int f (int a, int &b),

то компилятор не сможет узнать, какая из этих функций вызывается

так как нет синтаксических различий между вызовом функции, которая получает параметр по значению, и вызовом функции, которая получает параметр по ссылке.

Пример неоднозначности при использовании аргументов по умолчанию:

```
#include <iostream.h>

    int f(int a){return a;}

    int f(int a,  int b = 1){return a * b;}

int main(){

    cout << f(10,2);  // Вызывается   f(int, int)

    cout << f(10);     /* Неоднозначность - что вызывается:

        f(int, int) или f(int) ?  */

return 0;

}
```

ПРАВИЛА ОПИСАНИЯ ПЕРЕГРУЖЕННЫХ ФУНКЦИЙ.

- Перегруженные функции должны находиться *в одной области видимости,* иначе произойдет сокрытие аналогично одинаковым именам переменных во вложенных блоках.

- Перегруженные функции могут иметь *параметры по умолчанию,* при этом значения одного и того же параметра в разных функциях должны совпадать. В различных вариантах перегруженных функций может быть различное количество параметров по умолчанию.

- Функции не могут быть перегружены, если описание их параметров отличается только *модификатором* **const** или *использованием ссылки* (например, **int** и

Шаблоны функций

- В C++ есть мощное средство параметризации — **шаблоны.**

ШАБЛОНЫ ФУНКЦИЙ.

- *Многие алгоритмы не зависят от типов данных, с которыми они работают.*
  - *классический пример — сортировка.*
- *Естественно, возникает желание, параметризовать алгоритм таким образом, чтобы его можно было использовать для различных типов данных.*

Первое, что может прийти в голову — передать информацию о типе в качестве параметра

- например, одним параметром в функцию передается указатель на данные, а другим — длина элемента данных в байтах

Использование дополнительного параметра означает генерацию дополнительного кода, что снижает эффективность программы

- особенно при рекурсивных вызовах и вызовах во внутренних циклах

Кроме того, отсутствует возможность контроля типов

Шаблоны функций.

Другим решением будет написание для работы с различными типами данных нескольких перегруженных функций

Но в таком случае в программе будет несколько одинаковых по логике функций, и для каждого нового типа придется вводить новую

Шаблоны функций.

ШАБЛОНЫ ФУНКЦИЙ.

Существуют шаблоны функций и шаблоны классов.

С помощью шаблона функции можно определить алгоритм, который будет применяться к данным различных типов, а конкретный тип данных передается функции в виде параметра на этапе компиляции.

- *Компилятор автоматически генерирует правильный код, соответствующий переданному типу.*

Таким образом, создается функция, которая автоматически перегружает сама себя и при этом не содержит накладных расходов, связанных с параметризацией.

Формат простейшей функции-шаблона:

```
template <class Type> заголовок{
    /* тело функции */ }
```

Вместо слова **Type** может использоваться произвольное имя.

В общем случае шаблон функции может содержать несколько параметров, каждый из которых может быть не только типом, но и просто переменной, например:

```
template <class A, class B, int i> void f(){ ... }
```

Пример:

функция, сортирующая методом выбора массив из **n** элементов любого типа, в виде шаблона может выглядеть так:

```
template <class Type>
    void sort_vybor(Type *b, int n){
    Type a;     //буферная переменная для обмена элементов
    for (int i = 0;  i<n-l;  i++){
        int imin = I;
        for (int j = i + 1; j<n; j++)
            if (b[j] < b[imin]) imin = j;
            a = b[i]; b[i] = b[imin]; b[imin] = a;
        }
    }
```

Пример:

главная функция программы, вызывающей эту функцию-шаблон, может иметь вид:

```
#include <iostream.h>
template <class Type> void sort_vybor(Type *b, int n);
int main(){
    const int n = 20;
    int i, b[n];
    for (i = 0;  i<n; i++) cin >> b[i];
    sort_vybor(b, n);       // Сортировка целочисленного массива
    for (i = 0;  i<n; i++) cout << b[i] << ' ';
    cout << endl;
    double a[] = {0.22,  117,  -0.08, 0.21, 42.5};
    sort_vybor(a, 5); // Сортировка массива вещественных чисел
    for (i = 0;  i<5; i++) cout << a[i] << ' ';
    return 0;
}
```

ШАБЛОНЫ ФУНКЦИЙ.

Первый же вызов функции, который использует конкретный тип данных, приводит к созданию компилятором кода для соответствующей версии функции.

- *Этот процесс называется* ***инстанцированием*** *шаблона* ***(instantiation)****.*

Конкретный тип для инстанцирования либо определяется компилятором автоматически, исходя из типов параметров при вызове функции, либо задается явным образом.

- *При повторном вызове с тем же типом данных код заново не генерируется.*

На месте параметра шаблона, являющегося не типом, а переменной, должно указываться константное выражение.

Пример:

явное задание аргументов шаблона при вызове:

```
tempiate <class X, class Y, class Z> void f(Y, Z);
void g(){
   f<int, char*, double>("Vasia", 3.0);
   f<int, char*>("Vasia". 3.0);
         // Z определяется как double
   f<int>("Vasia", 3.0);
         // Y определяется как char*, а Z - как double
   // f("Vasia", 3.0);       ошибка: X определить невозможно
}
```

ШАБЛОНЫ ФУНКЦИЙ.

Чтобы применить функцию-шаблон к типу данных, определенному пользователем (структуре или классу), ***требуется перегрузить операции для этого типа данных, используемые в функции.***

Как и обычные функции, шаблоны функций могут быть ***перегружены как с помощью шаблонов, так и***

Можно предусмотреть специальную обработку отдельных параметров и типов с помощью ***специализации шаблона функции.***

Допустим, мы хотим более эффективно реализовать общий алгоритм сортировки для целых чисел.

В этом случае можно «вручную» задать вариант шаблона функции для работы с целыми числами :

```
void sort_vibor <int> (int *b, int n){

...   // Тело специализированного варианта функции

}
```

Шаблоны функций.

ШАБЛОНЫ ФУНКЦИЙ.

Сигнатура шаблона функции включает не только ее тип и типы параметров, но и фактический аргумент шаблона.

Обычная функция никогда не считается специализацией шаблона, несмотря на то, что может иметь то же имя и тип возвращаемого значения.

Функция main()

- Функция, которой передается управление после запуска программы, должна иметь имя **main**.

ФУНКЦИЯ MAIN().

Функция, **main** *может возвращать значение в вызвавшую систему и принимать параметры из внешнего окружения.*

Возвращаемое значение должно быть целого типа.

Стандарт предусматривает два формата функции:

- без параметров:

тип main(){ /* … */}

- с двумя параметрами:

тип main(int argc, char* argv[]){ /* … */}

Функция main().

ФУНКЦИЯ MAIN().

- *При запуске программы параметры разделяются* ***пробелами***.
- *Имена параметров в программе могут быть любыми, но принято использовать* **argc** *и* **argv**.
- *Первый параметр* (**argc**) *определяет количество параметров, передаваемых функции, включая имя самой программы, второй параметр* (**argv**) *является указателем на массив указателей типа* **char***.
- *Каждый элемент массива содержит указатель на отдельный параметр командной строки, хранящийся в виде С-строки, оканчивающейся* ***нуль-символом***.
- *Первый элемент массива* (**argv[0]**) *ссылается на полное имя*

ФУНКЦИЯ MAIN().

Если функция **main()** *ничего не возвращает, вызвавшая система получит значение, означающее успешное завершение.*

- ***Ненулевое значение означает аварийное завершение.***

Оператор возврата из **main()** *можно опускать.*

Пример:

```
#include <iostream.h>
void main(int argc, char* argv[]){
for (int i = 0; i<argc; i++) cout << argv[i] << '\n':
}
```

Пусть исполняемый файл программы имеет имя **main.exe** *и вызывается из командной строки:*

```
d:\BC\main.exe one two three
```

На экран будет выведено:

```
D:\BC\ MAIN.EXE
one
two
three
```

Функция main().