ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C++

1

Функции
Файлы

Никитин Михаил Евгеньевич, 2013

Инструкция return

- Инструкция return выполняет две важные операции
  - Она обеспечивает немедленное возвращение управления к инициатору вызова функции
  - Ее можно использовать для передачи значения, возвращаемого функцией

Завершение функции

- Управление от функции передается инициатору ее вызова в двух ситуациях:
  - либо при обнаружении закрывающейся фигурной скобки
  - либо при выполнении инструкции return

Возврат значений

- Каждая функция, кроме типа void, возвращает какое-нибудь значение.
- Это значение явно задается с помощью инструкции return.
- Другими словами, любую не void-функцию можно использовать в качестве операнда в выражении.
  - `x = power(y);`
  - `if (max(x, y)) > 100) cout << "больше";`
  - `switch (abs(x)) { ....`

Функции, которые возвращают указатели

- Функции могут возвращать указатели.
- Указатели возвращаются подобно значениям любых других типов данных.
- Чтобы вернуть указатель, функция должна объявить его тип в качестве типа возвращаемого значения.

```
int *f();
```

```
#include <iostream>
using namespace std;

char *find_substr(char *sub, char *str);
int main()
{
  char *substr;
  substr = find_substr("три", "один два три четыре");
  cout << "Найденная подстрока: " << substr;

  return 0;
}
```

```
char *find_substr(char *sub, char *str)
{
   int t;
   char *p, *p2, *start;
   for (t = 0; str[t]; t++) {
      p = &str[t]; // установка указателей
      start = p;
      p2 = sub;
      while (*p2 && *p2 == *p) { // проверка совпадения
         p++; p2++;
      }
 /* Если достигнут конец p2-подстроки, то эта подстрока была найдена. */
      if (!*p2) return start; // Возвращаем указатель на начало найденной
подстроки.
   }
   return 0; // подстрока не найдена
}
```

Рекурсивные функции

- Рекурсивная функция — это функция, которая вызывает сама себя
- Классическим примером рекурсии является вычисление факториала числа с помощью функции factr()

Рекурсивные функции


```
#include <iostream>
using namespace std;

int factr(int n);
int fact(int n);
int main()
{
// Использование рекурсивной версии.
   cout << "Факториал числа 4 равен " << factr(4);
   cout << '\n';

// Использование итеративной версии.
   cout << "Факториал числа 4 равен " << fact(4);
   cout << '\n';

   return 0;
}
```

Рекурсивные функции


```
// Рекурсивная версия.
int factr(int n)
{
   int answer;
   if (n == 1) return(1);
   answer = factr(n - 1)*n;
   return(answer);
}

// Итеративная версия.
int fact(int n)
{
   int t, answer;
   answer = 1;
   for (t = 1; t <= n; t++) answer = answer* (t);
   return (answer);
}
```

Ссылочные параметры

- В C++ можно сориентировать компилятор на автоматическое использование вызова по ссылке (вместо вызова по значению) для одного или нескольких параметров конкретной функции.
- Ссылочный параметр автоматически получает адрес соответствующего аргумента.

Ссылочные параметры

- При выполнении кода функции, а именно при выполнении операций над ссылочным параметром, обеспечивается его автоматическое разыменование, и поэтому программисту не нужно использовать операторы, работающие с указателями
- Ссылочный параметр объявляется с помощью символа & который должен предшествовать имени параметра в объявлении функции

Ссылочные параметры


```
#include <iostream>
using namespace std;

void f(int &i);
int main()
{
   int val = 1;
   cout << "Старое значение переменной val: " << val << '\n';
   f(val); // Передаем адрес переменной val функции f().
   cout << "Новое значение переменной val: " << val << '\n';
   return 0;
}

void f(int &i)
{
   i = 10;
}
```

Перегрузка функций

- Перегрузка функций — это механизм, который позволяет двум родственным функциям иметь одинаковые имена.

- В C++ несколько функций могут иметь одинаковые имена, но при условии, что их параметры будут различными.

Перегрузка функций

- Такую ситуацию называют перегрузкой функций (function overloading), а функции, которые в ней задействованы, — перегруженными (overloaded)
- Перегрузка функций — один из способов реализации полиморфизма в C++
- В языках программирования и теории типов полиморфизмом называется способность функции обрабатывать данные разных типов

Перегрузка функций


```
#include <iostream>
using namespace std;

void f(int i); // один целочисленный параметр
void f(int i, int j); // два целочисленных параметра
void f(double k); // один параметр типа double

int main()
{
   f(10); // вызов функции f(int)
   f(10, 20); // вызов функции f (int, int)
   f(12.23); // вызов функции f(double)
   return 0;
}
```

Перегрузка функций


```
void f(int i)
{
   cout << "В функции f(int), i равно " << i << '\n';
}
void f(int i, int j)
{
   cout << "В функции f(int, int), i равно " << i;
   cout << ", j равно " << j << '\n';
}
void f(double k)
{
   cout << "В функции f(double), k равно " << k << ' \n';
}
```

Перегрузка функций

- Для определения того, какую версию перегруженной функции вызвать, компилятор использует тип и/или количество аргументов.

- Перегруженные функции должны отличаться типами и/или числом параметров

Перегрузка функций

- Рассмотрим три функции из стандартной библиотеки: abs(), labs() и fabs(). Они были впервые определены в языке C.
- Функция abs() возвращает абсолютное значение (модуль) целого числа
- Функция labs() возвращает модуль длинного целочисленного значения (типа long)
- Функция fabs() — модуль значения с плавающей точкой (типа double)

Перегрузка функций


```
include <iostream>
using namespace std;

// Функция myabs() перегружается тремя способами.
int myabs(int i);
double myabs(double d);
long myabs(long l);

int main()
{
   cout << myabs(-10) << "\n";
   cout << myabs(-11.0) << "\n";
   cout << myabs(-9L) << "\n";
   return 0;
}
```

```
int myabs(int i)
{
   if (i<0) return -i;
   else return i;
}

double myabs(double d)
{
   if (d<0.0) return -d;
   else return d;
}

long myabs(long l)
{
   if (1<0) return -1;
   else return 1;
}
```

Аргументы по умолчанию

- В C++ мы можем придать параметру некоторое значение, которое будет автоматически использовано, если при вызове функции не задается аргумент, соответствующий этому параметру.
- Задание аргументов, передаваемых функции по умолчанию, синтаксически аналогично инициализации переменных.

Аргументы по умолчанию


```
void myfunc(double num = 0.0, char ch = 'X')
{
    .
    .
    .
}

myfunc(198.234, 'A'); // Передаем явно заданные значения.

myfunc(10.1); // Передаем для параметра num значение 10.1, а для параметра ch позволяем применить  аргумент, задаваемый по умолчанию ('X').

myfunc(); // Для обоих параметров num и ch позволяем применить аргументы, задаваемые по умолчанию.
```

Аргументы по умолчанию

- При создании функций, имеющих значения аргументов, передаваемых по умолчанию, необходимо помнить две вещи
  - Эти значения по умолчанию должны быть заданы только однажды, причем при первом объявлении функции в файле.
  - Все параметры, которые принимают значения по умолчанию, должны быть расположены справа от остальных.

Аргументы по умолчанию


```
#include <iostream>
using namespace std;

void clrscr(int size = 25);
int main()
{
   int i;
   clrscr(); // Очищаем 25 строк.
   clrscr(10); // Очищаем 10 строк.
   return 0;
}
void clrscr(int size = 25)
{
   for (; size; size--) cout << '\n';
}
```

Программа скомпилируется?

Выполнится правильно?

Никитин Михаил Евгеньевич, 2013

Аргументы по умолчанию


```
#include <iostream>
using namespace std;

void clrscr(int size = 25);
int main()
{
   int i;
   clrscr(); // Очищаем 25 строк.
   clrscr(10); // Очищаем 10 строк.
   return 0;
}
void clrscr(int size)
{
   for (; size; size--) cout << '\n';
}
```

Программа скомпилируется?

Выполнится правильно?

Никитин Михаил Евгеньевич, 2013

Аргументы по умолчанию


```
#include <iostream>
using namespace std;

void clrscr(int b, int size = 25);
int main()
{
   int i;
   clrscr(); // Очищаем 25 строк.
   clrscr(10); // Очищаем 10 строк.
   return 0;
}
void clrscr(int b, int size )
{
   for (; size; size--) cout << '\n';
}
```

Программа скомпилируется?

Выполнится правильно?

Аргументы по умолчанию


```
#include <iostream>
using namespace std;

void clrscr(int size = 25, int b);
int main()
{
   int i;
   clrscr(); // Очищаем 25 строк.
   clrscr(10); // Очищаем 10 строк.
   return 0;
}
void clrscr(int size, int b )
{
   for (; size; size--) cout << '\n';
}
```

Программа скомпилируется?

Выполнится правильно?

Никитин Михаил Евгеньевич, 2013

Файлы

- Большинство компьютерных программ работают с файлами, и поэтому возникает необходимость создавать, удалять, записывать читать, открывать файлы.
- Файл – именованный набор байтов, который может быть сохранен на некотором накопителе.

Файлы

- Для работы с файлами необходимо подключить заголовочный файл <fstream>
- В <fstream> определены несколько классов и подключены заголовочные файлы <ifstream> - файловый ввод и <ofstream> - файловый вывод
- Файловый ввод/вывод аналогичен стандартному вводу/выводу, единственное отличие – это то, что ввод/вывод выполнятся не на экран, а в файл

Файлы

- Для записи в файл создать объект класса ofstream
- Если нужно считать данные из файла, то создается объект класса ifstream.

Файлы

- Например, необходимо создать текстовый файл и записать в него строку Работа с файлами в C++. Для этого необходимо проделать следующие шаги:
  - создать объект класса ofstream;
  - связать объект класса с файлом, в который будет производиться запись;
  - записать строку в файл;
  - закрыть файл.

Файлы


```
#include <fstream>
using namespace std;

int main(int argc, char* argv[])
{
ofstream fout("cppstudio.txt");
fout << "Работа с файлами в C++";
fout.close();
system("pause");
return 0;
}
```

Файлы


```
#include <fstream>
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
    setlocale(LC_ALL, "rus"); // корректное отображение Кириллицы
    char buff[50]; // буфер промежуточного хранения считываемого из файла текста
    ifstream fin("cppstudio.txt"); // открыли файл для чтения
    fin >> buff; // считали первое слово из файла
    cout << buff << endl; // напечатали это слово
    fin.getline(buff, 50); // считали строку из файла
    fin.close(); // закрываем файл
    cout << buff << endl; // напечатали эту строку
    system("pause");
    return 0;
}
```

Файлы

- В C++ предусмотрена такая функция -is_open(), которая возвращает целые значения: 1 — если файл был успешно открыт, 0 — если файл открыт не был.

```
ifstream fin("cppstudio.doc");
if (!fin.is_open())
{
    cout << "Файл не может быть открыт!\n";
    return 1;
}
```

Режимы открытия файлов

- Режимы открытия файлов устанавливают характер использования файлов.

- Для установки режима в классе ios_base предусмотрены константы, которые определяют режим открытия файлов

Режимы открытия файлов

Константа	Описание
ios_base::in	открыть файл для чтения
ios_base::out	открыть файл для записи
ios_base::ate	при открытии переместить указатель в конец файла
ios_base::app	открыть файл для записи в конец файла
ios_base::trunc	удалить содержимое файла, если он существует
ios_base::binary	открытие файла в двоичном режиме

```
ofstream fout("cppstudio.txt", ios_base::app);
```

```
fout.open("cppstudio.txt", ios_base::app);
```

```
fout.open("cppstudio.txt", ios_base::out |
ios_base::trunc);
```

Структуры в C++

- Структура — это , некое объединение различных переменных (даже с разными типами данных), которому можно присвоить имя.
- Например можно объединить данные об объекте Дом: город (в котором дом находится), улица, количество квартир, интернет(проведен или нет) и т.д. в одной структуре.

Структуры в C++


```
#include <iostream>
using namespace std;

struct building      //Создаем структуру!
{
   char *owner;         //здесь будет храниться имя владельца
   char *city;          //название города
   int amountRooms;     //количество комнат
   float price;         //цена
};

int main()
{
  setlocale(LC_ALL, "rus");

  building apartment1;    //это объект структуры с типом данных, именем структуры, building
```


Никитин Михаил Евгеньевич, 2013

Структуры в C++


```
    apartment1.owner = "Денис"; //заполняем данные о владельце и т.д.
    apartment1.city = "Симферополь";
    apartment1.amountRooms = 5;
    apartment1.price = 150000;

    cout << "Владелец квартиры: " << apartment1.owner << endl;
    cout << "Квартира находится в городе: " << apartment1.city << endl;
    cout << "Количество комнат: " << apartment1.amountRooms << endl;
    cout << "Стоимость: " << apartment1.price << " $" << endl;

    return 0;
}
```