

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

C++

ВОПРОСЫ

- ⊙ Что такое объектно-ориентированное программирование?
- ⊙ Какие еще методологии (парадигмы) программирования существуют?
- ⊙ Зачем нужны парадигмы программирования? Какие проблемы они пытаются решить?
- ⊙ Что такое объект? Чем он определяется?
- ⊙ Что такое класс?
- ⊙ В чем состоит проблема идентичности объекта?
- ⊙ Назовите «3 кита ООП». Что они означают?

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ ЯЗЫКИ И C++

ОО-ЯЗЫКИ ПРОГРАММИРОВАНИЯ

1. C#
2. C++
3. Java
4. Delphi
5. Eiffel
6. Simula
7. D
8. lo
9. Objective-C
10. Object Pascal
11. VB.NET
12. Visual DataFlex
13. Perl
14. PowerBuilder
15. Python
16. Scala
17. JavaScript
18. JScript .NET
19. Ruby
20. Smalltalk
21. Ada
22. Xbase++
23. X++
24. Vala
25. PHP

РЕЙТИНГ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Position Feb 2012	Position Feb 2011	Delta in Position	Programming Language	Ratings Feb 2012	Delta Feb 2011	Status
1	1	=	Java	17.050%	-1.43%	A
2	2	=	C	16.523%	+1.54%	A
3	6	↑↑↑	C#	8.653%	+1.84%	A
4	3	↓	C++	7.853%	-0.33%	A
5	8	↑↑↑	Objective-C	7.062%	+4.49%	A
6	5	↓	PHP	5.641%	-1.33%	A
7	7	=	(Visual) Basic	4.315%	-0.61%	A
8	4	↓↓↓↓	Python	3.148%	-3.89%	A
9	10	↑	Perl	2.931%	+1.02%	A
10	9	↓	JavaScript	2.465%	-0.09%	A
11	13	↑↑	Delphi/Object Pascal	1.964%	+0.90%	A
12	11	↓	Ruby	1.558%	-0.06%	A
13	14	↑	Lisp	0.905%	-0.05%	A
14	26	↑↑↑↑↑↑↑↑	Transact-SQL	0.846%	+0.29%	A
15	17	↑↑	Pascal	0.813%	+0.08%	A
16	22	↑↑↑↑↑	Visual Basic .NET	0.796%	+0.21%	A-
17	32	↑↑↑↑↑↑↑↑	PL/SQL	0.792%	+0.38%	A
18	24	↑↑↑↑↑	Logo	0.677%	+0.10%	B
19	16	↓↓↓	Ada	0.632%	-0.17%	B
20	25	↑↑↑↑	R	0.623%	+0.06%	B

- ◎ В 1980 году Бьерн Страуструп в AT&T Bell Labs стал разрабатывать расширение языка C под условным названием C++
- ◎ Первый коммерческий транслятор: 1983 год
- ◎ Главное нововведение C++ - механизм классов, дающий возможность определять и использовать новые типы данных.

ОСНОВНЫЕ ОТЛИЧИЯ C++ ОТ C

1. В C++ появились классы и объекты.
2. В C++ появились две новые операции: `new` и `delete` – замещение функций `malloc` и `free`:
 1. `new` = `malloc` + конструктор,
 2. `delete` = `free` + деструктор.
3. Появился механизм обработки исключений:
`try` + `catch`

Прежде чем переходить к ОО-особенностям языка C++ изучим синтаксические отличия C++ от C

ВСТРАИВАЕМЫЕ ФУНКЦИИ

В C++ появились ключевое слово `inline`, означающее рекомендацию компилятору сделать функцию *встраиваемой*, то есть вместо генерации вызывающего её кода подставлять непосредственно её тело.

Но тело встроенной функции, подобно макросу, должно быть написано там же, где её заголовок.

Таким образом, функция

```
inline double Sqr(double x) {return x*x;}
```

будет вычисляться так же быстро, как $x*x$.

АРГУМЕНТЫ ПО УМОЛЧАНИЮ

- ⊙ В C++ Один или больше последних аргументов функции могут задаваться по умолчанию:

```
void f(int x, int y=5, int z=10);
```

```
// void g(int x=5, int y); /* Неправильно! По умолчанию задаются только  
последние аргументы */
```

```
f(1); // будет вызвано f(1, 5, 10)
```

```
f(1, 2); // будет вызвано f(1, 2, 10)
```

```
f(1, 2, 3); // будет вызвано f(1, 2, 3)
```

С И С++ - ПЕРЕДАЧА ПАРАМЕТРОВ

- ⊙ В С аргументы всегда передаются по значению (происходит копирование значения аргумента в локальную функцию)
- ⊙ Можно передавать указатель:

```
void foo (int *x)
{
    *x = 17;
}
int main ()
{
    int z = 5;
    foo (&z);
    /* остальные варианты больше не
имеют смысла */
    /* z теперь равно 17 */
}
```

С И С++ - ПЕРЕДАЧА ПАРАМЕТРОВ

- ⊙ Передача параметров по указателю **ОЧЕНЬ ОПАСНО!**
- ⊙ Можно забыть * в теле функции
- ⊙ Или забыть & при вызове (сколько раз такое было при вызове функции “scanf”?!)
- ⊙ В Си++ можно *передавать параметры по ссылкам:*

```
void foo (int& x)
{
    x = 17;
}
int main ()
{
    int z = 5;
    foo (z);
    /* z теперь равно 17 */
}
```

ССЫЛКИ В C++

- ◎ Ссылку в C++ можно понимать или как альтернативное имя объекта, или как безопасный вариант указателей. Ссылки имеют три особенности, отличающие их от указателей:
 - ◎ При объявлении ссылка **обязательно инициализируется** ссылкой на уже существующий объект данного типа.
 - ◎ Ссылка **пожизненно** указывает на один и тот же адрес.
 - ◎ При обращении к ссылке **операция * производится автоматически.**

ПРОСТРАНСТВА ИМЕН

- В Си++ структуры («`struct`»), объединения («`union`») и классы («`class`») стали создавать полноправные типы данных. Кроме того, каждая структура и объединение (как и, разумеется, каждый класс) получили своё полноправное пространство имён.

```
struct Foo {
    typedef unsigned char byte;
    byte data [16];
};

int main () {
    Foo::byte x;
    x = 17;
}
```

- Оператор «`::`» - оператор разрешения области видимости, позволяет определить по какому пути надо идти, чтобы найти необходимое имя.

ПРОСТРАНСТВА ИМЕН

- Могут определяться не только в рамках классов, но и ключевым словом «`namespace`» :

```
mytypes.h
namespace MyFavouriteTypes {
    typedef unsigned char byte;
}
```

- С помощью директивы «`using namespace`» вы говорите, что «дальше я буду ссылаться на такое-то пространство имён без указания его имени». Эту директиву можно написать и внутри функции:

```
#include "mytypes.h"
int main () {
    using namespace MyFavouriteTypes;
    byte data [16];
    ...
}
```

БИБЛИОТЕКА ВВОДА-ВЫВОДА IOSTREAM

- ◎ Никаких printf и scanf!
- ◎ Каноничный Hello World на C++

```
#include <iostream>

int main()
{
    std::cout << "Hello,world!\n";
    return 0;
}
```

БИБЛИОТЕКА IOSTREAM

- ◎ Пример ввода-вывода посредством iostream:

Используем std

Вывод

Ввод

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

int main()
{
    int x;
    cout << "enter choice:";
    cin >> x;
    while (x < 1 || x > 4)
    {
        cout << "Invalid choice, try again:";
        cin >> x;
        // not a numeric character, probably
        // clear and pull off the non-numeric character
        if (cin.fail())
        {
            cin.clear();
            char c;
            cin >> c;
        }
    }
}
```

Обработка неправильного
ввода

КЛАССЫ В C++

ОПИСАНИЕ КЛАССОВ В C++

```
class <имя> {  
    [ private: ]  
        [ <описание скрытых элементов> ]  
    [ protected: ]  
        [ <описание защищенных элементов> ]  
    public:  
        [ <описание доступных элементов> ]  
}; // Описание заканчивается точкой с запятой
```

СПЕЦИФИКАТОРЫ ВИДИМОСТИ PRIVATE И PUBLIC

- ◎ Спецификаторы доступа `private` и `public` управляют видимостью элементов класса.
- ◎ Элементы, описанные после служебного слова `private`, видимы только внутри класса. Этот вид доступа принят в классе по умолчанию.
- ◎ Интерфейс класса описывается после спецификатора `public`.
- ◎ Действие любого спецификатора распространяется до следующего спецификатора или до конца класса. Можно задавать несколько секций `private` и `public`, порядок их следования значения не имеет.
- ◎ Ключевое слово `protected` будем обсуждать более подробно при обсуждении механизма наследования

ПРИМЕР КЛАССА

Конструктор (метод с именем класса,
ничего не возвращает)

```
class monster
{
    int health, ammo;

public:
    monster(int he = 100, int am = 10)
        { health = he; ammo = am;}
    void draw(int x, int y, int scale, int position);
    int get_health(){return health;}
    int get_ammo(){return ammo;}
};
```

Встроенные
(inline) методы

РЕАЛИЗАЦИЯ МЕТОДА КЛАССА

- ⊙ Если метод определен в классе, но его реализация относительно сложна, реализацию выносят в другое место программы, чтобы не загромождать описание класса

```
void monster::draw(int x, int y, int scale,  
int position)  
{ /* тело метода */}
```

ИСПОЛЬЗОВАНИЕ КЛАССОВ В ПРОГРАММЕ

- © Если ранее мы определили какой-либо класс, мы можем описать его экземпляры (объекты) в нашей программе следующим образом:

```
monster Vasia;// Объект класса monster с параметрами по умолчанию
monster Super(200, 300);// Объект с явной инициализацией
monster stado[100];// Массив объектов с параметрами по умолчанию
/* Динамический объект (второй параметр задается по умолчанию) */
monster *beavis = new monster (10);
monster &butthead = Vasia;// Ссылка на объект
```

ДОСТУП К ПОЛЯМ И МЕТОДАМ

- ⊙ *Доступ к открытым (public) элементам объекта аналогичен доступу к полям структуры:*

```
объект.поле  
указатель -> поле  
(*указатель).поле  
объект.метод( параметры )  
указатель -> метод( параметры )  
(*указатель).метод( параметры )
```

- ◎ С++ - ОО-язык программирования.
- ◎ Отличия С++ от С:
 - ◎ Методы описания классов и объектов
 - ◎ Inline – функции;
 - ◎ Методы обработки исключительных ситуаций
 - ◎ Работа с пространствами имен
 - ◎ Новые методы ввода-вывода
 - ◎ И многое-многое другое

- ◎ Описание классов в C++
 - ◎ Ключевое слово «`class`»
 - ◎ Модификаторы доступа `public`, `protected`, `private`
 - ◎ Конструктор – специальный метод класса, который вызывается при создании объекта.
 - ◎ Есть возможность описывать методы класса в отдельном файле, посредством операции указания области видимости «`::`»
 - ◎ Можно создавать объекты определенного класса (в том числе, и в динамической памяти с использованием `new`).
 - ◎ Доступ к методам и полям класса осуществляется посредством операций «`.`» и «`->`»