

С++

Лекция 1

Введение в классы



- ✓ Типы данных
- ✓ Указатели и ссылки
- ✓ Строки и I/O stream
- ✓ Абстрактные типы данных
- ✓ Классы

Типы данных

- **bool**
- **int**
- **float**
- **double**
- **char**

Signed
Unsigned

**Что такое
переполнение?**

Приведение типов (casting)

```
double x = 3.7;  
int i = (int) (x);  
i = (int) (x + 0.5);  
cout << i << endl;
```

Переменная != объект в памяти

	Переменная	Объект
Что?	имя	данные
Когда?	время компиляции	runtime(время исполнения)
Где?	исходных код	память
Ограничения?	область видимости	время жизни

Указатели

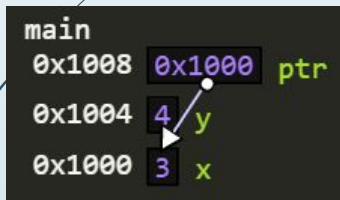
- Каждый объект находится в памяти по определенному адресу.
- Получить адрес объекта можно с помощью оператора &.

```
int main() {  
    int x = 3;  
    double y = 5.5;  
    cout << &x << endl;  
    cout << &y << endl;  
}
```

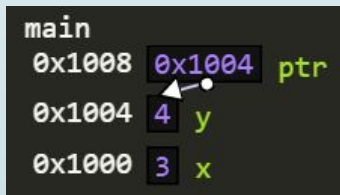
Указатели

- Указатель - переменная, которая содержит адрес другой переменной
- Формат объявления указателя:
 - **тип *имя_переменной;**

```
main
0x1008 0x1000 ptr
0x1004 4 y
0x1000 3 x
```



```
main
0x1008 0x1004 ptr
0x1004 4 y
0x1000 3 x
```



```
int main() {
    int x = 3;
    int y = 4;
    int *ptr = &x;
    cout << ptr << endl;
    ptr = &y;
    cout << ptr << endl;
}
```

Указатели

Базовый тип указателя определяет тип данных, на которые он будет ссылаться.

```
int *ip; // указатель на int  
double *dp; // указатель на double
```

Не существует реального средства, которое может помешать указателю ссылаться на "бог-знает-что"

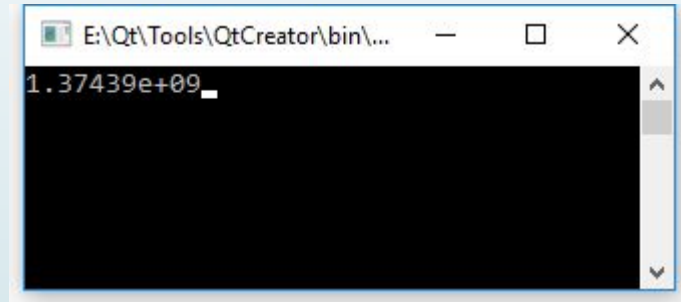
Указатели

```
int main()
{
    double x, y;
    int *p;

    x = 123.23;
    p = (int *) &x;

    y = *p;
    cout << y;

    return 0;
}
```



ССЫЛКИ

Ссылка она как указатель, только ссылка.

- Ссылка должна указывать на некоторый объект
- Ссылка должна быть инициализирована при объявлении

```
int j, k;  
int &i = j;  
j = 10;  
  
cout << j << " " << i << endl;  
  
k = 121;  
i = k;  
  
cout << j;
```

ССЫЛКИ

- Адрес который содержит ссылка **нельзя** изменить
- **Нельзя** сослаться на ссылку
- **Нельзя** создать массив ссылок
- **Нельзя** создать указатель на ссылку
- Ссылки **нельзя** использовать для битовых полей структур

So Many * and &



- Используется для определения типа данных...
 - * создание указателя
 - & создание ссылки
- Используется в качестве оператора в выражениях ...
 - * взять объект по адресу
 - & взять адрес объекта

```
int *ptr;
```

```
int &ref;
```

```
cout << *ptr << endl;
```

```
cout << &x << endl;
```

Ссылки vs. Указатели

Ссылки	Указатели
Псевдоним (ещё одно имя) для объекта	Хранит адрес объекта

```
int main() {  
    int x = 3;  
    int &y = x;  
    int *z = &x;  
}
```

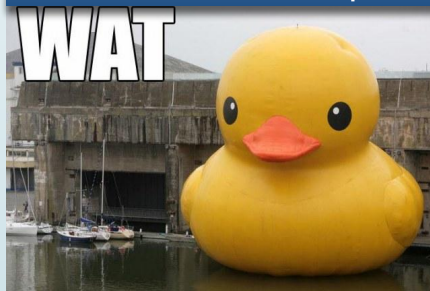
- В указатель можно положить адрес другого объекта.
- Положить в ссылку другой объект нельзя!

Передача по значению

При передачи параметров по значению передают **ТОЛЬКО ЗНАЧЕНИЯ** объектов, но не сами объекты!

```
void swap(int x, int y) {  
    int temp = x;  
    x = y;  
    y = temp;  
}  
  
int main() {  
    int a = 3;  
    int b = 7;  
    cout << a << ", " << b;  
    swap(a, b);  
    cout << a << ", " << b;  
}
```

Ничего не происходит



Передача по ссылке

При передаче по ссылке передается **сам объект**, просто с другим именем (псевдонимом).

```
void swap(int &x, int &y) {  
    int temp = x;  
    x = y;  
    y = temp;  
}
```

```
int main() {  
    int a = 3;  
    int b = 7;  
    cout << a << ", " << b;  
    swap(a, b);  
    cout << a << ", " << b;  
}
```

Где заканчивается массив?

- Что случится если указатель выйдет за пределы массива?
 - Неопределенное поведение!
 - Чтение/запись случайного участка памяти.
 - Программа будет крашиться или не будет или будет только иногда.
- Как следить за границами массива?
 - Хранить длину массива и следить за ней
 - **Положить сигнальное значение в конец массива**

C-Style Строки

- В C строки представлены как массив символов.

```
char str1[6] = { 'h', 'e', 'l', 'l', 'o', '\0' };
```

```
char str2[6] = "hello";
```

Компилятор автоматически добавит '\0'
в конец массива.

- Нулевой символ обозначает конец строки.
 - '\0'
 - ASCII код 0
- Создание указателя на строку.

```
char *strPtr = str1;
```

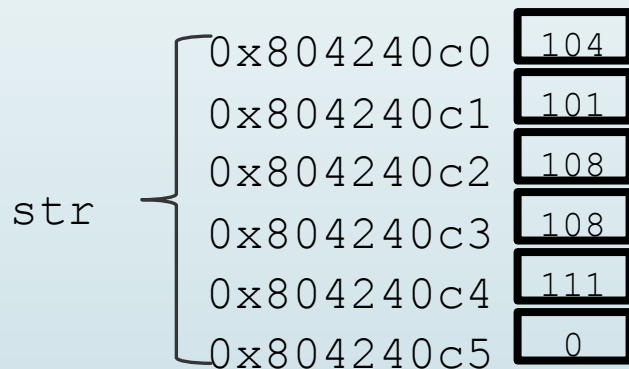
C-style строка - массив

- char СИМВОЛ это просто число

ASCII Код

СИМВОЛ	Число
'\0'	0
...	
'e'	101
'f'	102
'g'	103
'h'	104
...	

```
char str[6] = "hello"
```



Компилятор
АВТОМАТИЧЕСКИ ДОБАВИТ
0 в КОНЕЦ массива.



C-Style ловушки

Проверяет на равенство адреса

Не скомпилируется.
Не совпадают типы.

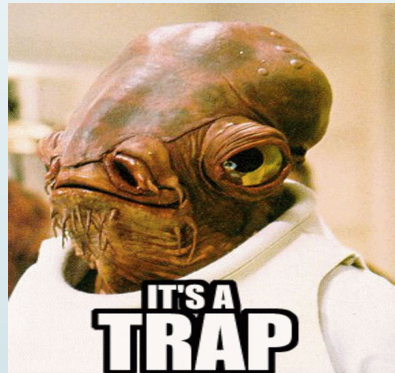
Указателю
присваивается
адрес другой
строки.

```
char str1[6] = "hello";  
char str2[6] = "hello";  
char str3[6] = "apple";  
char *ptr = str1;
```

```
// Test for equality?  
str1 == str2;
```

```
// Copy strings?  
str1 = str3;
```

```
// Copy through pointer?  
ptr = str3;
```



Проход C-Style строки

```
char str[6] = "hello";  
cout << strlen(str) << endl; // Prints 5
```

- Цикл до конца строки.

Указатель на
начало строки

```
int strlen(const char *str) {  
    const char *ptr = str;  
    while (*ptr != '\0') {  
        ++ptr;  
    }  
    return ptr - str;  
}
```

Продолжаем пока не
встретим символ
конца строки

Инкремент указателя

Возвращает количество
сделанных итераций
(Без учета '\0'.)

C-Style Strings and cout

- Как **нельзя** вывести массив:

```
int array[3] = { 1, 2, 3 };  
cout << array << endl;
```

Приводится к `int*`.
Выведется адрес,
а не 1,2,3.

- Но можно вывести C-style строку.

```
char str[6] = "hello";  
cout << str << endl;
```

Turns into a `char*`.
Prints out "hello".

- `cout` всегда воспринимает `char*` как C-style строки
 - Выводит символы до тех пор, пока не встретит символ конца строки.

C++ строки

	C-Style Strings	C++ Strings
Library Header	<code><cstring></code>	<code><string></code>
Объявление	<code>char cstr[];</code> <code>char *cstr;</code>	<code>string str;</code>
Длина	<code>strlen(cstr)</code>	<code>str.length()</code>
Копирование	<code>strcpy(cstr1, cstr2)</code>	<code>str1 = str2</code>
Обращение к элементу	<code>cstr[i]</code>	<code>str[i]</code>
Конкатенация	<code>strcat(cstr1, cstr2)</code>	<code>str1 += str2</code>
Сравнение	<code>strcmp(cstr1, cstr2)</code>	<code>str1 == str2</code>

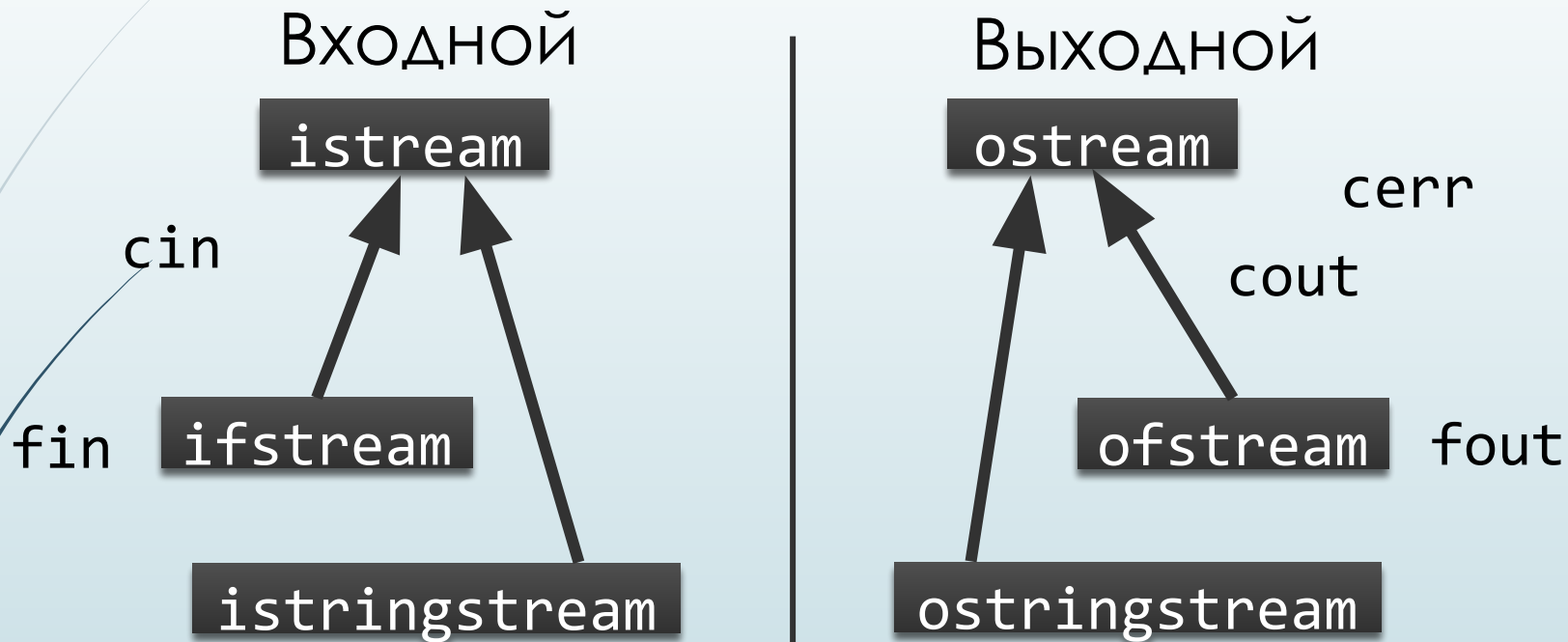
string to C-style string: `char *cstr = str.c_str();`

C-style string to string: `string str = string(cstr);`

Сравнение строк

- C++
 - С помощью операторов `==`, `!=`, `<`, `<=`, `>`, `>=`
- C-style
 - Нельзя использовать операторы, они сравнивают адреса.
 - Функция `strcmp`.
 - `strcmp(A, B)` возвращает:
 - отрицательное если A меньше B
 - 0 если A равно B
 - положительно если A больше B

ВИДЫ ПОТОКОВ



ВЫХОДНОЙ ПОТОК

- Для записи в выходной поток используется оператор <<.
- Вывод определяется типом данных.

```
char c;  
cout << c;
```

Выведет один символ

```
double d;  
cout << d;
```

Выведет число с
плавающей точкой

```
string s;  
cout << s;
```

Выведет строку

```
char *cstr;  
cout << cstr;
```

Выведет c-style
строку, символы до
тех пор пока не
встретится '/0'

Stream Input

- Для чтения входного потока используется оператор `>>`.
- Поведение определяется типом данных

```
char c;  
cin >> c;
```

```
int i;  
cin >> i;
```

```
string s;  
cin >> s;
```

```
double d;  
cin >> d;
```

On to classes!

struct

- Объединяет различные типы данных
- **C style**
- **Содержит только данные**
- **По умолчанию значения не определены**
- **Все данные доступны**

class

- Объединяет различные типы данных
- **C++ style**
- **Содержит данные и функции**
- **Можно инициализировать с помощью конструкторов**
- **Можно ограничивать доступ к данным**

Введение в классы

- Класс состоит из **полей** и **методов**.

```
class Triangle {  
    double a;  
    double b;  
    double c;  
  
    Triangle(double a_in, double b_in, double c_in) { ... }  
  
    double perimeter() const { ... }  
    void scale(double s) { ... }  
};
```

```
int main() {  
    Triangle t1(3, 4, 5);  
    t1.scale(2);  
    cout << t1.perimeter();  
}
```

Методы класса

C Style(struct)

```
void Triangle_scale(  
    Triangle *tri, double s) {  
    tri->a *= s;  
    tri->b *= s;  
    tri->c *= s;  
}
```

tri



```
int main() {  
    Triangle t1 = {3, 4, 5};  
    Triangle_scale(&t1, 2);  
}
```

Нужно передать
указатель на t1

C++ Style (class)

```
class Triangle {  
    double a;  
    double b;  
    double c;  
  
    void scale(double s) {  
        this->a *= s;  
        this->b *= s;  
        this->c *= s;  
    }
```

this



```
int main() {  
    Triangle t1(3, 4, 5);  
    t1.scale(2);  
}
```

Компилятор все
сделал сам

struct vs. class

- **Единственное** различие между структурами и классами в С++ это уровень доступа по умолчанию
 - struct – public по умолчанию
 - class – private по умолчанию

Тем не менее классы и структуры используются в С++ для разных целей

```
struct Triangle {  
    double a;  
    double b;  
    double c;  
    ...  
};
```

a, b, c
public

```
class Triangle {  
    double a;  
    double b;  
    double c;  
    ...  
};
```

a, b, c
private

Доступ без this

- Можете просто не писать this-> компилятор сделает это сам :)

```
class Triangle {  
private:  
    double a;  
    double b;  
    double c;  
  
public:  
    void scale(double s) {  
        this->a *= s;  
        this->b *= s;  
        this->c *= s;  
    }  
};
```

```
class Triangle {  
private:  
    double a;  
    double b;  
    double c;  
  
public:  
    void scale(double s) {  
        a *= s;  
        b *= s;  
        c *= s;  
    }  
};
```

Модификаторы доступа

- Все поля и методы класса имеют модификаторы доступа
 - **Public**: Может быть вызван из любого места в программе.
 - **Private**: Может использоваться только внутри класса.

Модификаторы доступа

```
class Triangle {  
private:  
    double a;  
    double b;  
    double c;  
  
public:  
    void scale(double s) {  
        this->a *= s;  
        this->b *= s;  
        this->c *= s;  
    }  
};
```

Поля - private.
Методы - public.

Обращение к a,b,c
внутри класса
допустимо

```
int main() {  
    Triangle t1(3, 4, 5);  
    t1.scale(2);  
    cout << t1.scale();  
  
    // Die triangle! DIE!  
    t1.a = -1;  
}
```

Все норм,
метод
`scale()`
public

Ошибочка, а
приватное поле
класса

Конструктор

- имя конструктора совпадает с именем класса
- не имеет типа возвращаемого значения
- создает экземпляр класса
- инициализирует переменные объекты класса



Имя
совпадает
с именем
класса

```
class Triangle {  
private:  
    double a;  
    double b;  
    double c;
```

С помощью
передаваемых
параметров
инициализируются поля
класса

```
public:
```

```
    Triangle(double a_in, double b_in, double c_in) {  
        a = a_in;  
        b = b_in;  
        c = c_in;  
    }
```

Но... иногда это не совсем то что
нужно.

В этих строках происходит
присвоение, а не инициализация

```
    double perimeter() const { ... }  
    void scale(double s) { ... }  
};
```

Явный Default Constructor

- Если явно не задать конструктор по умолчанию, компилятор создает его сам (неявный конструктор).
- Неявный конструктор пустой
 - Если определен хоть один явный конструктор, то неявный не будет создан
 - Примерно так можно представить себе как выглядит неявный конструктор:

```
class Triangle {  
    Triangle() {  
        // nothing here  
    }  
};
```

Инициализация полей по умолчанию

- Если поля класса явно не инициализируются они инициализируются значениями по умолчанию

```
struct Person {  
    int age;  
    string name;  
    bool isNinja;  
    // implicit default constructor  
    // Person() {}  
};  
  
int main() {  
    Person alex;  
    Person jon = { 25, "jon", true };  
}
```

мусор (не
обязательно
0)

String по умолчанию
инициализируется как
пустая строка

The Stack	
main	
alex Person	
0x1000 age	0
0x1004 name	""
0x1008 isNinja	false
jon Person	
0x1009 age	25
0x1013 name	"jon"
0x1017 isNinja	true

Get и Set методы

```
class Triangle {  
private:  
    double a;  
    double b;  
    double c;  
  
public:  
    double get_a() const {  
        return a;  
    }  
  
    void set_a(double a_in) {  
        a = a_in;  
        check_invariants();  
    }  
}
```

Каждый раз при
изменении значения a
вызывается функция
`check_invariants()`

Good Abstraction Design

- Инкапсуляция
 - С++ классы объединяют в себе данные методы обработки этих данных.
 - Инкапсуляция позволяет ограничить доступ одних компонентов программы к другим.
- Разделение **интерфейсов** от **реализации**.
 - Работайте только с интерфейсами и скрывайте реализацию
 - Избегайте зависимостей в реализации.

Спасибо!

Вопросы?

Контакты:

t.me/itmo_cpp

claorisel@gmail.com