

Введение в C++

Ссылки

Было в C:

```
int x=1, z=0;
```

```
int *p = &x;
```

```
void f(int y, int *q)
```

```
{
```

```
    y=2;
```

```
    *q=3;
```

```
}
```

```
f(z, p) // z не изменится, x изменится
```

Стало в C++

```
int x=1, z=0;
```

```
int *p = &x;
```

```
void f(int ссылка&y, int *q)
```

```
{
```

```
    y=2;
```

```
    *q=3;
```

```
}
```

```
f(нельзя написать const или depenz, p) // z=2 ; x=3
```

& в C++ это:

- U

- взятие адреса

- при создании переменных это ссылка

int &r = 2; (создаем ссылку и обязательно инициализируем)

const где ссылки тоже работает как где указатели (если не собираемся менять)

```
void f(const int &y, int *g)
```

```
{  
    *g = 3;
```

```
}  
  
f(1, &x) // x = 3
```

потому что создается объект и берется адрес

```
int &f(const int &y, int *g)
```

```
{  
    int k = y + *g;
```

```
    return k;
```

```
}
```

• int f(...) → нашел статус не важна (вернет my cop)

```
int f(const int &y, int *g)
```

```
{  
    int k = y + *g;
```

```
    return *g;
```

```
}
```

```
f(1, &x) = 3 // *p = 3 (перепределен x)
```

если передаем сложную структуру куда-то то она копируется, а можно передать ссылки

Знаение по умножению

```
int f(int x, int y = 3, float z = 4.5f);
```

можно передавать свои аргументы, но если не передать, то получите реализацию

$f(1, 2) \rightarrow f(1, 2, 4.5f)$

• можно использовать `for` `sizeof`

13) Т.к. в C++ все значения передаются последовательно, то з.п.у. передаются в конце

Перезузка

`double sqrt(double);`

`float sqrtf(float);`

`long double sqrtl(long double);`

в C++ можно:

`double s(double x) { return x+1; }`

//

#

`float s(float x) { return x+2; }`

//

#

`long double s(long double x) { return x+3; }`

как linker выдает ошибку `s`? → компилятор добавляет `son.unity` в `link`

функции и linker их различает. (использует `@`, `?` ... потому мы не пересекаем)

функция без имени (не меняем `link` и невозможна перезагрузка)

1) `extern "C" int f(int x);`

2) `extern "C"`
`{`
`int f(int x)`
`:`
`}`

13) `extern "C"` — некорректная конструкция в C

факт: Компилятор `extern "C"` в C++, а в C не надо

```
# ifdef __cplusplus
```

```
extern "C"  
{
```

```
# endif
```

```
+  
# ifdef __cplusplus
```

```
}
```

```
# endif
```

float a = f(2); → *ошибка компиляции (нет такого значения)*

float x = ^{int}3/2; // x = 1.0f

Конспект Егора

(То же самое + код)

"Одни делают что-то полезное, а другие пишут стандарт."
— Скаков П. С.

Чего нет в C++

- VLA

Как компилятор понимает что компилировать нужно на C++

1. `-std=c++20`
23
`g++` - не надо использовать (гнусная версия)
2. На основе расширения `.c` || `'c++'`

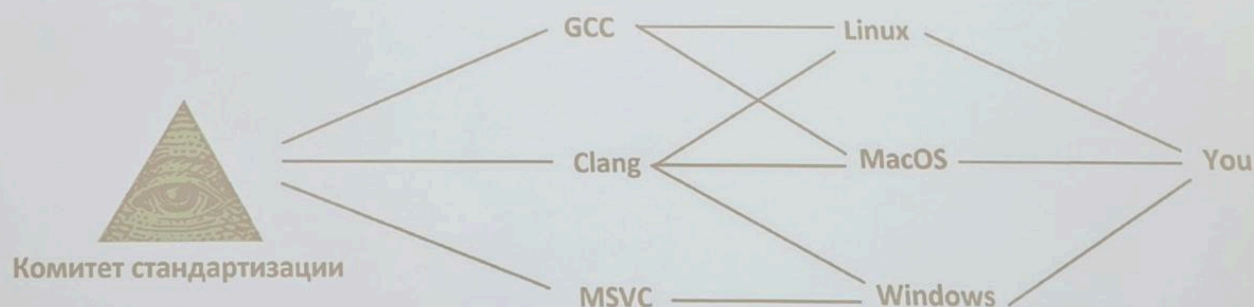
Хорошо:

`clang main.cpp` - работает
`clang == clang++`, можно вводить `clang++` вместо `clang`

Важно:

Следи за тем, **в каком стандарте компилируется прога** т.к. в C++ стандарты сильно отличаются

Свой комитет по стандартизации и свои стандарты



.2025

Интересно

- Самая первая версия C++ - просто препроцессор который компилировал .c файл чуть по-другому
- Если что-то добавили в стандарт - не факт что это будет в компиляторе C++
- C++ по большей части синтаксический сахар для C

Ссылки - константный указатель

Если передавать в функцию не ссылкой, то объект будет копироваться.

& - указатель на то, что передаваемый аргумент ссылка

C

```
void f(int y, int *q)
{
    y = 2;
    *q = 3;
}

main()
{
    int x = 1, z = 0;
    int *p = &x;
```

```
f(z, &p); // z - не поменяется, а x - поменяется т.к. присвоение через  
указатель  
}
```

C++

```
void f(int &y, int *q)  
{  
    y = 2;  
    *q = 3;  
}  
  
main()  
{  
    int x = 1, z = 0;  
    int *p = &x;  
    f(z, &x); // z - поменяется т.к. передается ссылка (==2), а x - поменяется  
    т.к. присвоение через указатель (==3)  
}
```

Можно просто создать ссылку:

```
int &r = z;
```

Когда не собираешься ничего менять по ссылке, то делай const ссылку в функции
Можно сделать так **только когда есть const**:

```
void f(const int &y, int *q)  
{  
    y = 2;  
    *q = 3;  
}  
  
f(1, &x);
```

Здесь создается новый объект для единицы

```
int &f(const int &y, int *q)  
{  
    int k = y + *q;  
    return k;  
}
```

```
int x = 1, z = 0;
int *p = &x;
int &r = z;
int w = f(1, &x); - ФИГНЯ т.к. возвращаем ссылку на локальную переменную,
которая уничтожится после отработки функции
```

кроме того можно:
f(1, &x) = 3; функция станет тройкой => нужно делать константные ссылки на функции

Значения по умолчанию

Пример использования - debug режим в функции который дефолтно равен false

Нужно указывать такие переменные в самом конце, если чередовать - не скомпилился

Примеры:

```
int f(int x, int y = 3, float z = 4.5f);

f(1, 2) == f(1, 2, 4.5f)
```

Перегрузка

! Возможна только по принимаемым аргументам, не по возвращаемым значениям

```
float s(float x) {
    return x+2.25;
}

double s(double x) {
    return x + 1.43453234234;
}

long double s(long double x) {
    return x + 3.23322313213213123132123;
}
```

Как работает:

В C++ функции на самом деле называются не так, как мы их называем внутри программы, в отличие от C. К имени функции добавляются закодированные типы элементов (если переменных очень много, то имя может получиться очень большим и поэтому на этапе компиляции C++ потребляет ОООООООООООООООООООЧЕНЬ много памяти) и тогда линкер может распарсить разные функции, который внутри кода будут называться одинаково

! Этот фикс компиляторовазависим

Как сделать так чтобы имена функций были такими какие они есть (как в C)

```
extern "C" int f(int x);
```

```
extern "C" – все функции внутри будут своими именами названы
{
int f(int x);
int def(double y);
}
```

Определяем компилируется в режиме C || C++ => убирать некоторые строчки кода если в режиме C

```
#ifdef __cplusplus
extern "C" – все функции внутри будут своими именами названы
{
int f(int x);
int def(double y);
}
#endif
```

ОШАБКА:

```
float s(float x) {
    return x+2.25;
}

double s(double x) {
    return x + 1.43453234234;
}

long double s(long double x) {
    return x + 3.23322313213213123132123;
```

```
}
```

```
float a = s(3/2); // 3/2 возвращает, а у нас нет функции принимающей int =>  
ОШИБКА
```