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
```
или &x

Стало в C++

```
int x=1, z=0;
int *p = &x;
void f(int &y, int *q)
{
    y=2;
    *q=3;
}
f( z, p)   // z=2; x=3
```
ссылка

нельзя поставить const т.к. берем адрес

& в C++ это:

- И
- взятие адреса
- при создании переменной это ссылка

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

Const для ссылки тоже работает как для указателя (если не собираемся менять)

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

```
f( p, &x)   // x = 3
```
(можем передать const)

потому что создается объект и берется адрес

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

- int f(...) → память станет не валидна (вернет мусор)

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

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

если передаем большую структуру куда-то то она копируется, а можно передать ссылку

Значение по умолчанию

```
int f(int x, int y = 3, float z = 4,5f);
```

можно передавать свои аргументы, но если не передать, то дополнится реализацией

f(1, 2) → f(1, 2, 4.5f)

- можно использовать для дебага

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

Перегрузка

```
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 линкеру выбрать нужную s? → Компилятор добавляет доп. инфу в имя функции и линкер их различает. (использует @, ? ... поэтому мы не пересечемся)

функция без магии (не меняем имя и невозможна перегрузка)

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

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

(13) `extern "C"` — незаконная конструкция в C

фикс: компилим `extern "C"` в C++, а в C не надо

```
#ifdef __cplusplus
extern "C"
{
#endif
+
#ifdef __cplusplus
}
#endif
```

```
float a = f(2);
```
→ ошибка компиляции (нет точного значения)

```
float x = 3/2; // x = 1.0f
```
(int)

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++

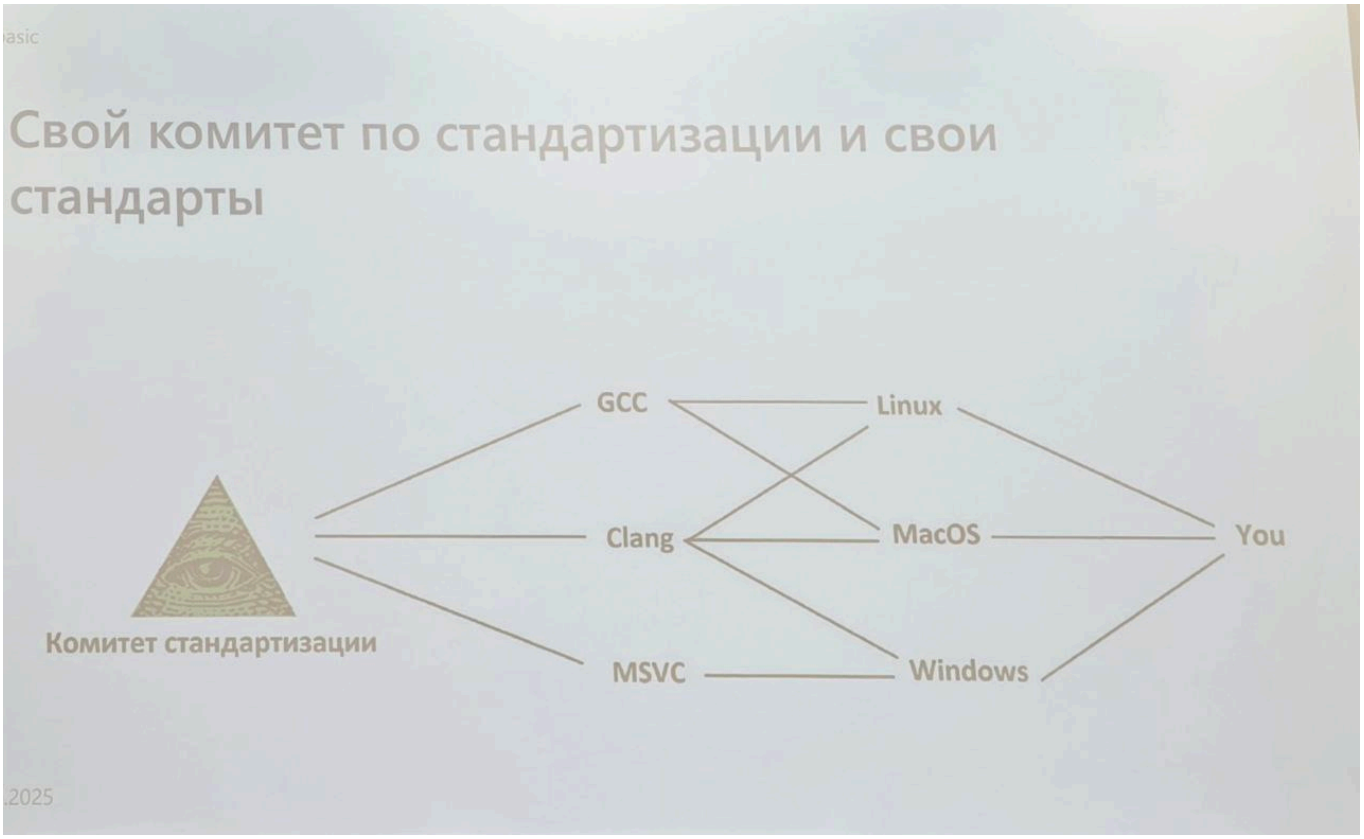
1. -std=c++20
   23
   gnu++ - не надо использовать (гнусная версия)
2. На основе расширения `.c` || '.c++'

Хорошо:

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

Важно:

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



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

```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 создастся 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;
inf w = f(1, &x); - ФИГНЯ т.к. возвращаем ссылку на локальную переменную, которая унистожится 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 равна false

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

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 ОООООООООООООООООЧЕНЬ много памяти) и тогда линкер может распарсить разные функции, который внутри кода будут называться одинаково

! Этот фикс компиляторозависим

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

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

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

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

```
#ifdef __cplusplus
exten "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
```