

Предусловие:



Неймспейс - работает как префикс для названия переменных

```
namespace ABC
{
    int f()
    {
        return r;;
    }
    float x;
}
```

ОШИБКА: неизвестное имя f(), оно скрыто в пр-ве имен

```
int main()
{
    return f();
}
```

Работает!

```
int main()
{
    ABC::return f();
}
```

В неймспейс кроме x будет ещё добавлен z и ещё одн неймспейс внутри\

```
namespace ABC
{
    int z;
    namespace F
    {
        float f;
    }
}

int main()
{
    float x = ABC:F:f;
}
```

Обращение к глобальной переменной

```
int x;  
using namespace ABC;  
int main()  
{  
    int x = ::x;  
    return f();  
}
```

Использование переменных из другого неймспейса в своём неймспейсе

```
namespace F  
{  
    float x;  
}  
namespace ABC  
{  
    using f::x;  
    int f()  
    {  
        return 5;  
    }  
}
```

Анонимный неймспейс - автоматически компилятор присваивает ему глобально уникальное "магическое имя". (аналог Static переменных)

```
namespace  
{  
    namespace f()  
    {  
        return 5;  
    }  
} компилятор автоматически здесь напишет: using namespace
```

**ВСЁ ЧТО ВЫ ИМПОРТИРУЕТЕ - НАХОДИТСЯ В
`namespace std`**

```
#include <math.h>:
    Напрямую подключается .h файл
#include <cmath>:
    КОСТЫЛЬ ИДИОТИЗМ ТУПИЗНА БРЕД
    using namespace std
#include <math.h>
```

Аналог malloc-ов

- Функции из C всё ещё доступны
- Системные функции выделения памяти всё ещё доступны

Новый способ выделения памяти: **new**

Есть 2 new, 1 обычный и другой с квадратными скобками (для массивов)

```
int *p = new int(2);
или
int *p = new int{2};
```

Удаление для обычного new:

```
delete p;
```

```
int *p = new int[5];
```

ЭТО ОШИБКА, ЖЁСТКИЙ UB:

```
delete p;
```

РАБОТАЕТ:

```
delete[] p;
```

ЗАЧЕМ new?

- Malloc - тупо выделяет нужное кол-во байтов
Мы рассматривали конструкторы, они автоматически вызываются при создании объекта.
MALLOC НЕ МОЖЕТ ВЫЗВАТЬ КОНСТРУКТОР.

Пример работы new (взаимодействие с конструкторами и деструкторами)

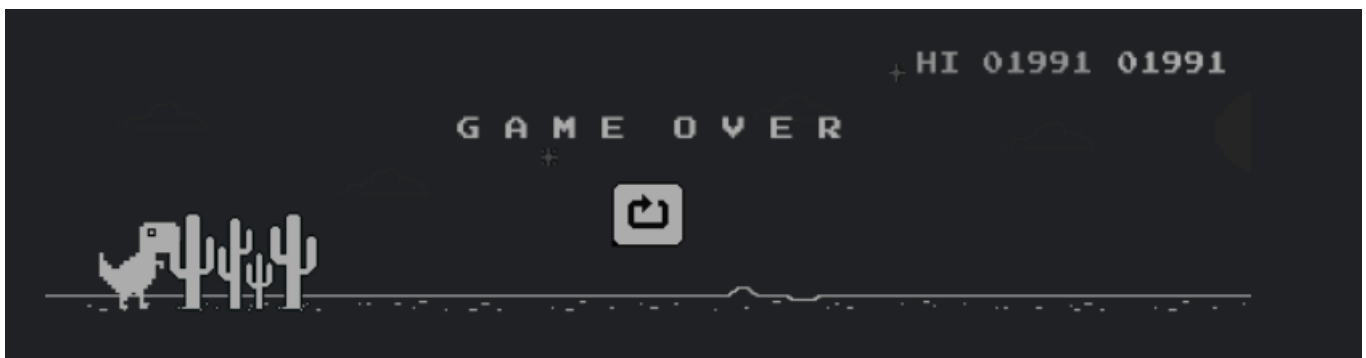
```
struct s
{
    s()
```

```

{
    printf("created\n");
}
~s()
{
    printf("destroyed");
}
}

```

Сначала 5 раз вызовется конструктор
 Потом 5 раз вызовется деструктор
 s *p = new S[5];
 delete[] p;



Виртуальный деструктор

Если деструктор у A – не виртуальный, то он вызовется по A, иначе – по B
 A *p = new B;
 delete p;

! В случае неудачи new бросат исключение

new будет бросать null если будет ошибка

Если деструктор у A – не виртуальный, то он вызовется по A, иначе – по B
 A *p = new(std::nothrow) B;
 delete p;

Конструктор и деструктор с malloc

```

#include <new>
struct S

```

```

{
    void f()
    {
        СОЗДАНИЕ БЕЗЫМЯННОГО ОБЪЕКТА ТИПА S
        s();
    }
}

```

В этом примере мы сначала отдельно выделяем память, а потом отдельно вызываем конструктор:

```
A *p = (A*) malloc(sizeof(A));
```

ТАК НЕ СРАБОТАЕТ:

```
p->A();
```

СРАБОТАЕТ, placement new. Здесь мы вызываем конструктор:

Вызывает конструктор внутри указателя.

```
new(p) A;
```

РАБОТАЕТ:

```
p->~A();
```

```
free(p);
```

```
char c[sizeof(A)];
```

```
A *p = new(c) A;
```

```
p->~A();
```

1. char c[sizeof(A)];

- Здесь создаётся массив байт (char) размером равным размеру класса A .
- Это сырая (неинициализированная) память, которая будет использована для размещения объекта A .

2. A *p = new(c) A;

- Это **placement new** — особая форма оператора new , которая не выделяет память, а использует переданный ей указатель (c) для конструирования объекта.
- Объект типа A создаётся в заранее выделенной памяти (в массиве c).
- Возвращается указатель p на созданный объект.

3. p->~A();

- Здесь происходит **явный вызов деструктора** для объекта, на который указывает `p`.
- Обычно деструктор вызывается автоматически, когда объект уничтожается (например, через `delete` или при выходе из области видимости), но в данном случае память управляется вручную, поэтому деструктор тоже вызывается вручную.

Перегрузка операторов

- ТРЕБОВАНИЕ: хотя бы 1 аргумент должен быть того типа, от которого вызывается операция
- Перегруженные операторы наследуют свойства своих изначальных операторов
SKKV:
функции вычисляют аргументы не лениво, есть хуйня, которая лениво
- Не всё можно перегрузить:
 1. `::` - оператор определения области
 2. `*`
 3. `?` - тернарный оператор
 4. `sizeof`
 5. `&&`
 6. `||`
 7. `.`
- Можно:
 1. Оператор вызова функции
 2. Оператор `[]` (например в `Vector`)
 3. Знак `=`

```
struct S
{
    int a, b;
}

S x, y, z;
ОШИБКА: неизвестно как складывать структуры
z = x + y;
```

Пример перегрузки операторов:

```
struct S
{
```

```
int a, b;
```

Оператор во внутренней форме: (первый аргумент неявный this)

S operator+(const S &c) <const> (сделает тип this – const. Тогда это эквивалентно "внешней форме перегрузки оператора")

```
{
    S res;
    res.a = a + c.a;
    res.b = b + c.b;
    return res;
}
```

S operator[](int z) <const> (сделает тип this – const. Тогда это эквивалентно "внешней форме перегрузки оператора")

```
{
    S res{1, 2+2};
    return res;
}
}
```

Оператор во внешней форме:

S operator-(const S &x, const S &y)

```
{
    S res;
    res.a = x.a - y.a;
    res.b = x.a - y.b;
    return res;
}
```

S x, y, z;

ОШИБКА: неизвестно как складывать структуры

z = x + y;

Две эквивалентных записи (благодаря перегрузке по оператору)

```
S z = x[3];
z = x.operator[](3);
```

! friend - в основном используется для операторов т.к. какие-то операторы невозможно описать во внутренней форме

Перегрузка инкремента префиксного и постфиксного:

Для постфиксного добавлен костыль, благодаря которому принимается число.

Изначально возвращается значение объекта, а мы хотим вернуть именно сам объект, поэтому делаем *this

```
S& opetor+=(const S & y)
```

```
{  
  
    return *this;  
}
```

Различия присвоения в С и С++

- В С:

ОШИБКА:

`(a = b) = c` - не сработает, тут `b` присвоится `a` и вернётся значение присвоения, а значению присвоения невозможно что-то присвоить

- В С++:

`(a = b) = c` - сработает, тут `b` присвоится `a` и вернётся объект, объекту присвоится значение `c`

УРА!!!!!!!!!!!!!!!!!!!!!!

(▽) ♥♥♥♥ Виктория ♥♥♥♥ (° ▽ °)

Гитбук чекай!

Приоритет операций: https://en.cppreference.com/w/cpp/language/operator_precedence

Мы будем использовать QT, ссылка в таблице.

Этапы:

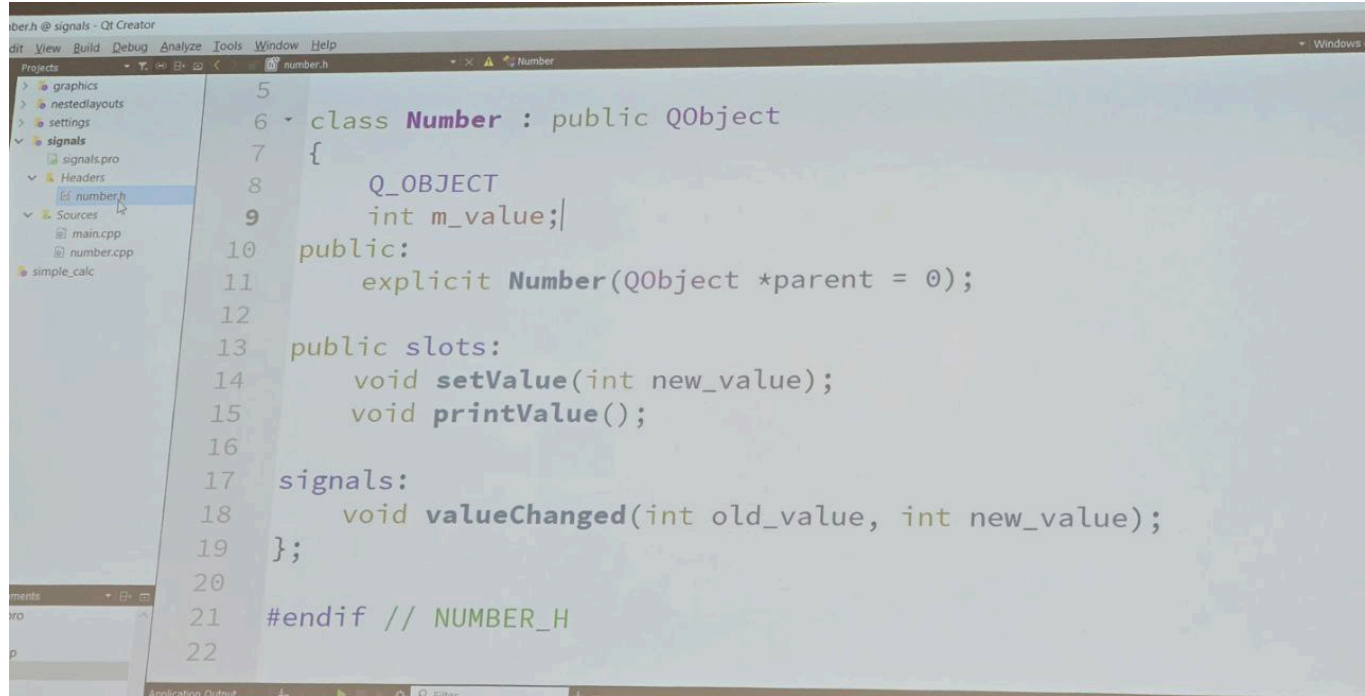
1. Скачать -> таблица курса
2. Отключаем интернет
3. Устанавливаем:
Опционально:
4. QtCreator - IDE

.pro файл - проект файла QMake, он похож на CMAKE

```
SOURCE += /  
    main.cpp  
    window.cpp  
HEADERS += \  
    windows.h  
QT += widgets opengl
```


Сигналы и слоты - организация системы через которую реализуют реакцию на например нажатие кнопки и вообще всю работу с интерфейсом

Сигналы нужно объявить в .h файле (slots нужно для старого синтаксиса, в новом синтаксисе он не обязателен):



```
5
6 class Number : public QObject
7 {
8     Q_OBJECT
9     int m_value;
10 public:
11     explicit Number(QObject *parent = 0);
12
13 public slots:
14     void setValue(int new_value);
15     void printValue();
16
17 signals:
18     void valueChanged(int old_value, int new_value);
19 };
20
21 #endif // NUMBER_H
22
```

Ключевые слова:

1. signals
2. emit - на защите тут унижают если не знаешь
3. connect

emit - выпускает сигнал в окружающее пространство:

```
emit valueChanged(a, b)
```

Чтобы работать с методами QT нужно:

1. Наследоваться от QObject
2. В самом начале прописать макрос Q_OBJECT

К одному сигналу может быть подключено сколько угодно слотов и наоборот. Нужно присоединять сигнал к тому слоту, который принимает столько аргументов, от скольких переменных выпущен сигнал. Иначе оставшаяся часть переменных обрежется.

*MOC - можно почитать

QObject::connect(<&источник>, <событие-инициатор>, <&приемник>, <событие-следствие>)

На примере Number:

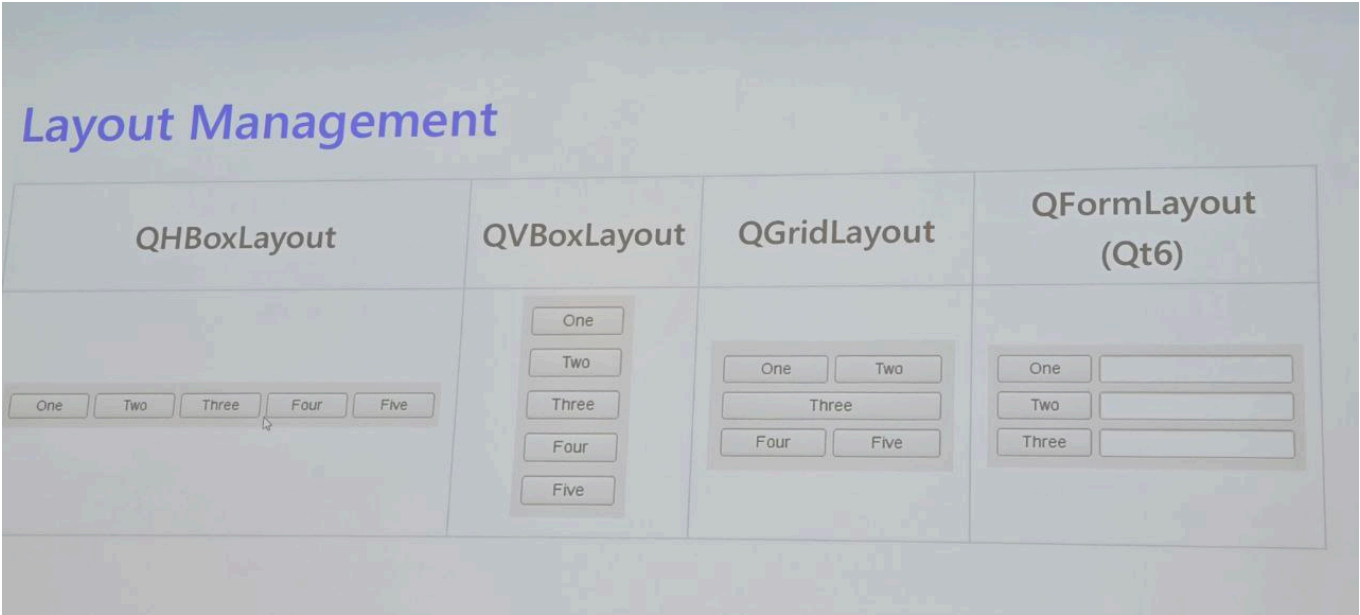
```
int main(int argc, char** argv)
{
    Number a,b;
    a.setvalue();

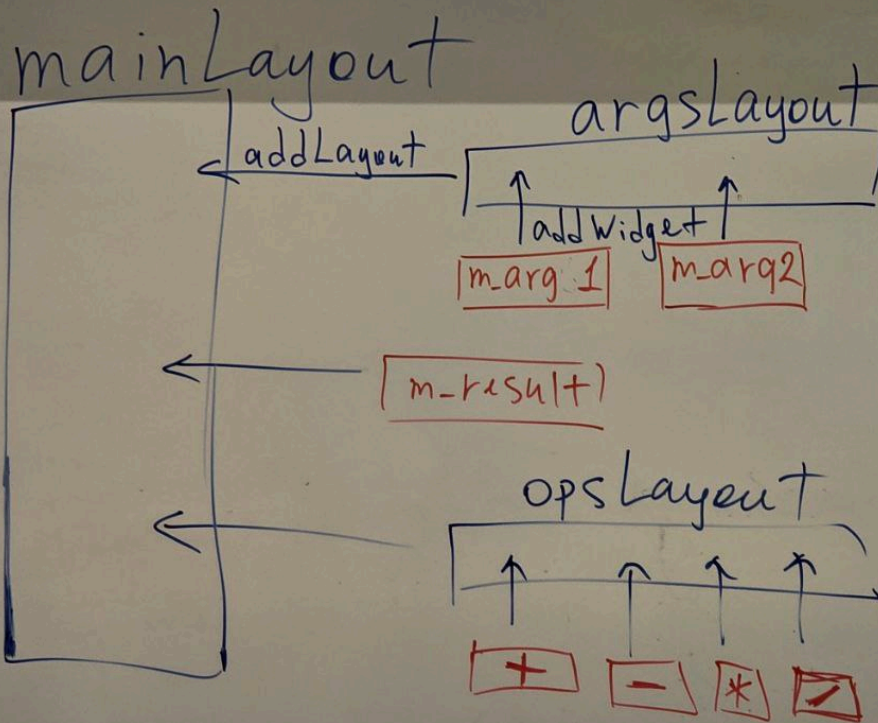
    Когда у а будет вызван сигнал valueChanged, то вызовется функция printvalue в
    а
    QObject::connect(&a, &Number::valueChanged, &a &Number::printValue);
    QObject::connect(&a, &Number::valueChanged, &b &Number::setValue);
    QObject::connect(&b, &Number::valueChanged, &b &Number::printValue);
}
```

a	a	b
slot	setvalue	setvalue
slot	printv	printv
signal	value Changed	value Changed

Интерфейс

4 вида лэяута нет в нашей версии QT

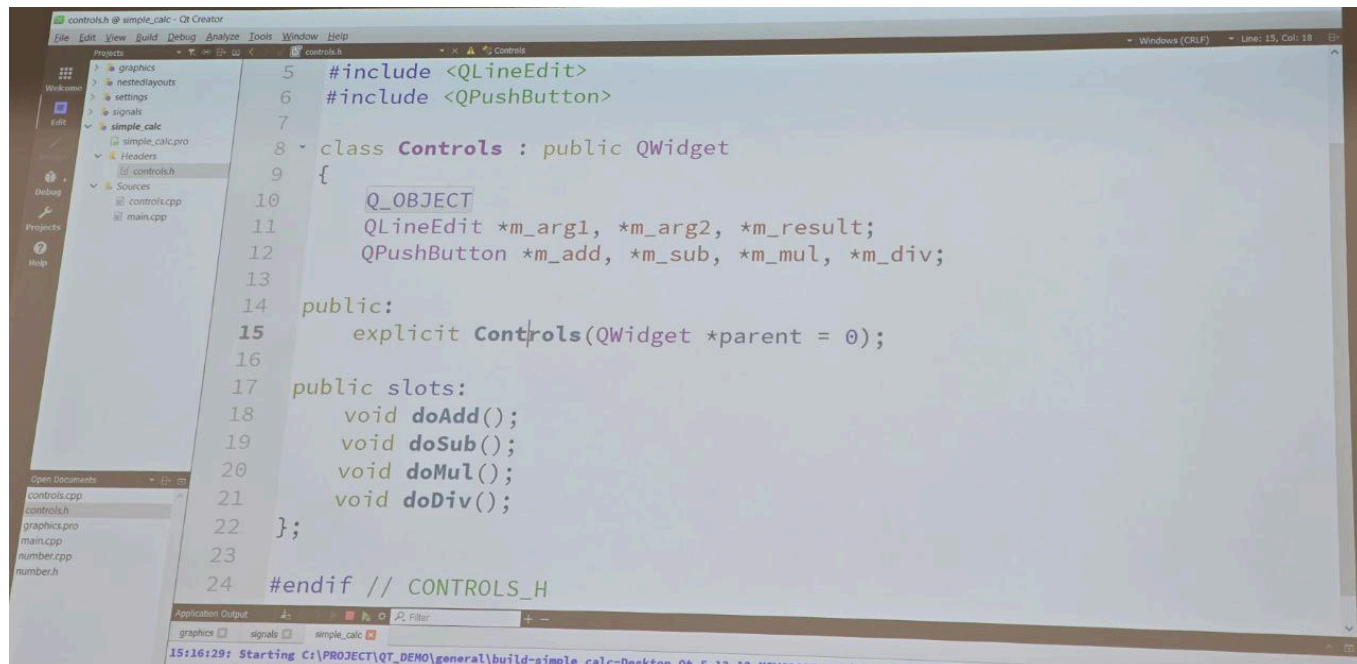




```

controls.cpp @ simple_calc - Qt Creator
File Edit View Build Debug Analyze Tools Window Help
main.cpp controls.h controls.cpp
4 Controls::Controls(QWidget *parent) : QWidget(parent)
5 {
6     QVBoxLayout *mainLayout = new QVBoxLayout;
7
8     QHBoxLayout *argsLayout = new QHBoxLayout;
9     argsLayout->addWidget(m_arg1 = new QLineEdit);
10    argsLayout->addWidget(m_arg2 = new QLineEdit);
11    mainLayout->addLayout(argsLayout);
12
13    mainLayout->addWidget(m_result = new QLineEdit);
14
15    QHBoxLayout *opsLayout = new QHBoxLayout;
16    opsLayout->addWidget(m_add = new QPushButton("+"));
17    opsLayout->addWidget(m_sub = new QPushButton("-"));
18    opsLayout->addWidget(m_mul = new QPushButton("*"));
19    opsLayout->addWidget(m_div = new QPushButton("/"));
20    mainLayout->addLayout(opsLayout);
21
22    setLayout(mainLayout);
23
  
```

controls.h для калькулятора:



The screenshot shows the Qt Creator IDE with the 'controls.h' file open. The file contains C++ code for a calculator's control interface. The code includes headers for QLineEdit and QPushButton, and defines a Controls class that inherits from QWidget. The class has private attributes for QLineEdit and QPushButton objects, and public slots for doAdd(), doSub(), doMul(), and doDiv(). The class constructor is explicit and takes a QWidget parent. The file ends with a #endif // CONTROLS_H directive. The left sidebar shows the project structure with 'simple_calc' as the active project. The bottom status bar shows the application output.

```
controls.h @ simple_calc - Qt Creator
File Edit View Build Debug Analyze Tools Window Help
Projects
  graphics
  nestedlayouts
  settings
  signals
  simple_calc
  simple_calc.pro
  Headers
    controls.h
  Sources
    controls.cpp
    main.cpp
Open Documents
  controls.cpp
  controls.h
  graphics.pro
  main.cpp
  number.cpp
  number.h
15:16:29: Starting C:\PROJECT\QT_DEMO\general\build-simple_calc-Desktop_Qt_5.12.12_MSVC2017_64bit_...
15:16:29: Starting C:\PROJECT\QT_DEMO\general\build-simple_calc-Desktop_Qt_5.12.12_MSVC2017_64bit_...
```

На гитбуке чекните, мне лень дальше писать