

# Информатика. Семинар №2

[https://dl.dropboxusercontent.com/  
u/96739039/sem4/infa\\_s02.pdf](https://dl.dropboxusercontent.com/u/96739039/sem4/infa_s02.pdf)

# Разбиением кода на отдельные файлы

- Каждая структура/класс – отдельный файл
- Для использования этого класса – директива `#include`:

`#include "Vector2.h"` – в локальной папке проекта

`#include <Vector2.h>` - в специально указанных директориях

# Отделение объявления от реализации

- компиляция + линковка
- директива `#pragma once`
- время вызова + время работы функции
- директива `inline`

# Умножение числа на Vector2

```
49  ··Vector2·operator*(float·k)·const
50  ··{
51  ····return·Vector2(x·*·k,·y·*·k);
52  ··}
53
54  ··float·x,·y;
55  };
56
57  Vector2·operator*(float·k,·const·Vector2&·v)
58  {
59  ··return·Vector2(v.x·*·k,·v.y·*·k);
60  }
```

# Перегрузка операторов ввода/вывода (>> / <<) для Vector2

```
62  #include <iostream>
63  std::ostream& operator<<(std::ostream& stream, const Vector2& v)
64  {
65      stream<<v.x<<" "<<v.y;
66      return stream;
67  }
68
69  std::istream& operator>>(std::istream& stream, Vector2& v)
70  {
71      stream>>v.x>>v.y;
72      return stream;
73  }
74
75  int main()
76  {
77      Vector2 v;
78      std::cin>>v;
79      std::cout<<v<<" "<<v.Len()<<"\n";
```

# Константные методы класса

```
63  ▢ ··Vector2&·operator+=(const·Vector2&·other)
64  ··{
65  ····x·+=·other.x;
66  ····y·+=·other.y;
67  ····return·*this;
68  ··}
69
70  ▢ ··float·Len()·const
71  ··{
72  ····return·sqrt(x·*·x·+·y·*·y);
73  ··}
74  ▢
75  ··float·x,·y;
76  };
77
78  ▢ int·main()
79  {
80  ··const·Vector2·v(3,·4);
81  ▢ ··std::cout·<<·v.Len()·<<·"\n";·//·ok
82  ··//·v·+=·Vector2(1,·1);·.....//·error
```

# Упражнение

Реализовать класс Vector2 со следующими методами:

- `operator <<, >>` (перегрузка ввода и вывода)
- `Vector2 operator+(const Vector2& other) const;`
- `Vector2& operator+=(const Vector2& other);`
- аналогично - и -=
- `float operator*(const Vector2& other) const;` (скалярное произведение)
- `float operator^(const Vector2& other) const;` (векторное произведение)
- \* на скаляр с «2х двух сторон»
- / на скаляр
- единичный вектор `norm()`
- перпендикуляр `(y, -x)`
- `len()`, `squareLen()`
- унарный минус `Vector2 operator-() const`
- конструкторы
- поворот на угол 2шт.: `rotate` и `getRotated` (первый меняет себя, второй – возвращает новое значение ... по аналогии с += и +)

Работу методов нужно проверить: в main'е вызвать каждый хотя бы по разу.

# Принципы ООП: инкапсуляция

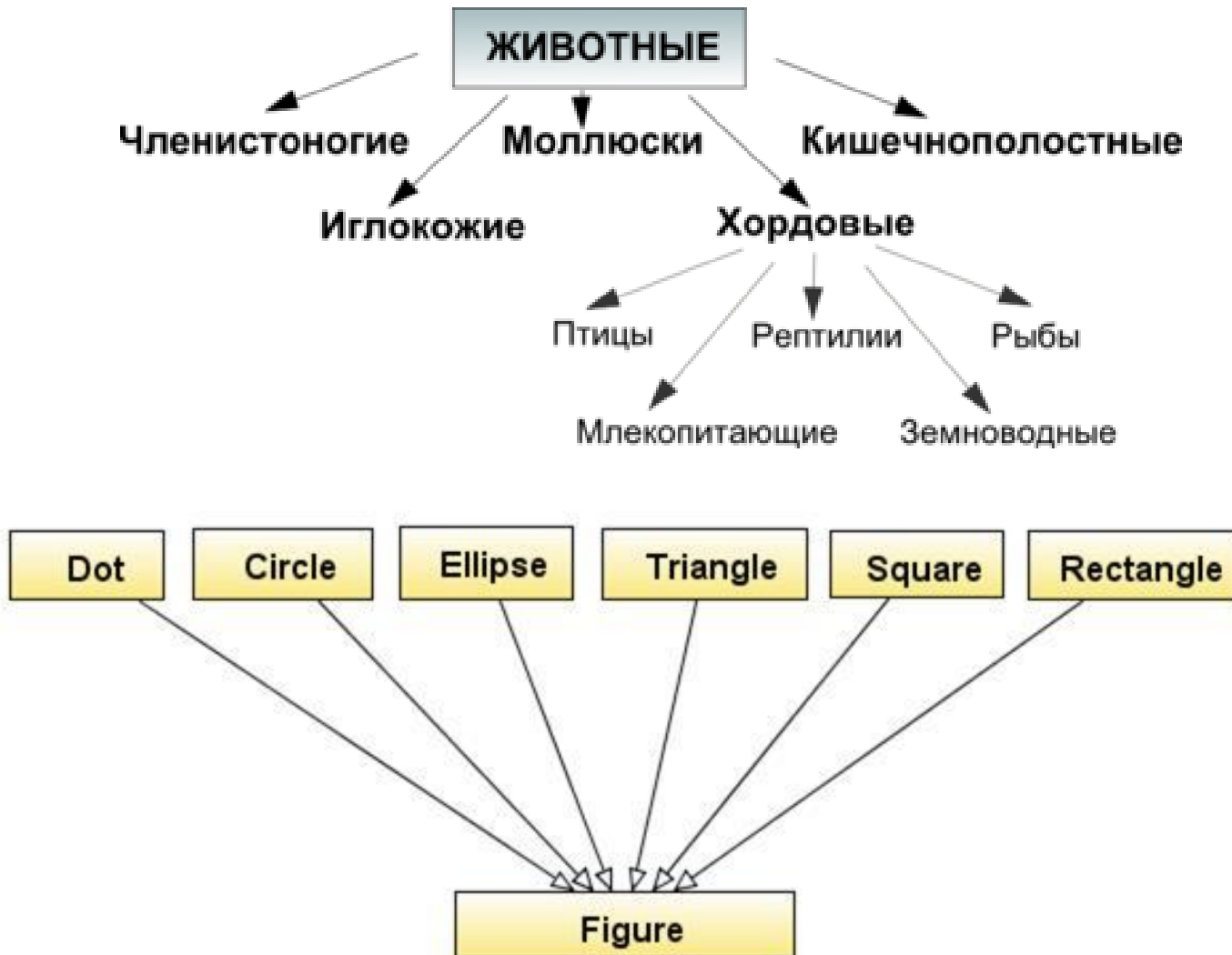
- **Инкапсуляция** (по-русски: «сокрытие») — это свойство объектов скрывать некоторые свои данные и способы их обработки (методы) от окружающей его цифровой среды и, в частности, от кривых ручонок малоопытных программистов, оставляя «снаружи» только необходимые и/или требуемые свойства и функциональные возможности.
- <http://avolberg.ru/theory/oop/encapsulation>



# Принципы ООП: наследование

- **Наследование** — важный механизм объектно-ориентированного подхода, позволяющий расширить и/или изменить структуру уже существующего (родительского) класса, путём написания нового класса (потомка), который полностью наследует все свойства и методы и, плюс, добавляет что-то своё. Далее можно начинать создавать и использовать в программе новые объекты с расширенными возможностями.

# Наследование



# Наследование: синтаксис

```
78 struct·Figure
79 {
80     ··int·color;
81 };
82
83 struct·Circle:·public·Figure
84 {
85     ··Vector2·center;
86     ··float·radius;
87 }
```

# protected

<i>Наследование</i>	<i>Доступ в базовом классе</i>	<i>Доступ в производном классе</i>
<i>public</i>	<i>public</i>	<i>public</i>
	<i>protected</i>	<i>protected</i>
	<i>private</i>	<i>private</i>
<i>protected</i>	<i>public</i>	<i>protected</i>
	<i>protected</i>	<i>protected</i>
	<i>private</i>	<i>private</i>
<i>private</i>	<i>public</i>	<i>private</i>
	<i>protected</i>	<i>private</i>
	<i>private</i>	<i>private</i>

# Выделение/освобождение памяти

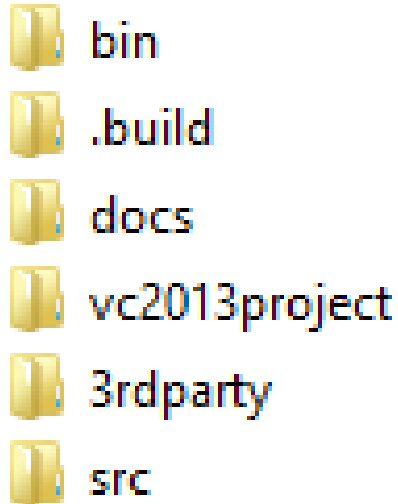
```
91  ..int* a = new int[15];
92
93  ..for (int i = 0; i < 15; ++i)
94  ..{
95  ...a[i] = i;
96  ..}
97
98  ..delete[] a;
```

В примере ниже выполняется **выделение памяти + вызов конструктора!**

P.S. Эквивалентна ли запись `circle->radius = 3 / 2` приведённой ниже ?

```
100  ..Circle* circle = new Circle();
101  ..circle->radius = 1.5f;
102  ..delete circle;
```

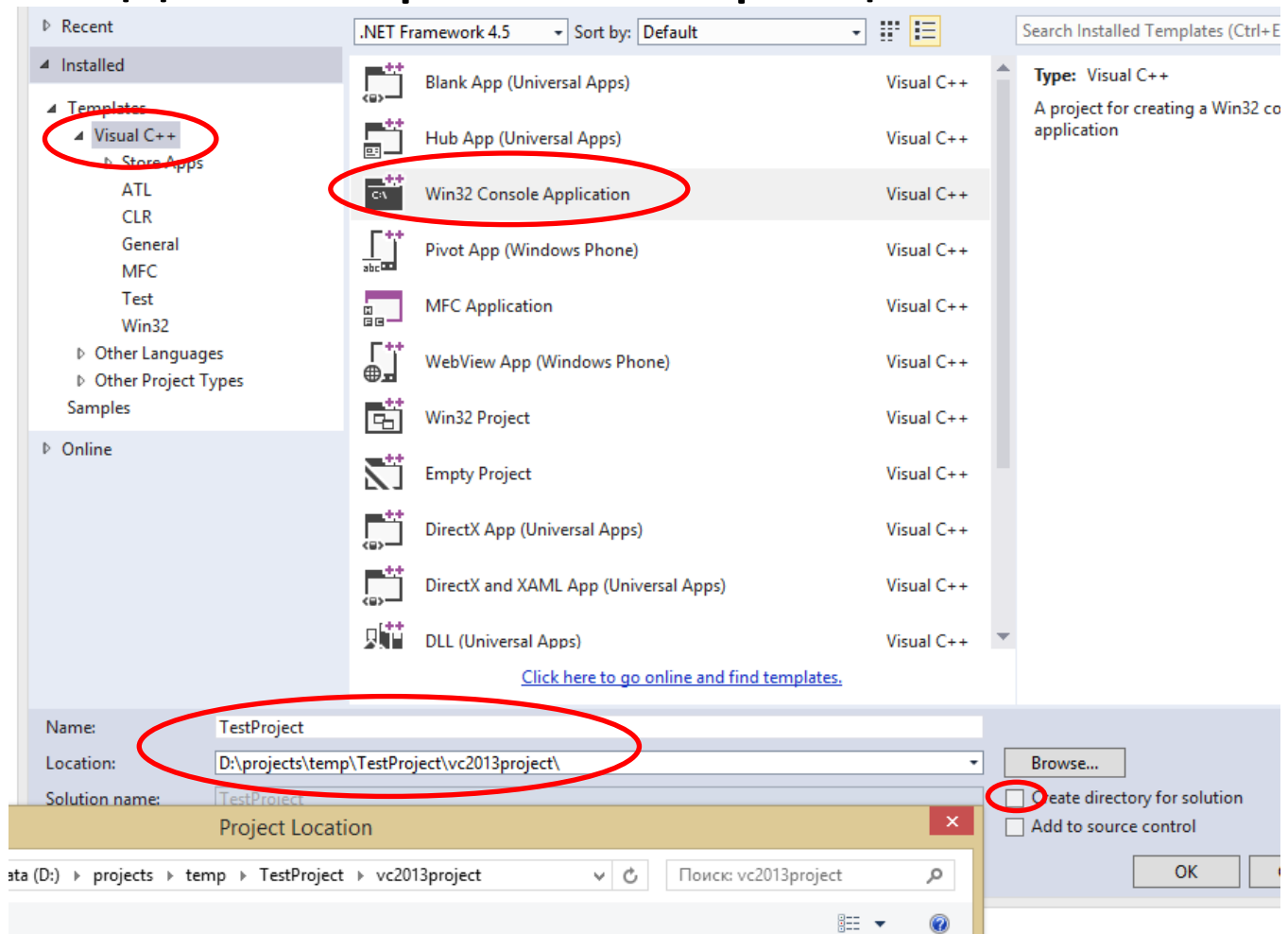
# Как организовать проект?



- bin – исполняемые файлы \*.exe, динамические библиотеки \*.dll, файлы ресурсов – фон, спрайты и т.п.
- .build – временные файлы, созданные при компиляции; пользователю и разработчику не нужны; можно сделать скрытой.
- docs – описание проекта или мат. части
- Vc201?project – хранит файл \*.sln, описывающий какие файлы входят в проект, зависимости от других библиотек и прочую служебную информацию
- 3rdparty – сторонние библиотеки
- src – исходные коды, написанные вами

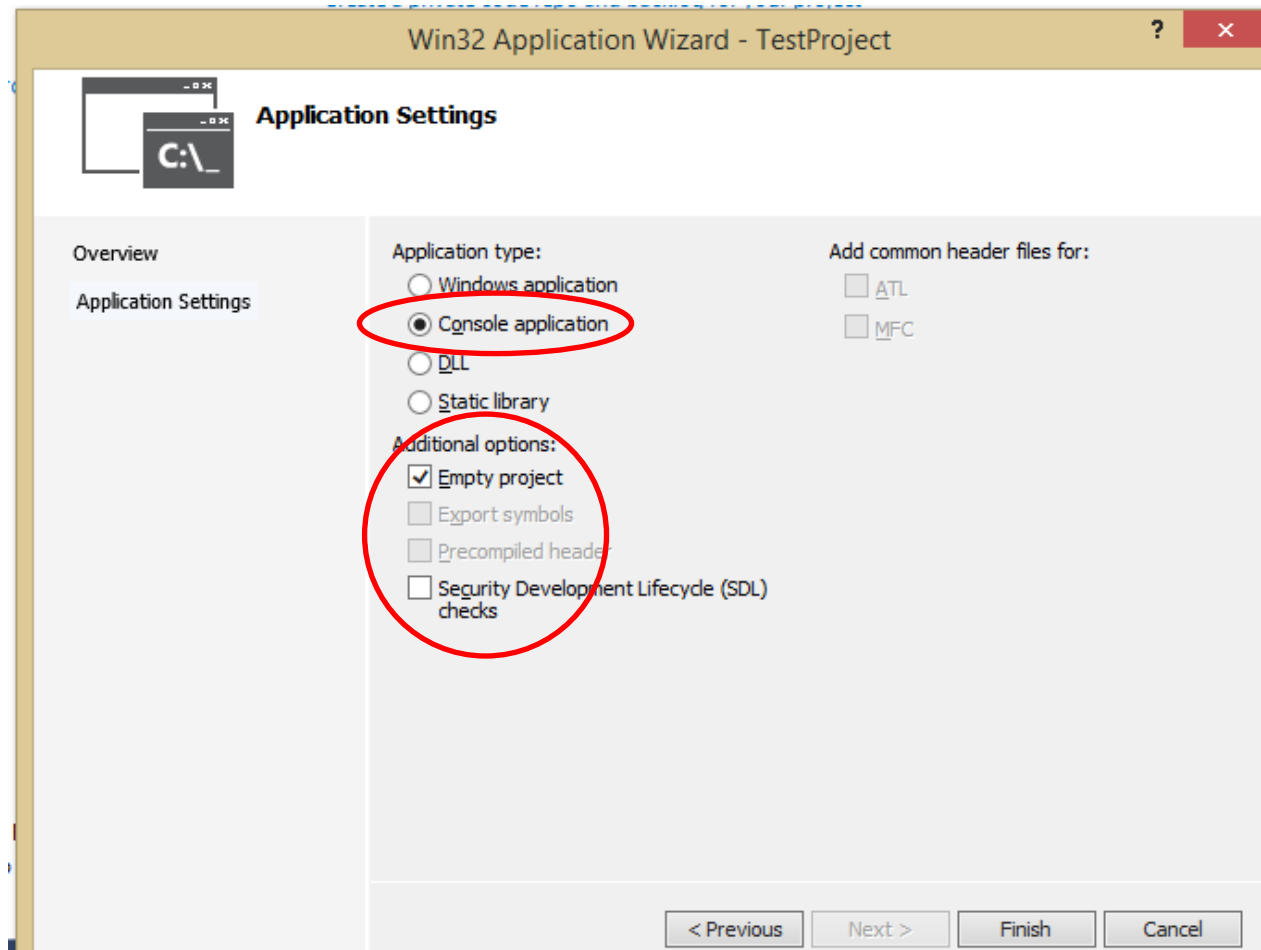
# Как организовать проект?

- При создании проекта обращаем внимание на:



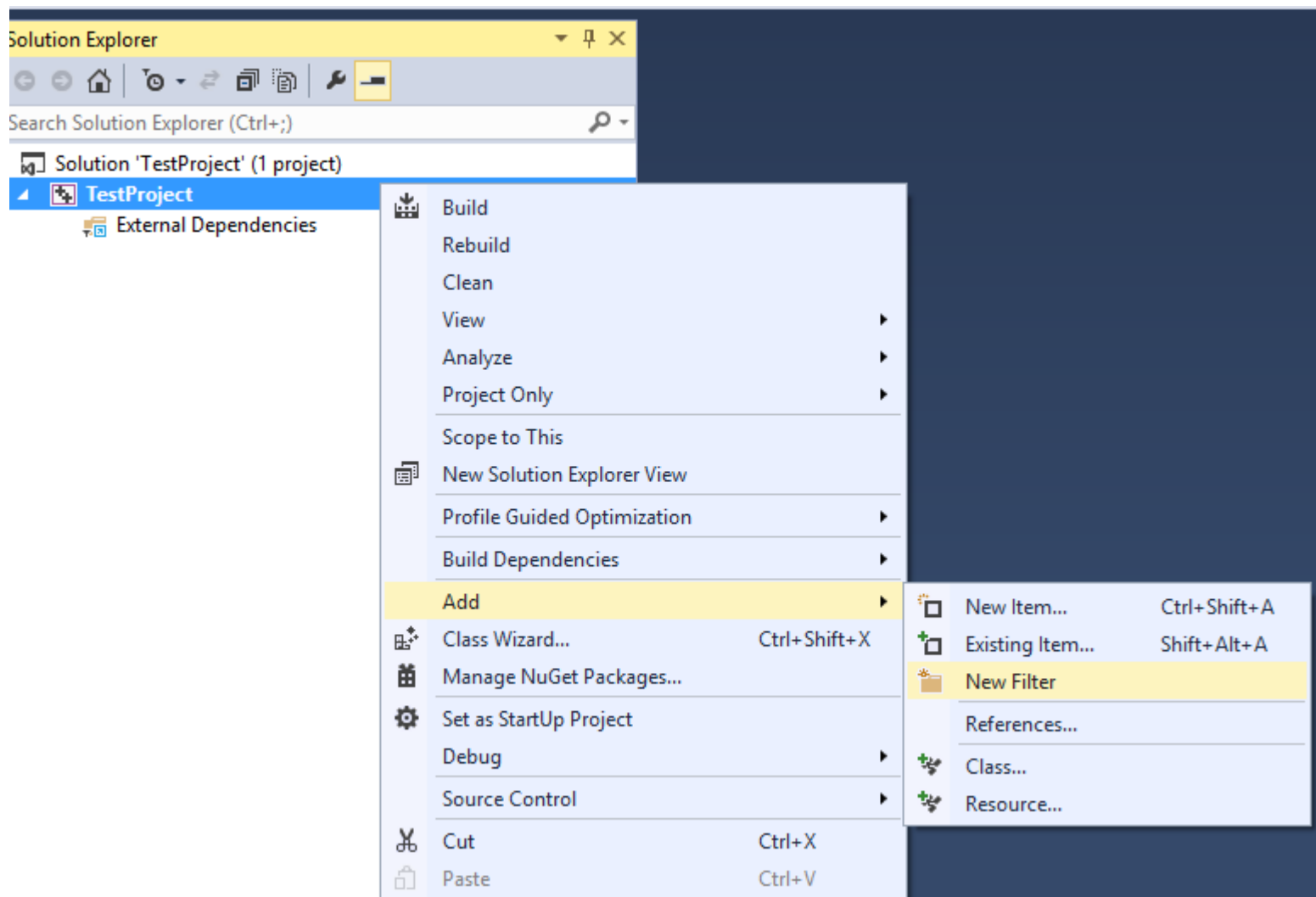
# Как организовать проект?

Обращаем внимание на обведенные «галочки»



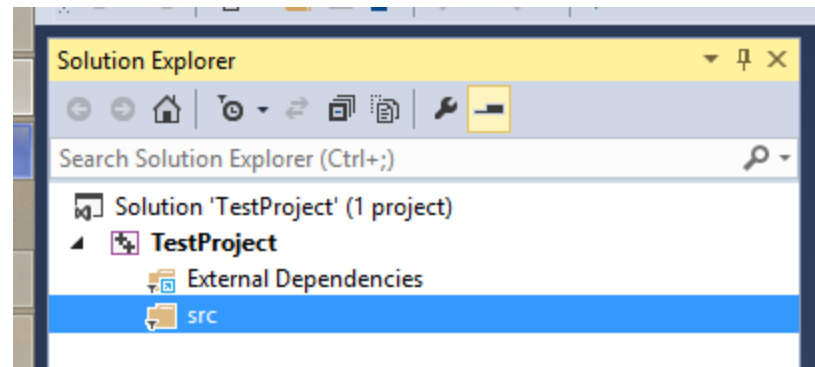


# Как организовать проект?



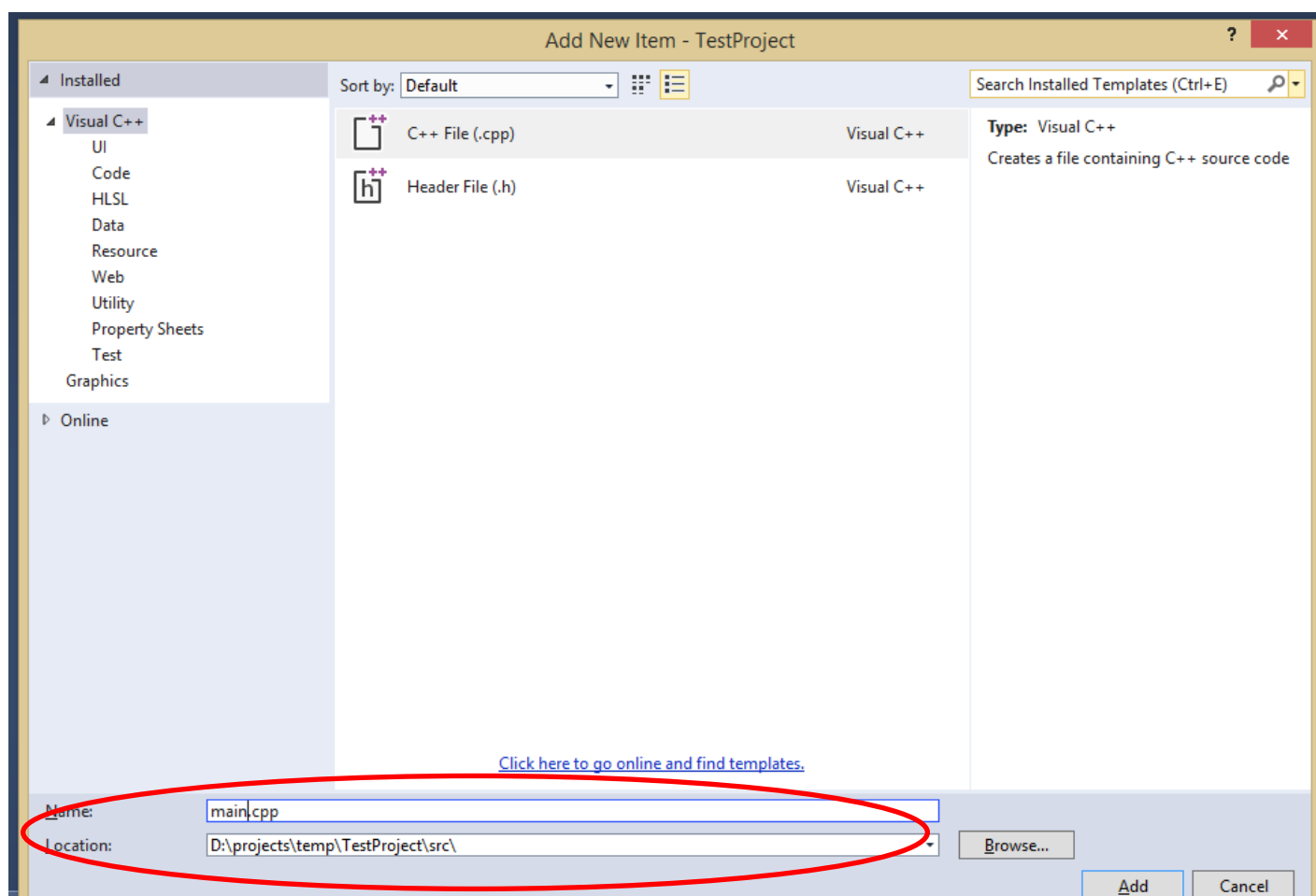
# Как организовать проект?

- Структура папок и файлов, вложенных в src на диске должна повторяться в Solution Explorer`е.



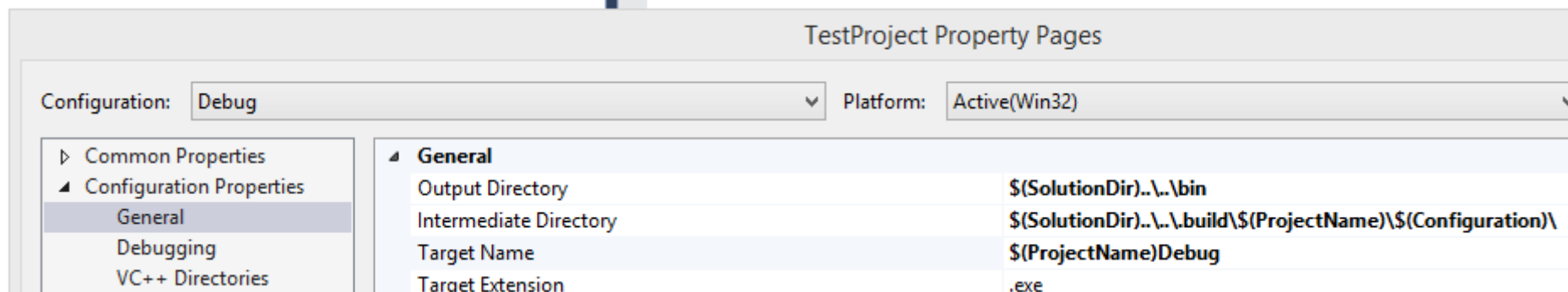
# Как добавить новый файл в проект?

Указываем правильное расположение файлов



# Настройки проекта

- Правой кнопкой мыши по название проекта, далее Свойства



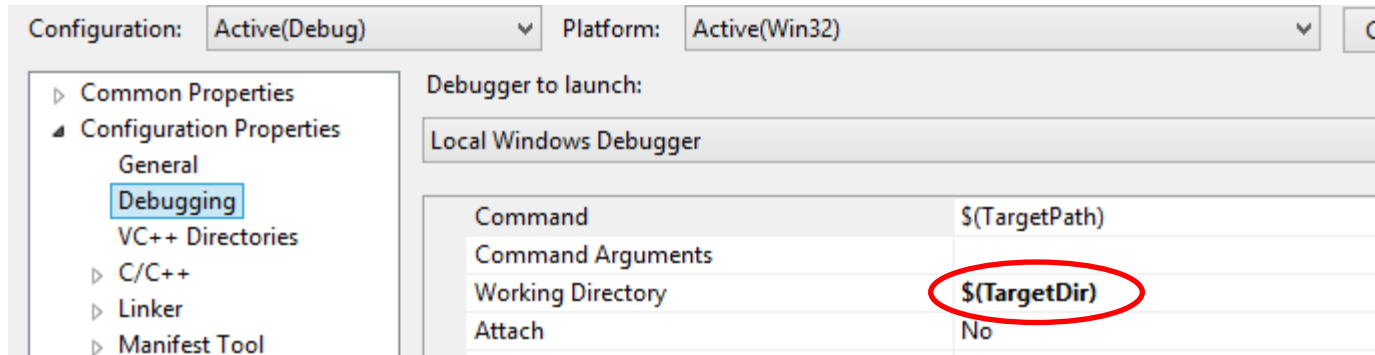
`$(SolutionDir)` – месторасположение файла \*.sln – внутри папки vc2013project

Output Directory – то, где окажутся скомпилированные вами файлы

Intermediate Directory – папка со вспомогательным «мусором».

Target Name – как будет называться ваш \*.exe файл

# Настройки проекта



Нужно, чтобы при пошаговой отладке и запуске программы из IDE текущей директорией устанавливалось место, где лежит скомпилированный \*.exe файл.

Теперь пишем программу «Здравствуй, мир!» и проверяем, что всё работает.

# Подключаем библиотеку для работы с графикой

- <http://www.sfml-dev.org/download/sfml/2.3.2/>
- Visual C++ 14 (2015) - 32-bit

Нужно разархивировать и положить в директорию 3rdparty.

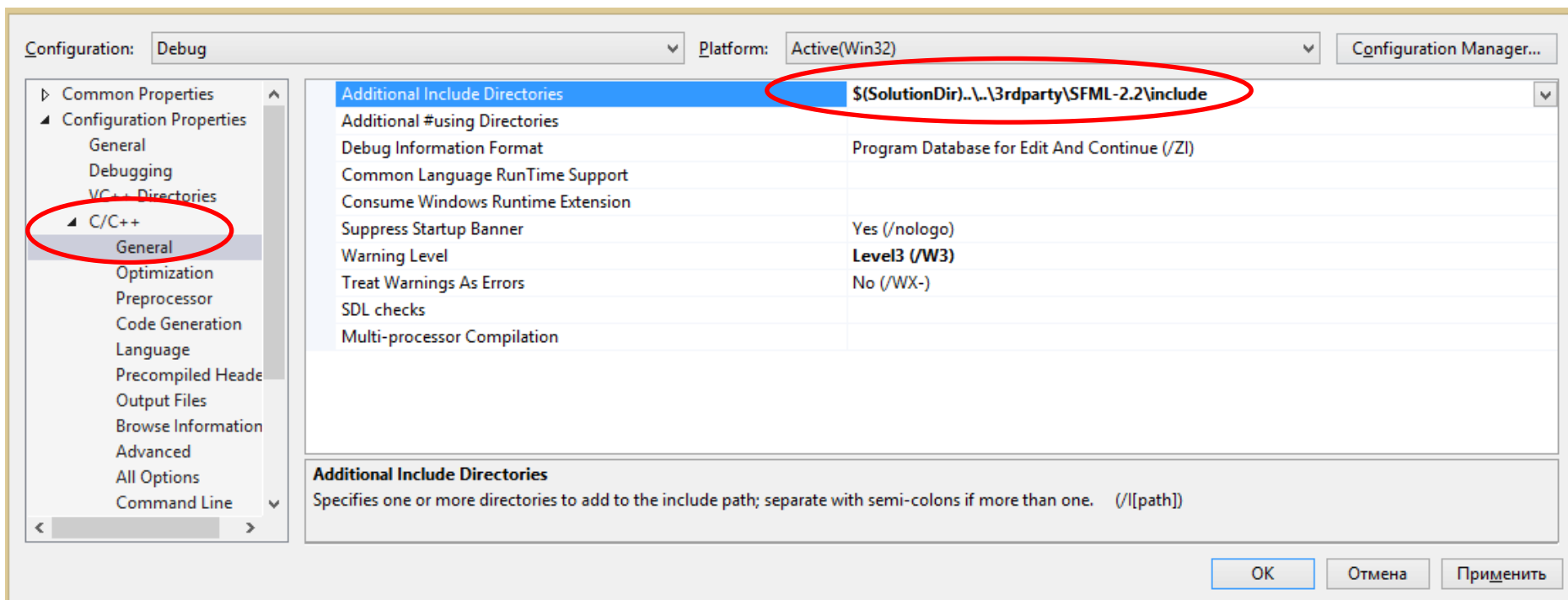
# Что такое библиотеки и какие они бывают?

**Библиотека** (от англ. *library*) — сборник подпрограмм или объектов, используемых для разработки программного обеспечения (ПО).

**Библиотеки уже скомпилированы -> в своей программе должны использовать такой же компилятор**

- Динамические (\*.dll – windows, \*.so – linux)  
Статические – (\*.lib – visual studio, \*.a – всё остальное в windows и в linux)

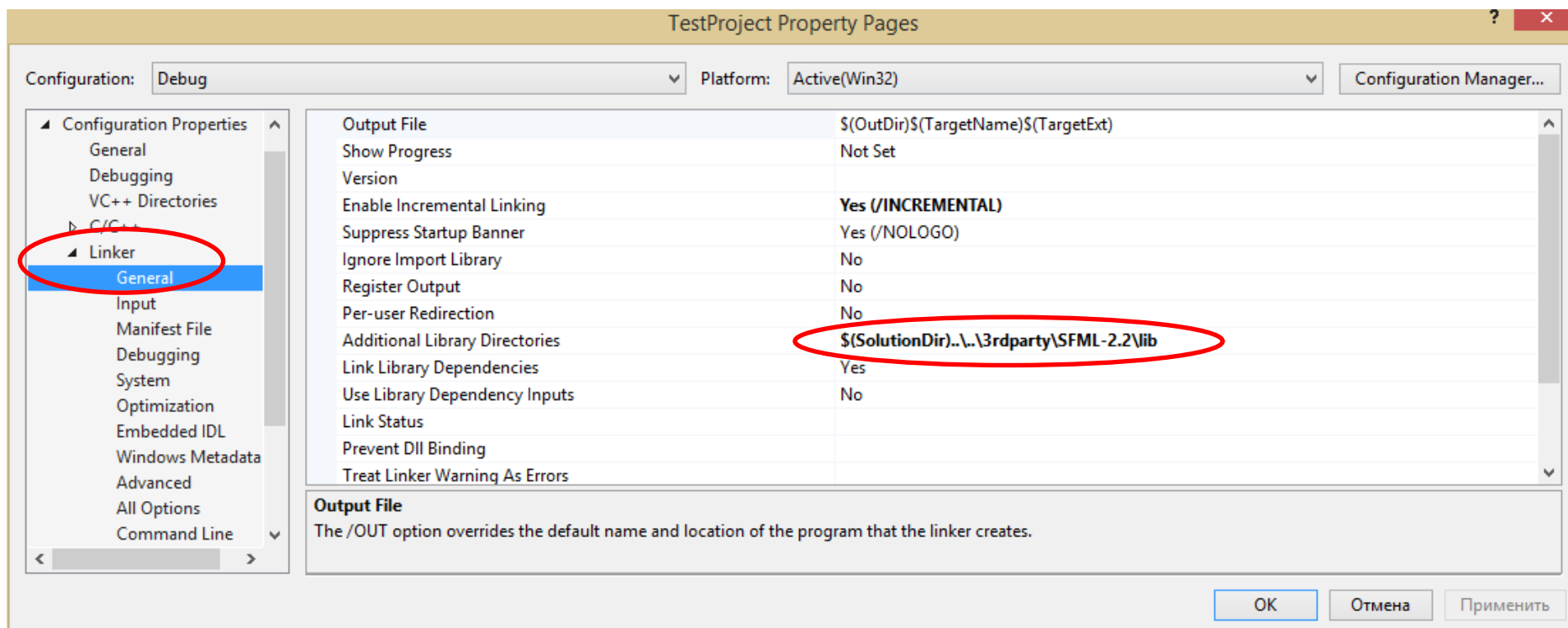
# Как «подключить» библиотеку к проекту?



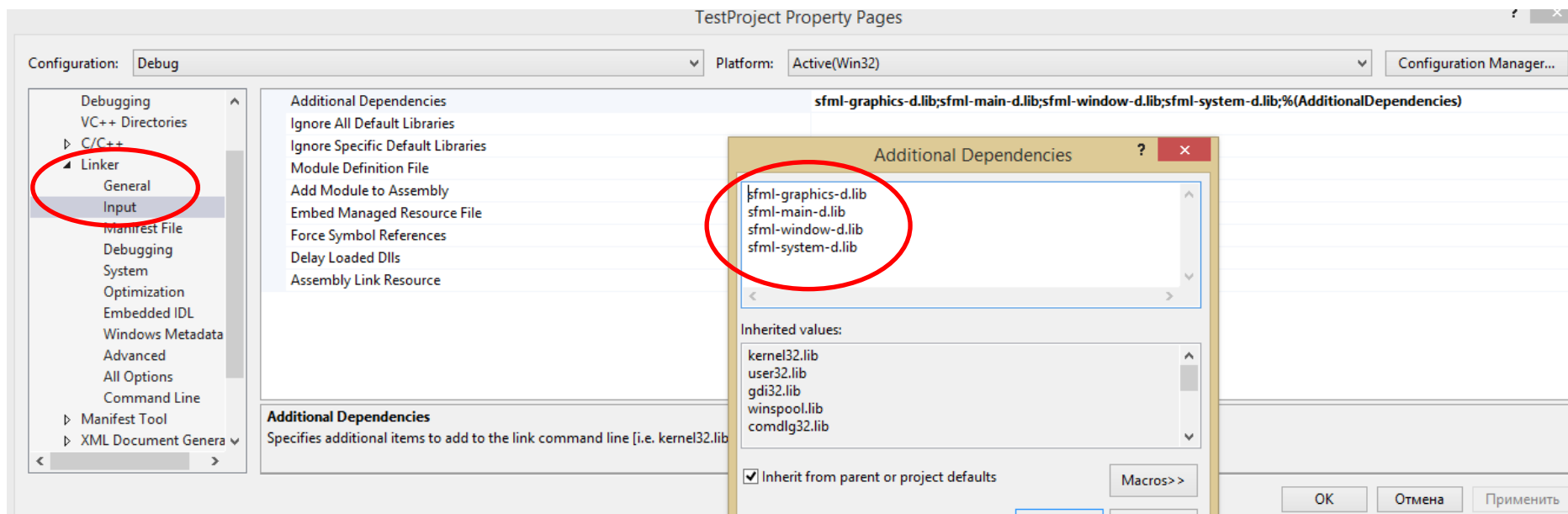
- Укажите актуальное название директории с библиотекой SFML



# Как «подключить» библиотеку к проекту?



# Как «подключить» библиотеку к проекту?



И последний шаг: скопировать все файлы \*.dll из 3rdparty/SFML-2.2/bin в /bin

# Пишем 1-ю программу...

<http://www.sfml-dev.org/tutorials/2.3/>

```
1  #include <SFML\Graphics.hpp>
2
3  int main()
4  {
5      sf::RenderWindow window(sf::VideoMode(800, 600), "My window");
6
7      while (window.isOpen())
8      {
9          window.clear(sf::Color::Red);
10         window.display();
11     }
12     return 0;
13 }
```

# Заккрытие окна

```
7  ..while (window.isOpen())
8  ..{
9  ..../* check all the window's events that were triggered
10 ....since the last iteration of the loop */
11 ....sf::Event event;
12 ....while (window.pollEvent(event))
13 ....{
14 ....// "close requested" event: we close the window
15 ....if (event.type == sf::Event::Closed)
16 ....    window.close();
17 ....}
18
19 ....window.clear(sf::Color::Red);
20 ....window.display();
21 ..}
```

# Рисуем геометрические примитивы

- <http://www.sfml-dev.org/tutorials/2.3/>

```
// circle
sf::CircleShape circle(50);
circle.setPosition(100, 100);

circle.setFillColor(sf::Color(150, 50, 250));
circle.setOutlineThickness(10);
circle.setOutlineColor(sf::Color(250, 150, 100));
window.draw(circle);

// line
sf::Vertex line[] =
{
    sf::Vertex(sf::Vector2f(10, 10)),
    sf::Vertex(sf::Vector2f(150, 150))
};
window.draw(line, 2, sf::Lines);

// rectangle
sf::RectangleShape rectangle(sf::Vector2f(120, 50));
rectangle.setSize(sf::Vector2f(100, 300));
rectangle.move(sf::Vector2f(400, 200));
window.draw(rectangle);
```

# Упражнение 1

- правильный N-угольник + вписанную и описанную окружности

$$r = \frac{a}{2 \operatorname{tg}\left(\frac{360^\circ}{2n}\right)}$$

- треугольник Серпинского

$$R = \frac{a}{2 \sin\left(\frac{360^\circ}{2n}\right)}$$

# Движение

```
sf::Clock clock;
```

```
while (window.isOpen())
```

```
{
```

```
    //sf::Time elapsed = clock.restart();
```

```
    sf::Time time = clock.getElapsedTime();
```

```
// circle
```

```
sf::CircleShape circle(50);
```

```
circle.setPosition(100 + 5 * time.asSeconds(), 100);
```

# Рисуем круг...

```
sf::CircleShape circle(50);  
while (window.isOpen())  
{  
    window.clear(sf::Color::Color(50, 50, 50));  
    window.draw(circle);  
    window.display();  
}
```

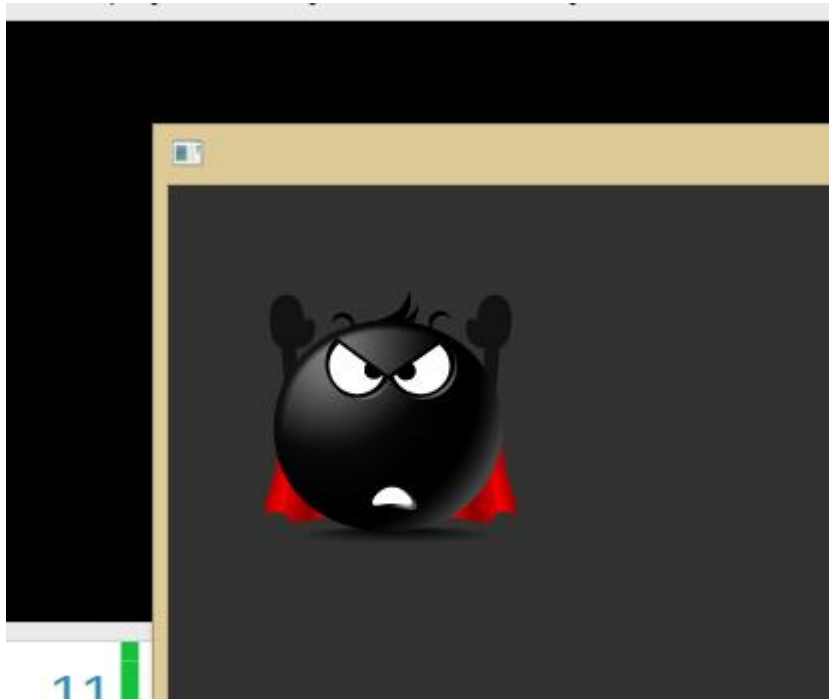


# И заставим его двигаться по нажатию клавиш...

```
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
{
    circle.move(-1, 0);
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
{
    circle.move(1, 0);
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
{
    circle.move(0, -1);
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
{
    circle.move(0, 1);
}
```

# На месте белого круга будем рисовать свою картинку (спрайт)

```
sf::Texture texture;  
texture.loadFromFile("smile.png");  
sf::Sprite circle(texture);
```



Много бесплатных иконок есть  
на [www.iconspedia.com](http://www.iconspedia.com)

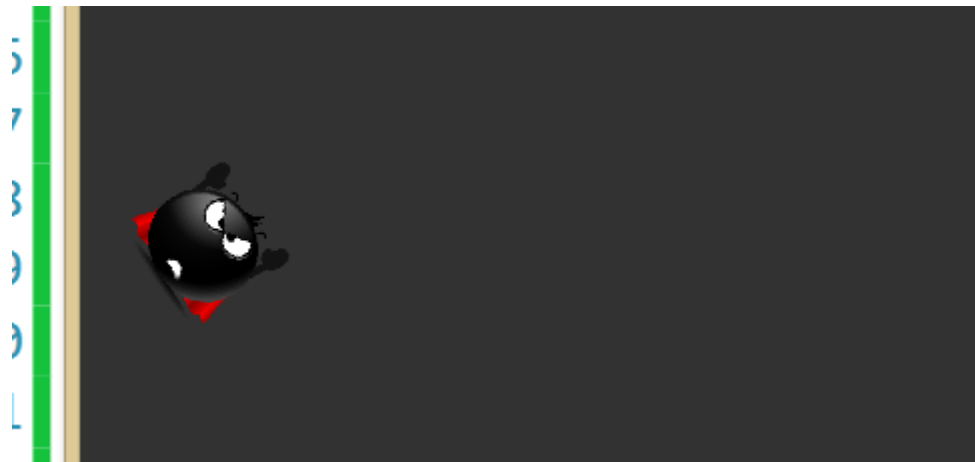
```
circle.setScale(0.5f, 0.5f);
```

# Обрабатывать события в общем случае можно так:

```
sf::Event event;
// while there are pending events...
while (window.pollEvent(event))
{
    // check the type of the event...
    switch (event.type)
    {
        // window closed
        case sf::Event::Closed:
            window.close();
            break;
        case sf::Event::KeyPressed:
        {
            if (event.key.code == sf::Keyboard::Left)
            {
                // do something
            }
        }
        // we don't process other types of events
        default:
            break;
    }
}
```

# Как сделать, чтобы персонаж был всегда ориентирован на курсор

```
sf::Vector2u circleSize = circle.getTexture()->getSize();  
circle.setOrigin(circleSize.x / 2, circleSize.y / 2);  
  
sf::Vector2i mousePosition = sf::Mouse::getPosition(window);  
sf::Vector2f center = circle.getPosition();  
sf::Vector2f d = sf::Vector2f(mousePosition.x, mousePosition.y) - center;  
  
const float Pi = 3.14159f;  
circle.setRotation(90 + atan2f(d.y, d.x) * 180 / Pi);
```



# Упражнение №2а

- герой не должен вылетать за экран;
- герой должен стрелять «лазерным лучом»
- герой должен стрелять отдельными патронами

```
if (sf::Mouse::isButtonPressed(sf::Mouse::Left))  
{  
    // left mouse button is pressed: shoot  
    // TODO  
}
```

## Упражнение №26 (альтернативное)

- Нарисовать линии поля  $E$  статических зарядов: уметь ставить заряд / проводить силовую линию через заданную точку.