Введение в современные компьютерные технологии

Практикум 2

Дрюк Андрей

Немнюгин Сергей Андреевич

Санкт-Петербургский государственный университет

2021

1. Переменная - ?
2. R-value ?
3. L-value - ?
4. Область видимости переменной - ?

1. Переменная - это одна или несколько физических ячеек оперативной памяти компьютера (именованная область памяти)
2. R-value - значение переменной
3. L-value – адрес переменной
4. Область видимости переменной - множество операторов программы, в которых данную переменную можно использовать.

Программой могут называть разные вещи.

а) исходный текст программы—обычный текстовый файл, содержащий запись операторов программы на языке программирования.

Исходный текст *компилируется* (транслируется), то есть переводится на язык машинных команд, понятный компьютеру. В этом случае создается *исполняемый файл.*

В более общем случае трансляция – перевод программы в представление на другом языке.

Виды трансляции:

Компиляция –преобразование в язык машинных команд, понятный процессору. В этом случае создается исполняемый файл.

Интерпретация –процесс чтения и исполнения исходного кода, выполняемый программой-интерпретатором.

Динамическая компиляция (JIT –Just In Time) –исходный или промежуточный код преобразуется в машинный код непосредственно во время исполнения.

Программа представляет собой последовательность операторов и других элементов языка, построенную в соответствии с определенными правилами и предназначенную для решения определенной задачи.

Правила, определяющие, какие последовательности символов можно использовать в программе, называются ***синтаксисом языка программирования***. Нарушение этих правил приводит к синтаксическим ошибкам. Эти ошибки выявляются на этапе трансляции программы.

Парадигма программирования -это совокупность идей и понятий, определяющих стиль написания программ (подход к программированию), определяющий организацию вычислений и структурирование работы, выполняемой компьютером.

Практически все современные языки являются ***мультипарадигменными***

Парадигмы языков высокого уровня:

1. императивная - процедурная;

2. функциональная;

3. логическая;

4.объектно-ориентированная

Императивное программирование – парадигма, описывающая процесс вычислений через последовательное изменение состояний.

Процедурное программирование – подход к программированию на императивном языке, заключающийся в объединении последовательно выполняемых операторов в более крупные структурные единицы –функции и процедуры.

Объектно-ориентированное программирование - парадигма, основанная на использовании объектов, которые могут содержать в себе данные («поля») и функции («методы»).

Основные понятия:

1. ***Класс***-универсальный, комплексный тип данных, состоящий из набора полей и методов.

```
class Figure {// создаем класс (описываем его поля и функции)
public:
    int x0 = 10; //поле int
    int y0 = 1; //поле int

    void print() { //метод
        std::cout << "x0 = " << x0 << " y0 = " << y0 <<
std::endl;
    }

};
```

2.***Объект***-сущность в адресном пространстве, появляющаяся при создании экземпляра класса.

```
int main()
{
    Figure fig;// создаем объект  (экземпляр
класса)
    fig.print(); //Вызываем метод объекта
}
```

Вывод:

x0 = 10 y0 = 1

3.***Инкапсуляция***-свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе.

4. ***Наследование***-свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью.

5. ***Полиморфизм***-свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

```
class Figure {
public:
    int x0 = 10;
    int y0 = 1;

    void print() {//тело метода}

};
```

```
class Circle : public Figure {

public:
    int radius = 5;
    void print() {
        std::cout <<"x0 = "<<x0 << " radius " <<
radius << std::endl;
    }
};
```

Функциональное программирование -парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании этого слова.

Логическое программирование - парадигма программирования, основанная на автоматическом доказательстве теорем.

Средства разработки программного обеспечения

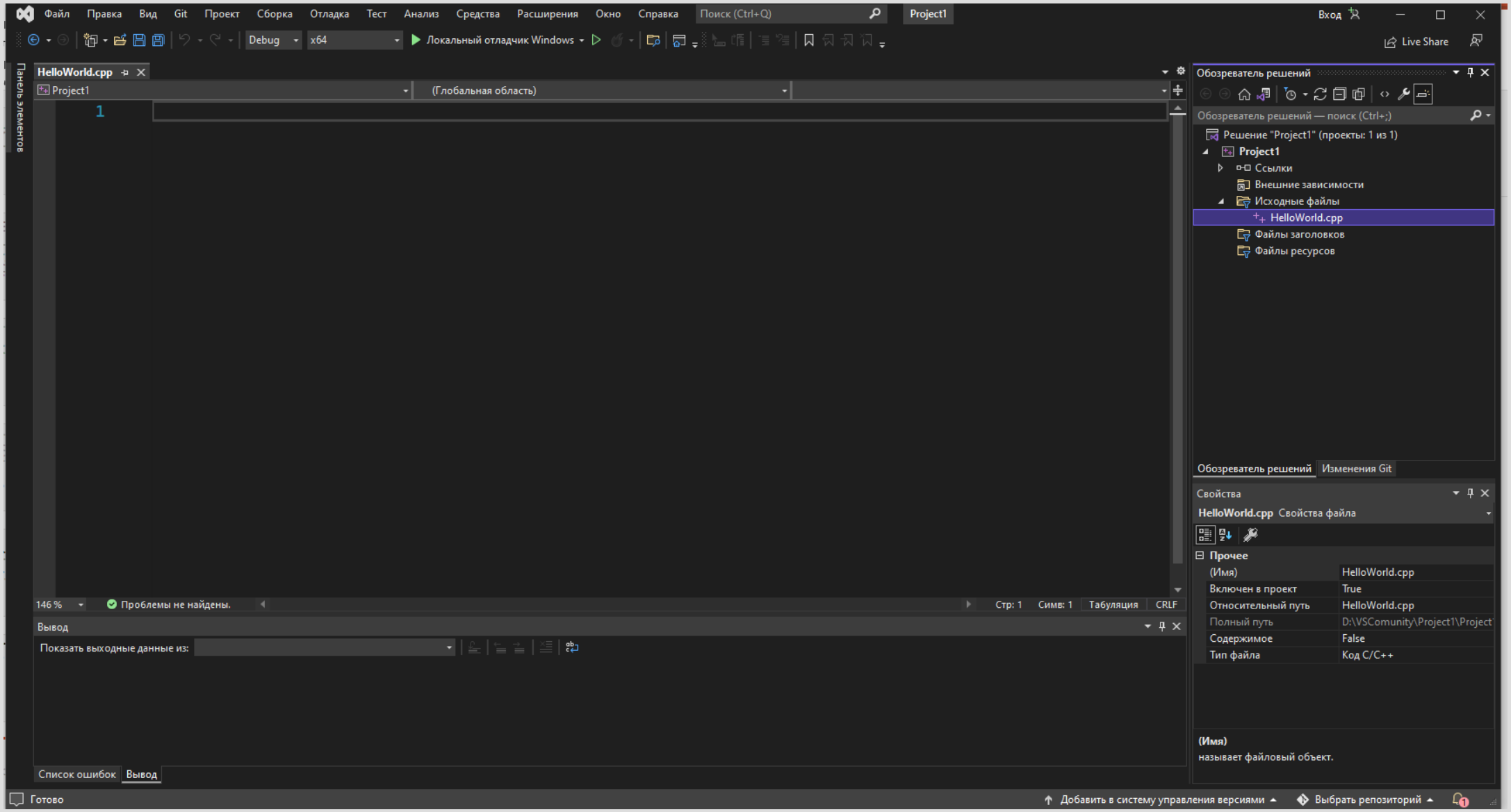
1. Microsoft Visual Studio Community
2. Visual Studio Code (Linux)
3. Qt - кроссплатформенный фреймворк для разработки программного обеспечения на языке программирования C++.
4. Eclipse, CLion и т д

Практика

Visual Studio Community

1. Создать проект->Пустой проект(консольное приложение) C++ ->
   Имя проекта
2. В обозревателе решений выбрать «исходные файлы» -
   >Добавить->Создать Элемент->Файл C++, и выбрать название

Практика


Файл Правка Вид Git Проект Сборка Отладка Тест Анализ Средства Расширения Окно Справка
Поиск (Ctrl+Q)
Project1
Вход
Live Share
Debug
x64
Локальный отладчик Windows
Панель элементов
HelloWorld.cpp
Project1
(Глобальная область)
1
Обозреватель решений
Обозреватель решений — поиск (Ctrl+;)
Решение "Project1" (проекты: 1 из 1)
Project1
Ссылки
Внешние зависимости
Исходные файлы
HelloWorld.cpp
Файлы заголовков
Файлы ресурсов
Обозреватель решений
Изменения Git
Свойства
HelloWorld.cpp Свойства файла
Прочее
(Имя) HelloWorld.cpp
Включен в проект True
Относительный путь HelloWorld.cpp
Полный путь D:\VSComunity\Project1\Project
Содержимое False
Тип файла Код C/C++
(Имя)
называет файловый объект.
146 %
Проблемы не найдены.
Стр: 1
Симв: 1
Табуляция
CRLF
Вывод
Показать выходные данные из:
Список ошибок
Вывод
Готово
Добавить в систему управления версиями
Выбрать репозиторий

Структура программ

Практика

```
#include <iostream>
/*Подключение библиотеки iostream,
Библиотека - файл, с описанием различных функций
Теперь можем использовать функции из это библиотеки*/


int main() {
//setlocale(LC_ALL, "Rus") // Для русских символов при выводе
std::cout << "Hello World" << std::endl;
}
```

Структура и порядок выполнения программ

```
#include <iostream>
/*Подключение библиотеки iostream,
Библиотека - файл, с описанием различных функций
Теперь можем использовать функции из этой библиотеки*/

using namespace std;
/*Используем пространство имен std
Пространство имен - определяет некую область,
на которую приходится действие оператора
*/

int main() {
//setlocale(LC_ALL, "Rus") // Для русских символов при выводе
cout << "Hello World" << endl;
}
```

Структура и порядок выполнения программ

1. `#include <iostream>` – добавление/подключение необходимых библиотек
`<iostream>` – заголовочный файл с классами, функциями и переменными для организации ввода-вывода в языке программирования C++

2. `using namespace std;` – использование определенного пространства имен
Иначе `std::cout << "Hello World" << std::endl;`

3. `int main(){//тело функции}` – точка входа в программу!!!

Директива препроцессора #include

1. #include <iostream> – добавление/подключение необходимых библиотек

<iostream> – заголовочный файл с классами, функциями и переменными для организации ввода-вывода в языке программирования C++

< > – поиск в стандартной директории с именем include

“ ” – поиск файла в текущей директории

2. Директивы препроцессора являются указаниями компилятору, а не компьютеру

Пространство имен и директива using

1. Пространство имен – область программы, в которой распознается определенная совокупность имен

2. Пространства имен используются для организации кода в виде логических групп и с целью избежать конфликтов имен, которые могут возникнуть, особенно в таких случаях, когда база кода включает несколько библиотек.

3. Все идентификаторы в пределах пространства имен доступны друг другу без уточнения.

4. Для получения доступа к идентификатору за пределами пространства имен необходимо:

а) Использовать полное имя идентификатора (включающее пространство имен): `std::cout;`

б) Использовать директиву using для отдельного идентификатора: `using std::cout;`

с) Использовать директиву using для всех идентификаторов в пространстве имен: `using namespace std;`

Функция main()

```
int main() //Заголовок функции
{  // начало функции
 //тело функции (переменные,операторы)
return 0; //оператор возврата
}
```

1. `int` – тип возвращаемого значения
2. `()` – функция не принимает информации от функции, которая произвела обращение к данной функции. В C++(не в C) принято, что пустые скобки равносильны использованию ключевого слова void. Возможен вариант void main (void) –функция не возвращает значения.
3. `return 0;` возвращает тип int

Типы данных и переменные

Основные типы переменных:

- Целочисленные:

Знаковые	*Беззнаковые*
	bool
char	*char**
int	*unsigned int*
short	*unsigned short*
long	*unsigned long*
long long	*unsigned long long*

- С плавающей точкой:
  - С одинарной точностью: *float*
  - С двойной точностью: *double*
  - С расширенной точностью: *long double*†
- Указатель: см. позже
- *void*: см. позже

---

*При компиляции со специальной опцией

†Определяется как 'larger than or equal to type double'. В стандартных реализациях оказывается равен типу double.

Типы данных и переменные

Размеры переменных:

- Microsoft

Тип	**Размер**
bool, char	1 байт
short	2 байта
float, int, *long*	4 байта
double, *long long*	8 байт

- Размер указателя зависит от разрядности архитектуры

Типы данных и переменные

Размеры переменных:

- Unix, 32-bit

Тип	**Размер**
bool, char	1 байт
short	2 байта
float, int, *long*	4 байта
double	8 байт

- Unix, 64-bit

Тип	**Размер**
bool, char	1 байт
short	2 байта
float, int	4 байта
double, *long*	8 байт

- Размер указателя зависит от разрядности архитектуры

Типы данных и переменные

```
имя_типа имя_переменной

int i;
double x;
string s;

Регистрочувствительный язык: int i, int I
– разные переменные
```

Операторы арифметические C++

Операция	Оператор	Синтаксис
Присваивание	=	a = b
Сложение	+	a + b
Вычитание	-	a - b
Унарный плюс	+	+a
Унарный минус	-	-a
Умножение	*	a * b
Деление	/	a / b
Операция модуль (остаток от деления)	%	a % b
Префиксный инкремент	++	++a
Постфиксный инкремент	++	a++
Префиксный декремент	−−	−−a
Постфиксный декремент	−−	a−−

Операторы сравнения C++

Операция	Оператор	Синтаксис
Равенство	==	a == b
Неравенство	!=	a != b
Больше	>	a > b
Меньше	<	a < b
Больше или равно	>=	a >= b
Меньше или равно	<=	a <= b

Операторы C++

Операция	Оператор	Синтаксис
Логическое отрицание, НЕ	!	!a
Логическое умножение, И	&&	a && b
Логическое сложение, ИЛИ	||	a || b
Побитовая инверсия	~	~a
Побитовое И	&	a & b
Побитовое ИЛИ (or)	|	a | b
Побитовое исключающее ИЛИ (xor)	∧	a ∧ b
Побитовый сдвиг влево	<<	a << b
Побитовый сдвиг вправо	>>	a >> b

Операторы C++

Операция	Синтаксис
Обращение к элементу массива	a[b]
Непрямое обращение	*a
Ссылка (“адрес a”)	&a
Обращение к члену структуры	a.b
Обращение к члену структуры, доступной по ссылке	a->b
Применение функции	a(a1, a2)
Тернарная условная операция	a ? b : c
Оператор расширения области видимости	a::b
Sizeof (размер)	**sizeof**(a)
Приведение типа	(type) a
Выделение памяти	**new** type
Выделение памяти для массива	**new** type[n]
Освобождение памяти	**delete** a
Освобождение памяти, занятой массивом	**delete[]** a

Операторы инкремента и декремента (++/--)

Постфиксный

```
int a = 0;
a++;
cout<<"1. a after a++ = " << a << endl;
cout<<"2. a++ = " << a++ << endl;
cout<<"3. a after a++ = " << a << endl;
```

Префиксный

```
int a = 0;
++a;
cout<<"1. a after ++a = " << a << endl;
cout<<"2. ++a = " << ++a << endl;
cout<<"3. a after ++a = " << a << endl;
```

Операторы инкремента и декремента (++/--)

Постфиксный

```
int a = 0;
a++;
cout<<"1. a after a++ = " << a << endl;
cout<<"2. a++ = " << a++ << endl;
cout<<"3. a after a++ = " << a << endl;
```

1. a after a++ = 1
2. a++ = 1
3. a after a++ = 2

Префиксный

```
int a = 0;
++a;
cout<<"1. a after ++a = " << a << endl;
cout<<"2. ++a = " << ++a << endl;
cout<<"3. a after ++a = " << a << endl;
```

1. a after ++a = 1
2. ++a = 2
3. a after ++a = 2

Операторы ветвления в C++

```
if(Условие){
  Действие 1;
}else{
  Действие 2;
}
```

```
if (Условие) {
    Действие 1;
}else  if (Условие 2){
    Действие 2;
}else if (условие 3){
    Действие 3;
}else{….
}
```

Операторы ветвления в C++

```
switch (переменная){
   case значение1:
      тело case 1;
      break ;
   case значение2:
      тело case 2;
   case значение3:
      тело case 3; // Выполнится и при значении 2
      break;
   case default:
      тело default; // Выполнится в остальных случаях
      break;
}
```

Операторы ветвления в C++

Тернарный оператор ?

a > b ? cout << a : cout << b;

// если a > b, то выполняется cout << a, иначе выполняется cout << b

Циклы в C++

Цикл for

for (объявление переменных; условие; изменение счетчика){
  тело цикла;
}

```
for (int i = 0; i < 10; ++i) {
    cout << i << endl;
}
```

Задание по циклу for

Возможные варианты вычисления:

- Прямой подсчет: $\pi = \int_{-1}^{1} \frac{dx}{\sqrt{1-x^2}}$
- Через обобщенные дроби: $\pi = \cfrac{4}{1+\cfrac{1^2}{3+\cfrac{2^2}{5+\cfrac{3^2}{7+\cfrac{4^2}{9+\cdots}}}}}$
- Разложение в ряд
  - Формула Виета (1593): $\frac{2}{\pi} = \frac{\sqrt{2}}{2} + \frac{\sqrt{2+\sqrt{2}}}{2} + \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} + \cdots$
  - Формула Виллиса (1655): $\frac{\pi}{2} = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdots$
  - Ряд Грегори-Лейбница (1671): $\frac{\pi}{4} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \cdots$
  - Ряд Нилаканта (XV century): $\frac{\pi}{4} = \frac{3}{4} + \frac{1}{2 \cdot 3 \cdot 4} - \frac{1}{4 \cdot 5 \cdot 6} + \frac{1}{6 \cdot 7 \cdot 8} - \frac{1}{8 \cdot 9 \cdot 10} \cdots$
- Методами Монте-Карло

Задание по циклу for

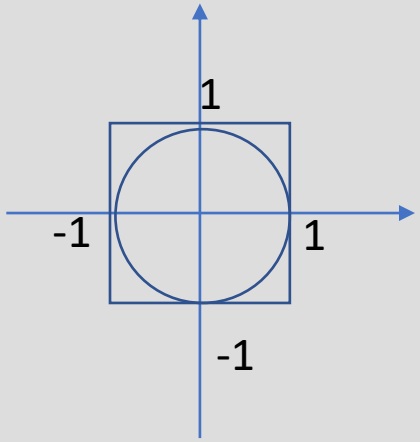
Вычислим число π используя ряд Грегори-Лейбница:

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} \cdots$$

Задание по циклу for

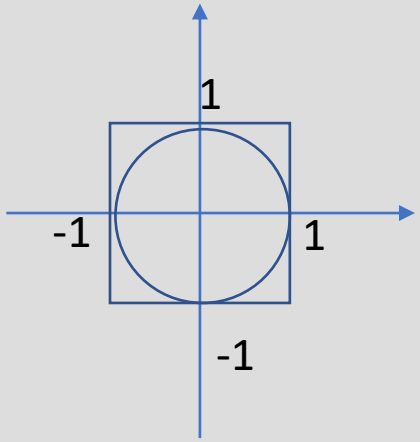
```
double pi = 0;
int sign = 0;
for (int i = 0; i < 10; ++i) {
   if ((i % 2)==0) {
      sign = 1;
   }
   else {
      sign = -1;
   }
   pi +=sign*4 / (1+2*i);
}
cout << fixed;// Формат плавающей точки (иначе значение разряды )
cout.precision(10); // число знаков после запятой
cout<<pi << endl;

return 0;
```

Домашнее задание. Найти pi методом Монте Карло



1. Генерируем два случайных числа (double x, double y) в диапазоне от -1, 1 из равномерного распределения
2. Вероятность попасть в квадрат 100%
3. Вероятность попасть в круг пропорциональна площади круга (чем она меньше, тем меньше вероятность попасть)
4. Отношение площадей $\frac{S_{\text{круг}}}{S_{\text{кв}}} = \frac{\text{Вероятность попасть в круг}}{\text{Вероятность попасть в квдарат (100\%)}} \approx \frac{N_{\text{круг}}}{N_{\text{кв}}} \approx \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$
5. Находим π

Домашнее задание. Найти pi методом Монте Карло



1. Подключить нужную библиотеку
2. Функция rand() - генерирует от 0 до RAND_MAX (тип int!!!)
3. Привести к диапазону от [-1, 1]. При приведении умножаем на 1.0, чтобы был double
4. Сгенерировать какое-то количество точек и посчитать, сколько точек будет в круге
5. Найти π

Частые ошибки

1. После операторов в c++ ставятся **точки с запятой** (компилятор должен указать, если забыли)
2. Переменные, которые планируется использовать по ходу программы, **определяем вне циклов и блоков if**
3. **Результат деления целочисленного числа на целочисленное –> целое** число (3/5 = 0). Там, где подразумеваются числа с плавающей точкой, пишем 3.0/5