Мы научим программированию

… но надо будет трудиться и самим ☹️

Зачем программирование вообще нужно?

- Нас окружают электронные устройства: компьютеры, смартфоны и. т.д.
- Все эти устройства «как живые» - реагируют на наши действия и действия других людей (человек послал нам сообщение в мессенджере – наш смартфон просигнализировал нам об этом)
- Любое «самостоятельное поведение» электронного устройства описывается программистами
- За каждой программой или сайтом – стоит один или много программистов
- Операционные системы – также пишутся программистами
- Без программ – все электронные устройства – это кирпичи

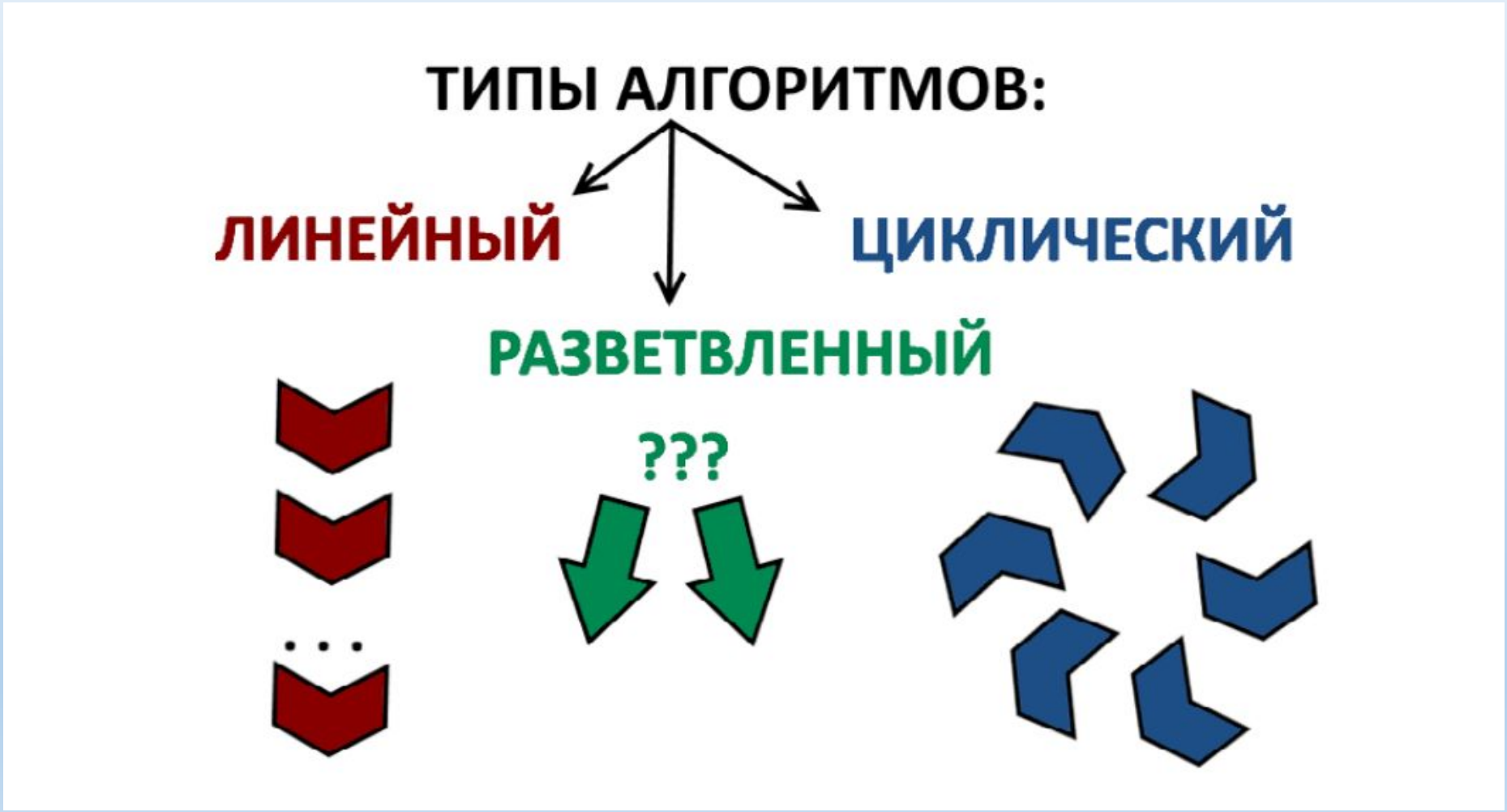
Любое поведение = последовательность действий

… то есть алгоритм!

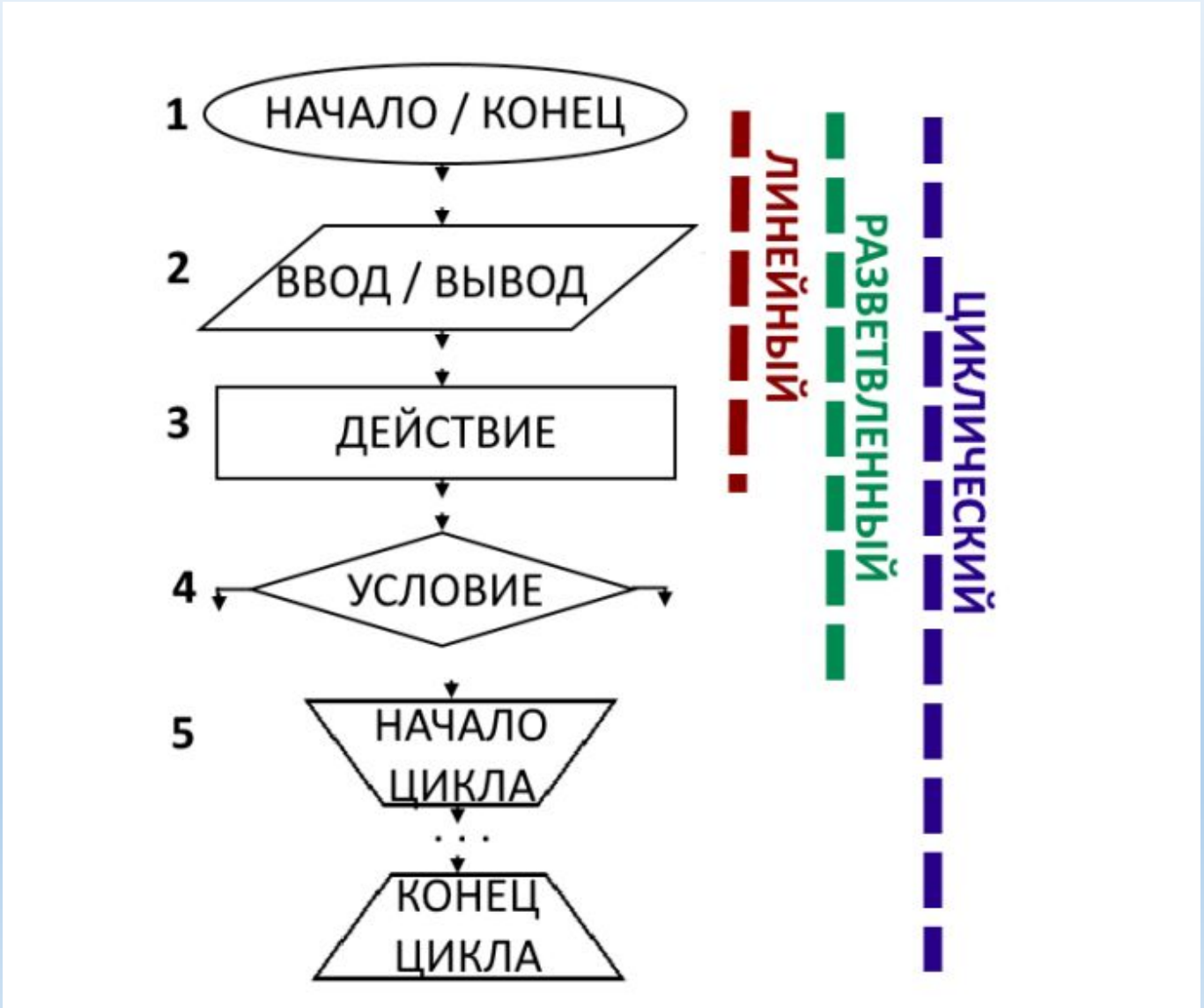
Какой может быть алгоритм? Как его можно охарактеризовать?

- Результативность. (Должен иметь цель)
- Корректность (Что получается после выполнения? Правильный ли результат)
- Точность (Не должно быть разночтений)
- Понятность (Должен быть понятен исполнителю)
- Дискретность. Все шаги должны быть расшифрованы в алгоритме
- Массовость. Можно ли алгоритм его применять для похожих задач?

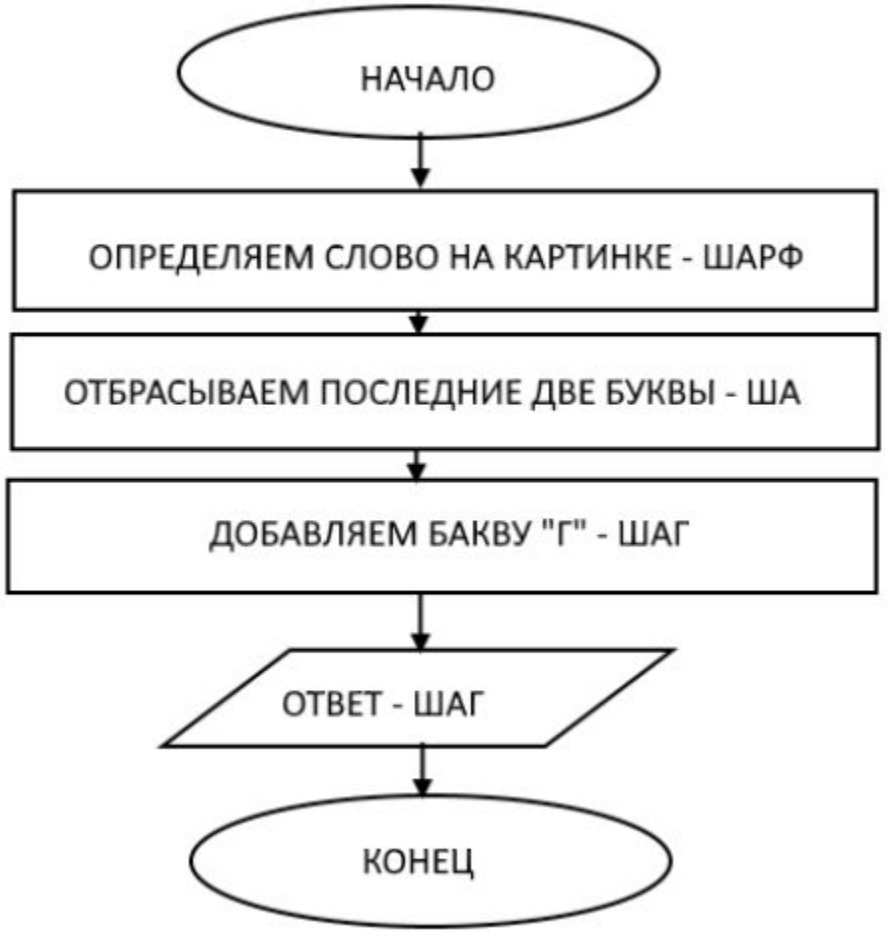
Жизнь переменчивая штука, переменчивы и алгоритмы.

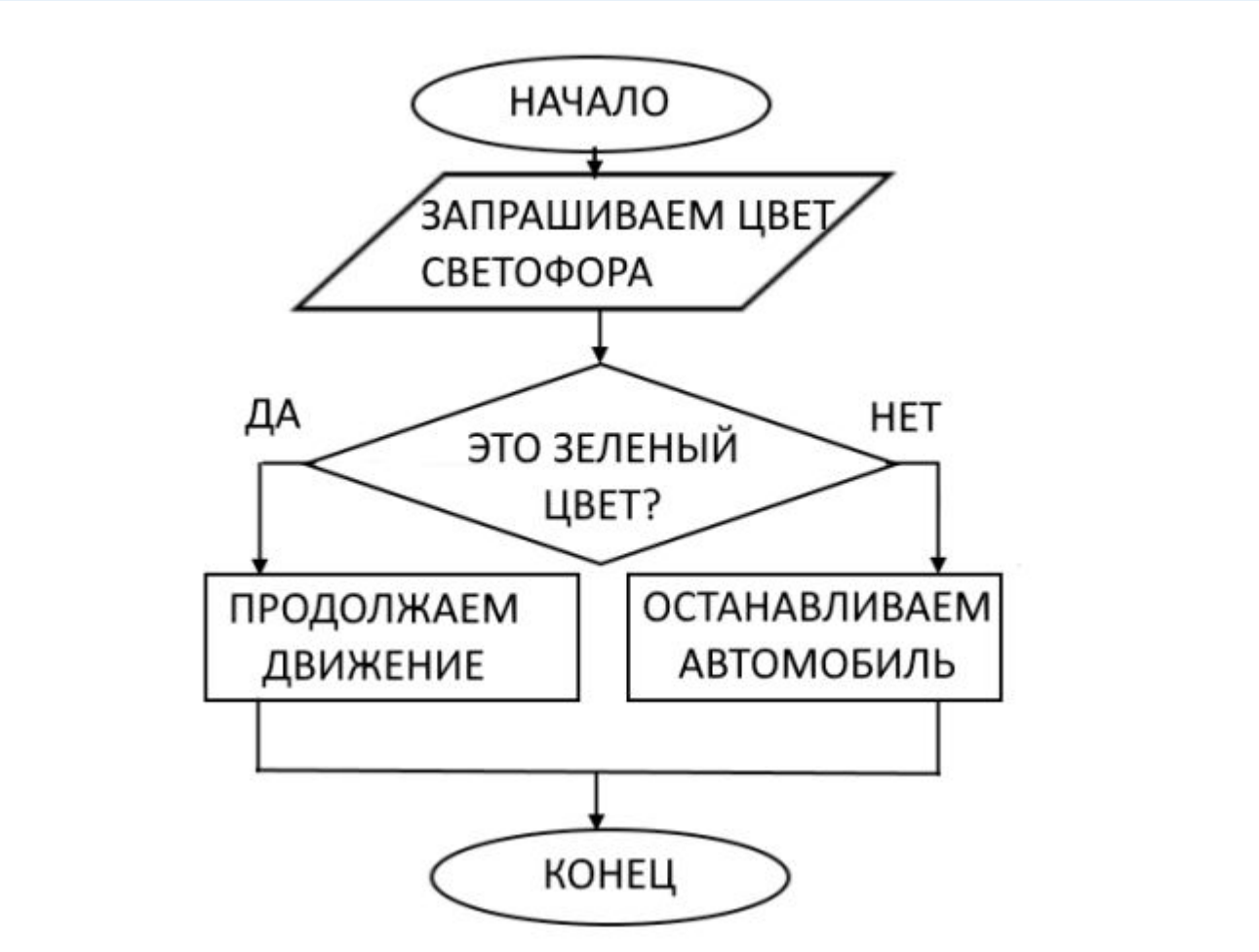


Блок-схема - графический способ описания алгоритмов



,, +Г

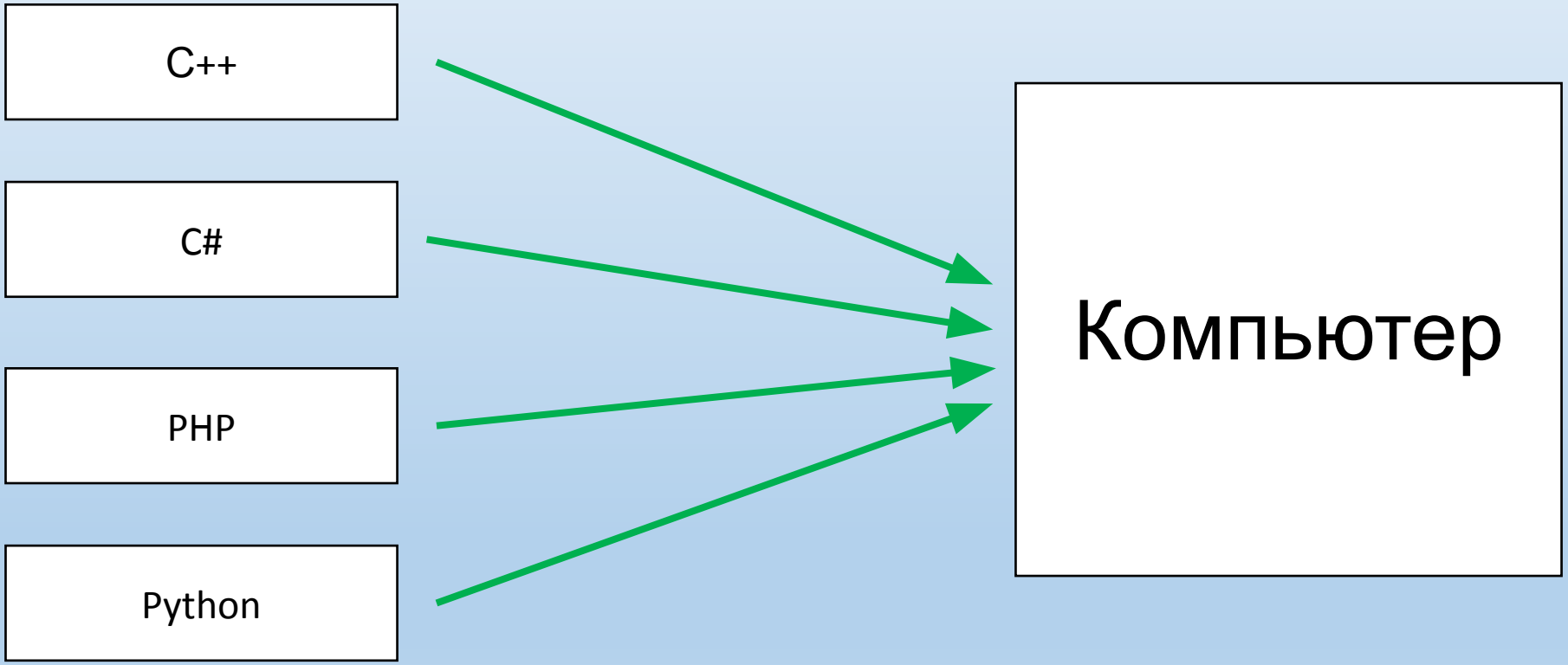

НАЧАЛО
ОПРЕДЕЛЯЕМ СЛОВО НА КАРТИНКЕ - ШАРФ
ОТБРАСЫВАЕМ ПОСЛЕДНИЕ ДВЕ БУКВЫ - ША
ДОБАВЛЯЕМ БАКВУ "Г" - ШАГ
ОТВЕТ - ШАГ
КОНЕЦ


НАЧАЛО
ЗАПРАШИВАЕМ ЦВЕТ СВЕТОФОРА
ДА
ЭТО ЗЕЛЕНЫЙ ЦВЕТ?
НЕТ
ПРОДОЛЖАЕМ ДВИЖЕНИЕ
ОСТАНАВЛИВАЕМ АВТОМОБИЛЬ
КОНЕЦ

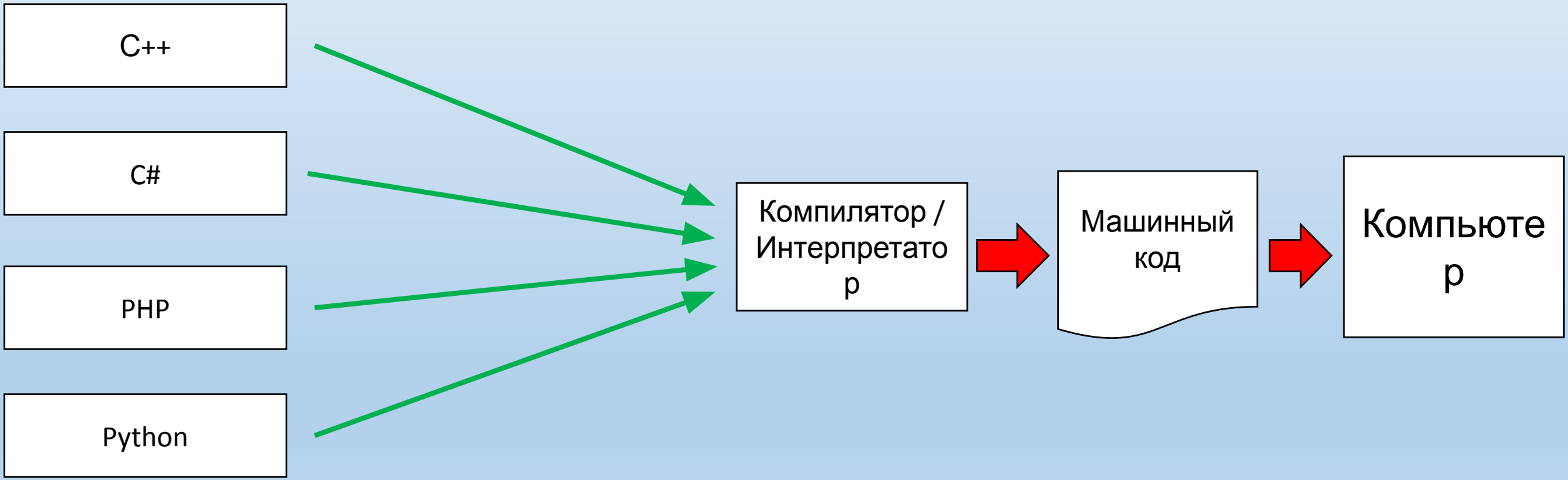
Любая программа – это реализация алгоритма

Поэтому блок-схемы используют для описания программ

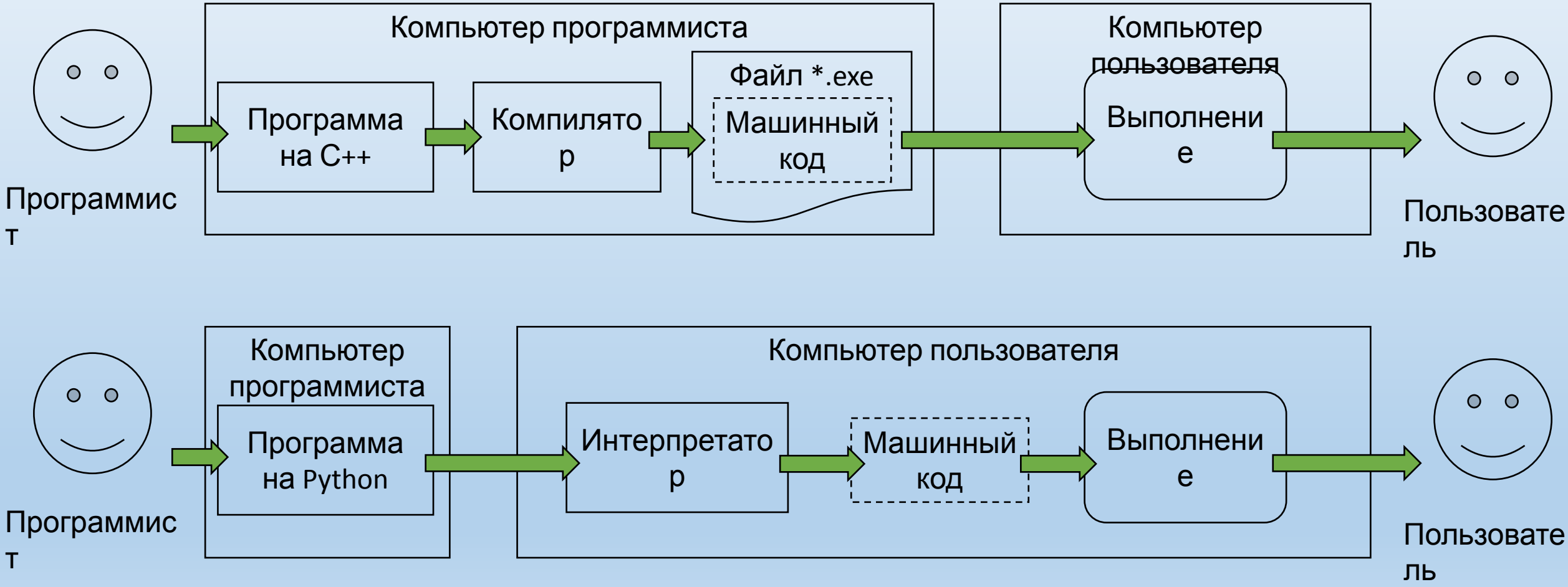
Куча языков – но компьютер один. Это как?
Он полиглот?



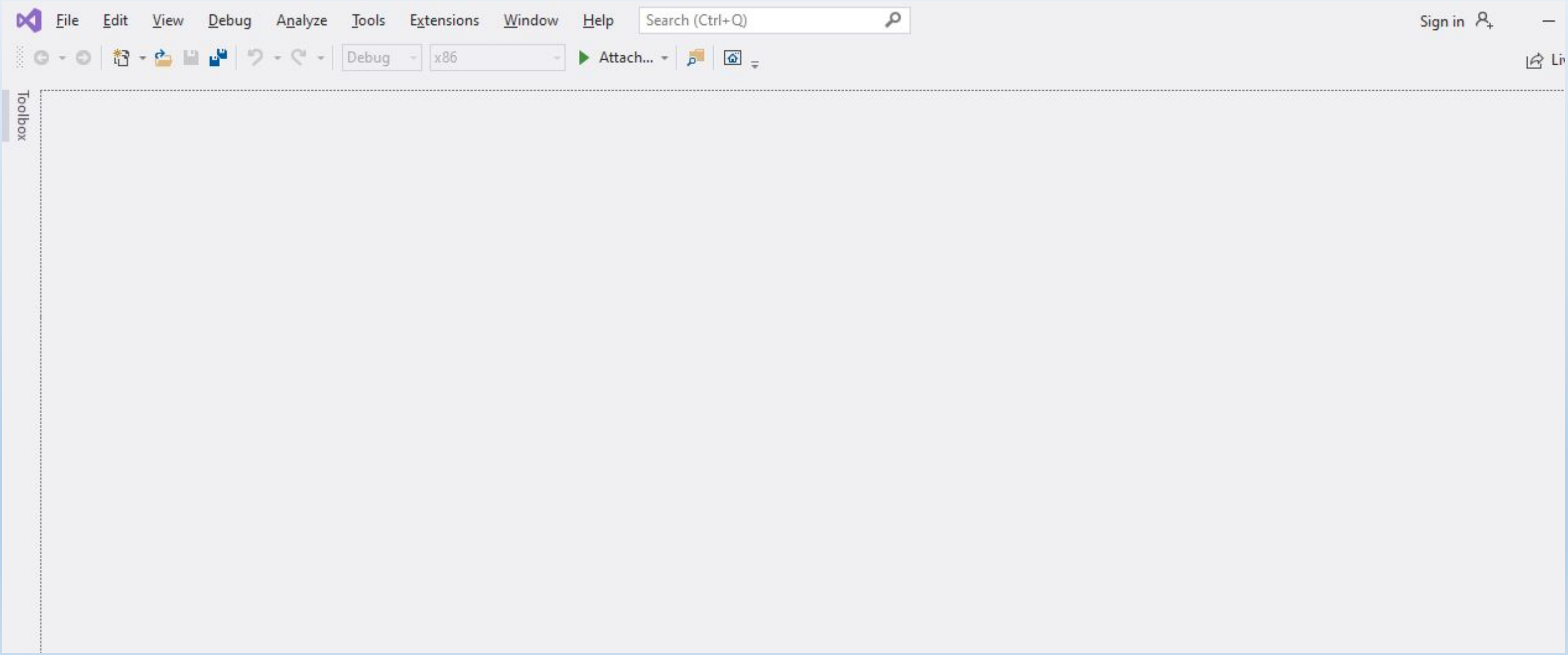
Нет, он не полиглот



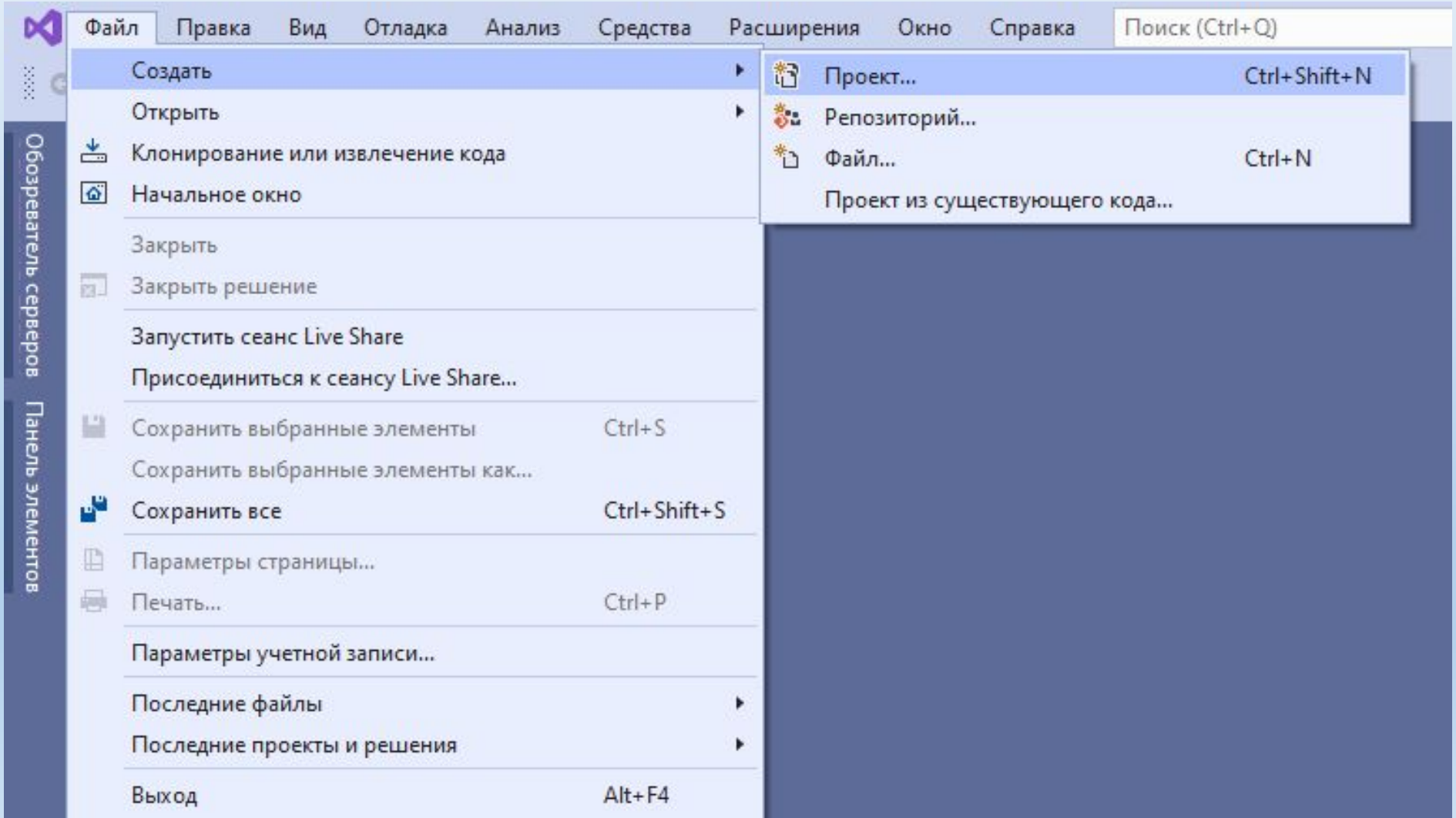
Интерпретатор vs Компилятор



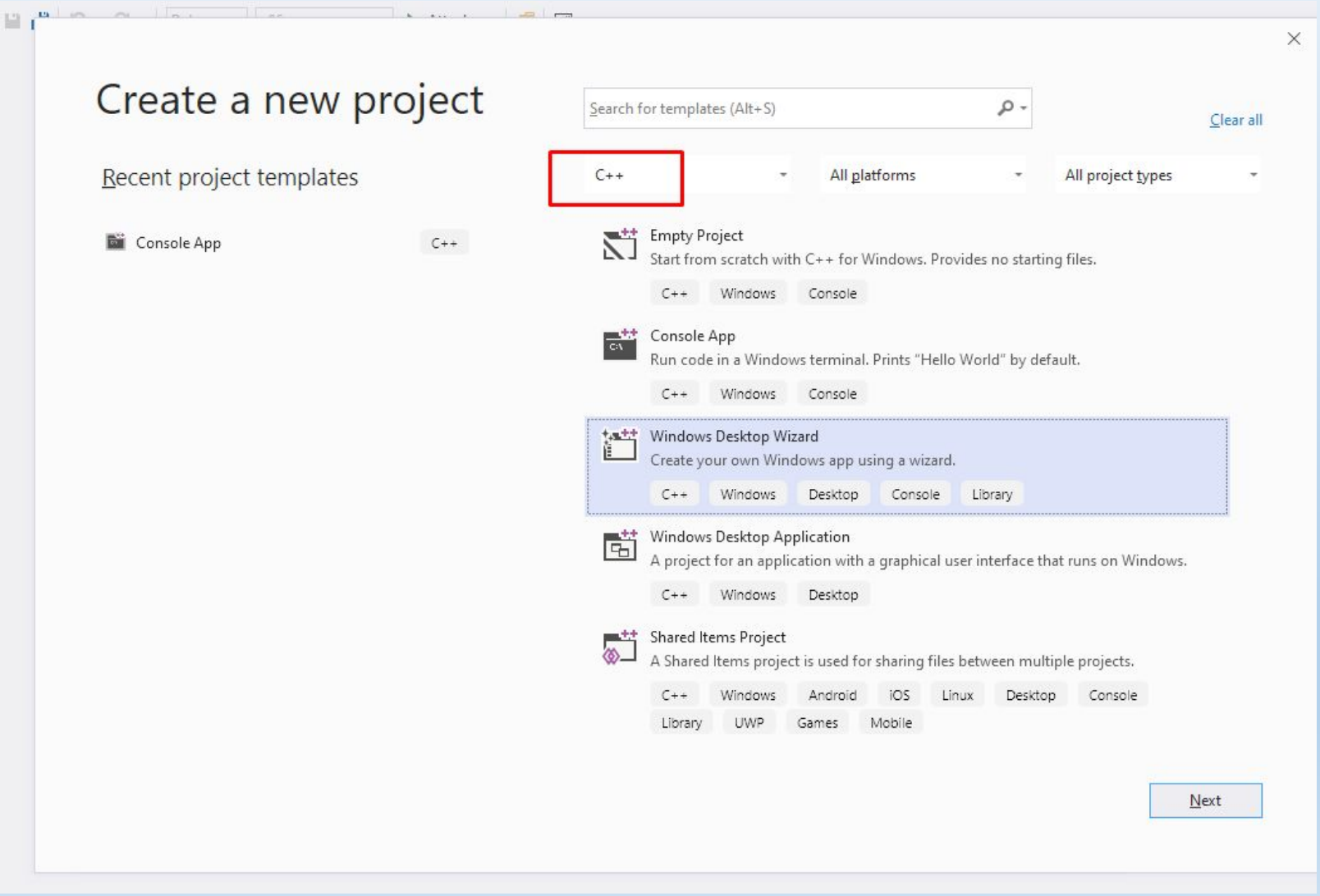
Создание нового проекта



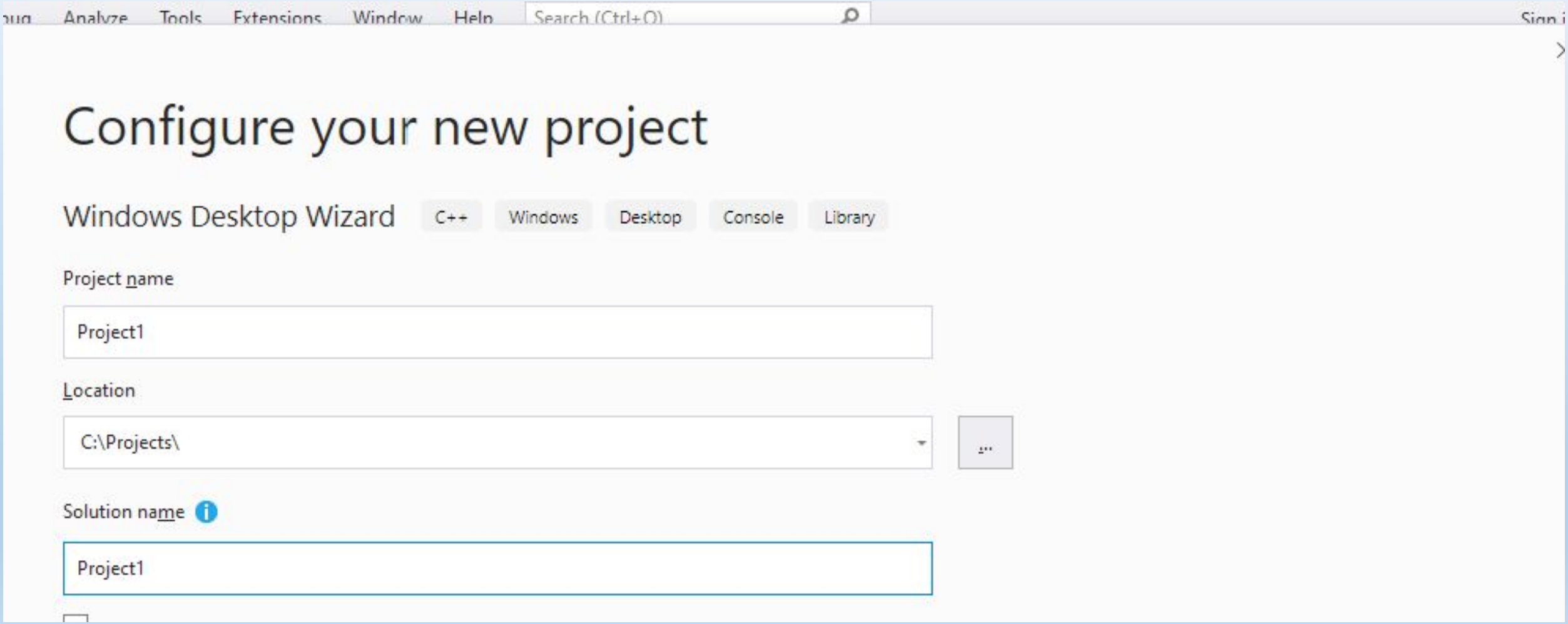
Создание нового проекта



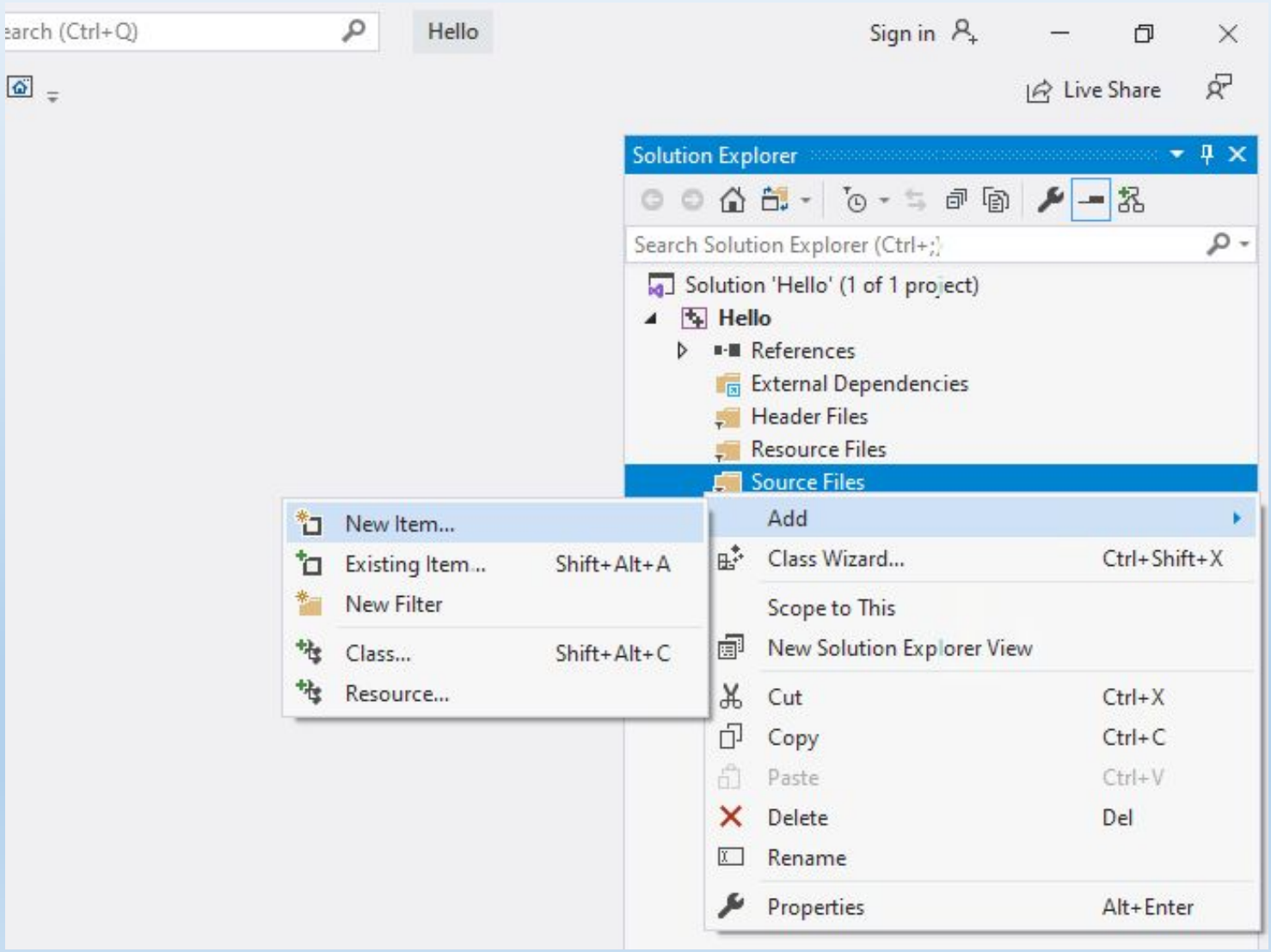
Создание нового проекта



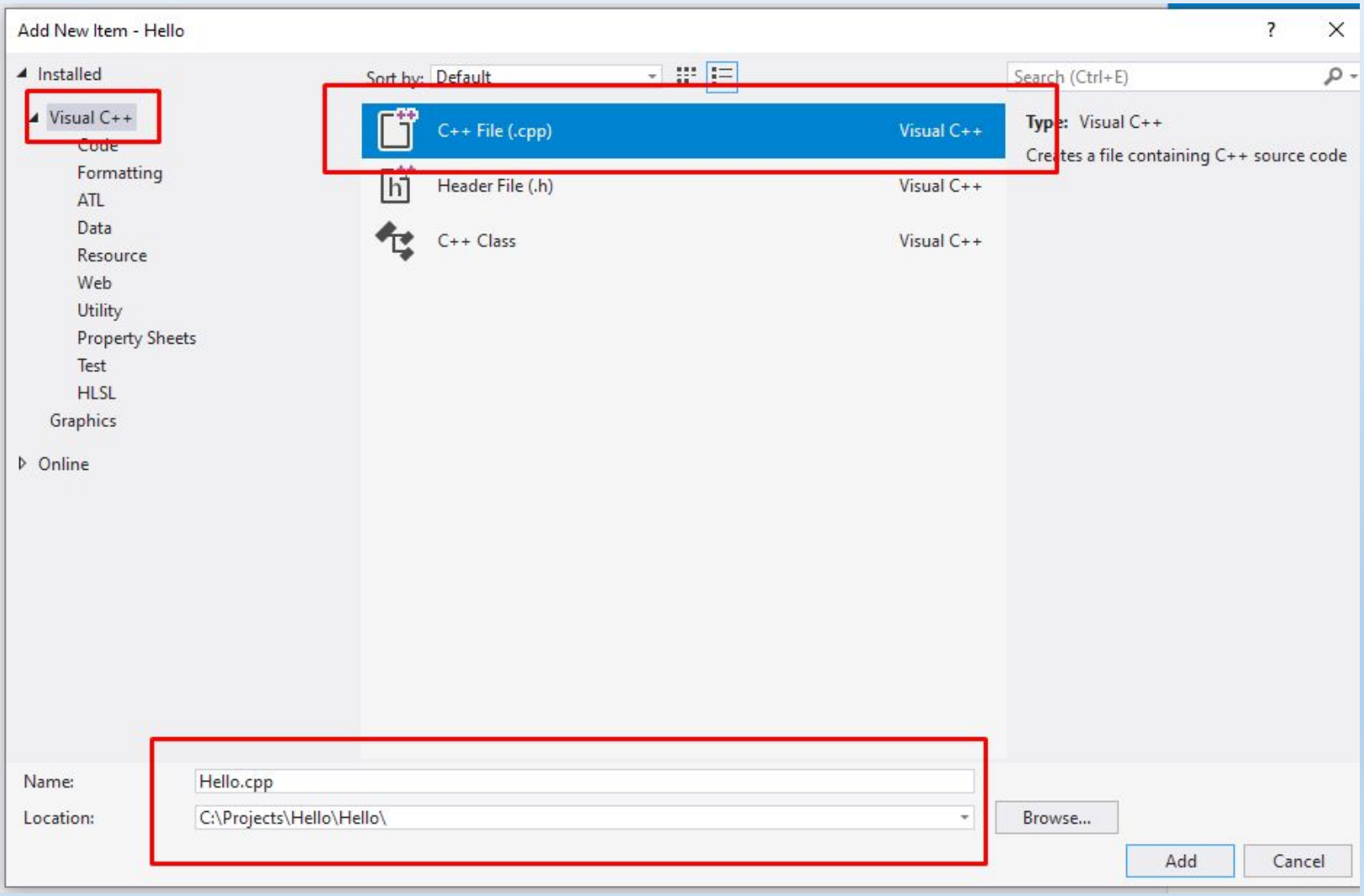
Создание нового проекта



Создание нового проекта



Создание нового проекта

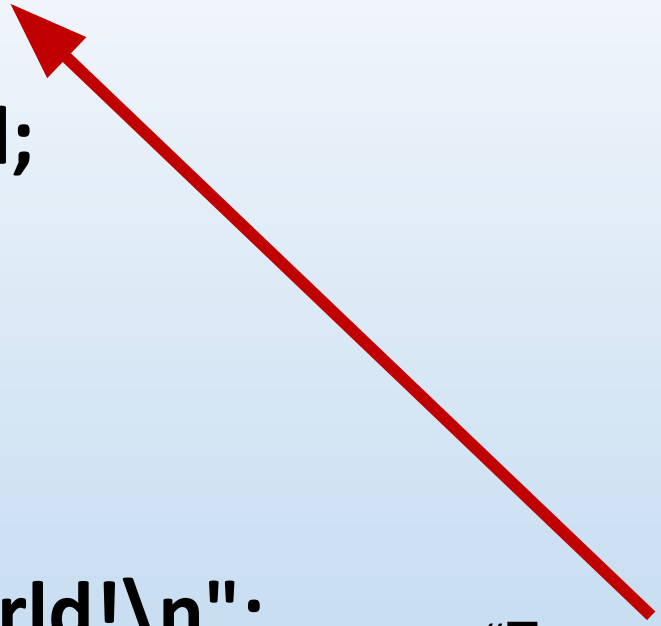


Первая программа

В открывшемся файле нужно набрать следующее:

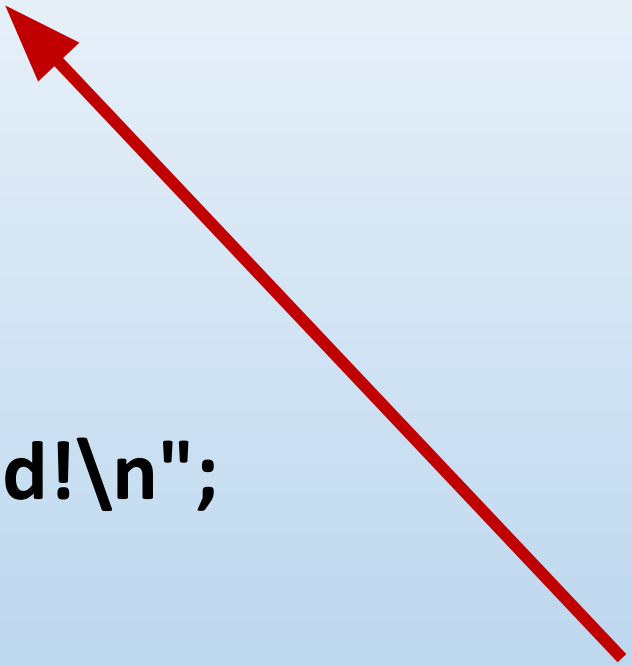
```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Hello, World!\n";
    return 0;
}
```

```
#include <iostream>
using namespace std;


int main()
{
    cout<<"Hello, World!\n";
    return 0;
}
```



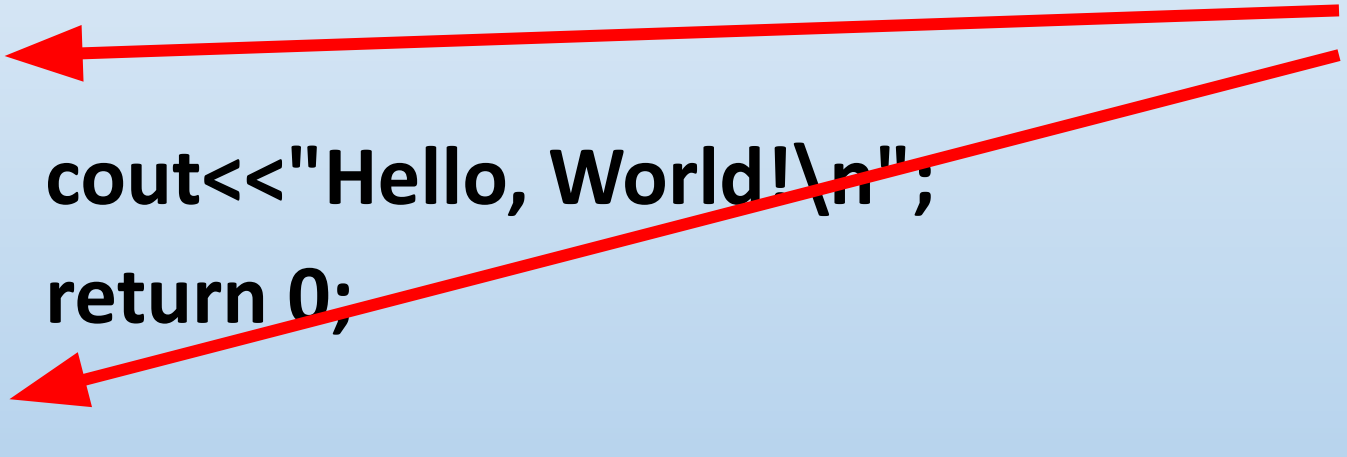
“Пристыковка”(включение) дополнительный файлов. Дело в том, что как правило программа использует дополнительные части, описанные в других файлах. В нашем случае используется оператор вывода, который описан в файле iostream. Но компилятору надо дать понять в каком файле описан этот оператор

```
#include <iostream>
using namespace std;


int main()
{
    cout<<"Hello, World!\n";
    return 0;
}
```



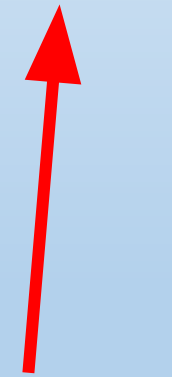
Внутри файла iostream cout заключён в пространство имён std (это некий контейнер). Если не написать “using namespace std;”, то пришлось бы написать std::cout << …

```
#include <iostream>
using namespace std;

int main()
{
    cout<<"Hello, World!\n";
    return 0;
}
```



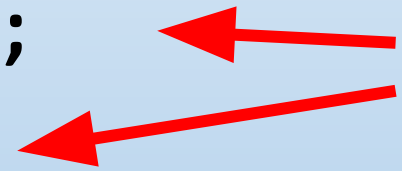
Инструкции, написанные между вот этими фигурными скобками будут выполняться, то есть сначала на экране отпечатается «Hello, World!», а потом выполниться оператор return 0;

```
#include <iostream>
using namespace std;


int main()
{
    cout<<"Hello, World!\n";
    return 0;
}
```



Между кавычками можно написать строку, которую вы хотите, чтобы компьютер вывел на экран и он в точности выведет эту строчку на экран

```
#include <iostream>
using namespace std;


int main()
{
    cout<<"Hello, World!\n";
    cout<<"I am Igor\n";
    return 0;
}
```



Если написать подряд два оператора – программа выведет две строки, указанные в каждом из операторов вывода друг за другом:

Hello, World!
I am Igor

```
#include <iostream>
using namespace std;


int main()
{
    cout<<"Hello, World!\n" <<"I am Igor\n";
    return 0;
}
```

Тот же самый эффект будет, если написать две строки в одном операторе вывода, но разделить их символами «<<»

```
Hello, World!
I am Igor
```

А что если нужно в строке вывести сам символ кавычек? Ведь кавычками мы обозначаем границы строк в исходном коде. Для этого надо перед кавычкой, которую нужно вывести на экран добавить символ «\»

```
#include <iostream>
using namespace std;

int main()
{
	cout<<"Peter and Basil are \"old friends\"";
	return 0;
}
```

Результат:

```
Peter and Basil are "old friends"
```

Другие escape-последовательности

```
\b          // Удаление последнего выведенного символа
\n                  // Перейти на начало новой строки
\t            // Перейти к следующей позиции табуляции
\\                          // Вывести обратную черту \
\"                        // Вывести двойную кавычку "
\'                      // Вывести одинарную кавычку '
```

Отключение escape-последовательностей

```
R"(текст_строки)"
```

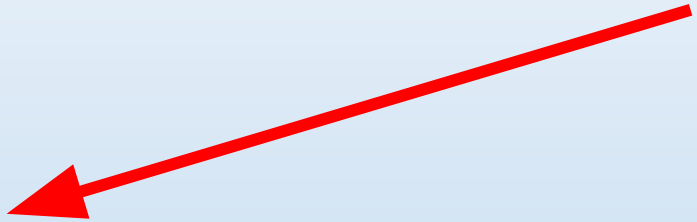
```
cout<<R"(hello\nworld)";  // на экране hello\nworld
cout<<R"("Test 'string'\t")"; // на экране "Test
                              // 'string'\t"
cout<<R"((Such brackets))";   // на экране
                              // (Such brackets)
```

```cpp
// Заголовок
#include <iostream>
// определение пространства имен, в котором есть cout<<
using namespace std;


// Главная функция
int main()
{
/* Следующая команда через 4 табуляции выводит текст
   Escape Sequences и переводит вывод на следующую
   строку
*/
   cout<<"\t\t\t\tEscape Sequences\n";
   // Выводит пустую строчку
   cout<<"\n";
/* Через 2 табуляции выводит текст \b, и еще через
   1 табуляцию Backspace Затем \n переводит вывод на
   следующую строку
*/
   cout<<"\t\t\\b"<<"\tBackspace\n";
   // Выводит пустую строку
   cout<<"\n";
/* Через 2 табуляции выводит текст \n, и еще через
   1 табуляцию New line
   Затем \n переводит вывод на следующую строку
*/
   cout<<"\t\t\\n"<<"\tNew line\n";
   // Выводит пустую строку
   cout<<"\n";
/* Через 2 табуляции выводит текст \t, и еще через
   1 табуляцию Horizontal tab Затем \n переводит
   вывод на следующую строку
*/
   cout<<"\t\t\\t"<<"\tHorizontal tab\n";
   // Выводит пустую строку
   cout<<"\n";
/* Через 2 табуляции выводит текст \\, и еще через
   1 табуляцию Backslash \
   Затем \n переводит вывод на следующую строку
*/
   cout<<"\t\t\\\\"<<"\tBackslash \\\n";
   // Выводит пустую строку
   cout<<"\n";
/* Через 2 табуляции выводит текст \t,
   и еще через 1 табуляцию Double quotation mark "
   Затем \n переводит вывод на следующую строку
*/
   cout<<"\t\t\""<<"\tDouble quotation mark \"\n";
   // Выводит пустую строку
   cout<<"\n";
/* Через 2 табуляции выводит текст \',
   и еще через 1 табуляцию Single quotation mark '
   Затем \n переводит вывод на следующую строку
*/
   cout<<"\t\t\'"<<"\tSingle quotation mark \'\n";
   // Выводит пустую строку
   cout<<"\n";

   return 0;

}
```

Вывод строк конечно же хорошо, но как запрограммировать сложение двух чисел, введённых пользователем?

```
#include <iostream>
using namespace std;

int main()
{
	int a, b, c;
	cout<<"Please, enter to numbers, for example 2 3 + Enter: ";
	cin >> a;
	cin >> b;
	c = a+b;
	cout <<"Result: "<< c;
	return 0;
}
```

```
#include <iostream>
using namespace std;


int main()
{
    int a, b, c;
    cout<<"Please, enter to numbers, for example 2 3 + Enter: ";
    cin >> a;
    cin >> b;
    c = a+b;
    cout <<"Result: "<< c;
    return 0;
}
```

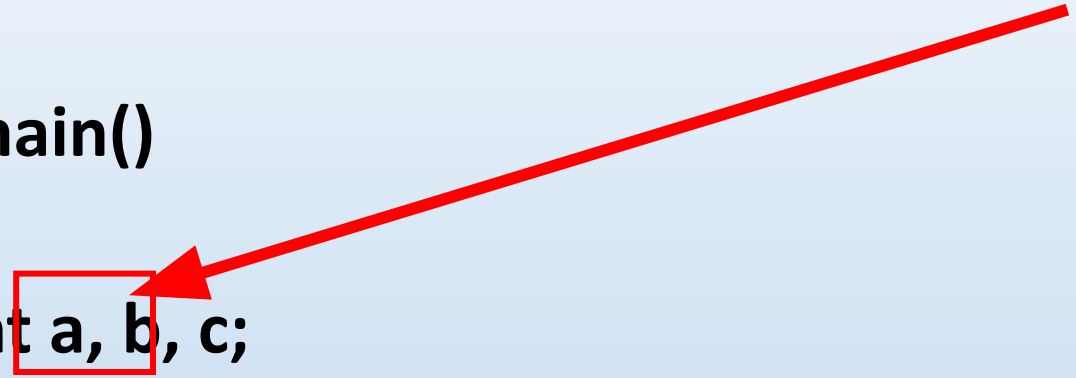


Объявление переменных. Переменная – это такая коробочка, в которой лежит конкретное значение. Это значение можно менять. Например, сначала присвоить ему число 10, потом через какое-то время его поменять и присвоить ему значение 11. В этой программе у нас три переменных с именами a, b и c. С помощью этих имён (a, b и c) остальные операторы могут обращаться к значению соответствующих переменных

Правила составления имён переменных

- Прописные и строчные буквы латинского алфавита (в c++ Age и age — это два разных имени)
- Цифры (цифра не может быть первым символом)
- Символ подчеркивания «_» (его можно использовать в том числе для обозначения пробелов, например age_of_man)
- Нельзя называть переменную ключевыми словами языка программирования
- Нельзя использовать никакие другие символы, кроме допустимых

```
#include <iostream>
using namespace std;


int main()
{
    int a, b, c;
    cout<<"Please, enter to numbers, for example 2 3 + Enter: ";
    cin >> a;
    cin >> b;
    c = a+b;
    cout <<"Result: "<< c;
    return 0;
}
```



int – это тип данных, то есть размер коробочек. Тип данных задаёт сколько байт нужно выделить для этой переменной в память и какие значения эта переменная может принимать. В c++ при объявлении любой переменной нужно указывать её тип

Типы данных в c++ (целочисленные)

Пояснение	Тип	Размер в байтах	Диапазон значений
Описывает целые числа	int	4	от -2147483648 до 2147483647
Описывает короткие целые числа	short	2	от -32768 до 32767
Описывает длинные целые числа	long	4	от -2147483648 до 2147483647
Описывает длинные целые числа	long long	8	от –9,223,372,036, 854,775,808 до 9,223,372, 036,854,775,807

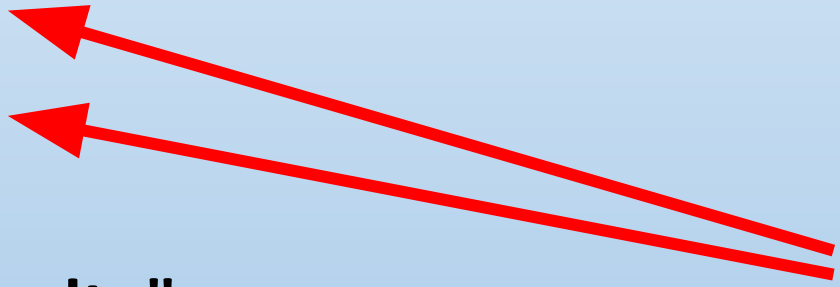
Типы данных в c++ (вещественные)

Пояснение	Тип	Размер в байтах
Описывает вещественные числа одинарной точности	float	4
Описывает вещественные числа двойной точности	double	8

Типы данных в c++ (логические)

Пояснение	Тип	Размер в байтах	Значения
Описывает логические значения	bool	1	true false

```
#include <iostream>
using namespace std;

int main()
{
    int a, b, c;
    cout<<"Please, enter to numbers, for example 2 3 + Enter: ";
    cin >> a;
    cin >> b;
    c = a+b;
    cout <<"Result: "<< c;
    return 0;
}
```



Оператор ввода cin. С помощью этого оператора мы кладём введённое с клавиатуры значение в переменные-коробочки a и b

```
#include <iostream>
using namespace std;

int main()
{
    int a, b, c;
    cout<<"Please, enter to numbers, for example 2 3 + Enter: ";
    cin >> a;
    cin >> b;
    c = a+b;
    cout <<"Result: "<< c;
    return 0;
}
```

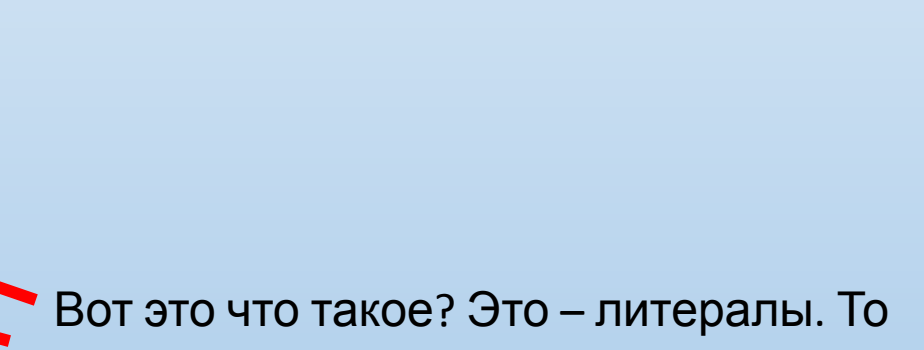
Складываем значение, лежащее в переменных a и b в переменную c

```
#include <iostream>
using namespace std;

int main()
{
    int a, b, c;
    cout<<"Please, enter to numbers, for example 2 3 + Enter: ";
    cin >> a;
    cin >> b;
    c = a+b;
    cout <<"Result: "<< c;
    return 0;
}
```

Выводим значение с на экран. Помимо строк с помощью оператора cout можно выводить значения переменных.

```
// Заголовок
#include <iostream>
// определение пространства имен, в котором есть cout<<
using namespace std;
// Главная функция
int main()
{
    // вывод пустой строки
    cout<<"\n";
    // Объявляем целочисленные константы
    int DayIn_2000Year=366;
    int HourInDay=24;
    // объявляем целочисленную переменную
    int HourIn_Year2000;
    // вычисляем искомое значение и
    // помещаем его в переменную HourIn_Year2000
    HourIn_Year2000=DayIn_2000Year*HourInDay;
    // выводим значение переменной
    // HourIn_Year2000 на экран
    cout<<"\t\t In 2000 year "<< HourIn_Year2000;
    cout<<" hours\n ";
    return 0;
}
```

```
// Заголовок
#include <iostream>
// определение пространства имен, в котором есть cout<<
using namespace std;
// Главная функция
int main()
{
    // вывод пустой строки
    cout<<"\n";
    // Объявляем целочисленные константы
    int DayIn_2000Year=366;
    int HourInDay=24;
    // объявляем целочисленную переменную
    int HourIn_Year2000;
    // вычисляем искомое значение и
    // помещаем его в переменную HourIn_Year2000
    HourIn_Year2000=DayIn_2000Year*HourInDay;
    // выводим значение переменной
    // HourIn_Year2000 на экран
    cout<<"\t\t In 2000 year "<< HourIn_Year2000;
    cout<<" hours\n ";
    return 0;
}
```



Вот это что такое? Это – литералы. То есть конкретные значения, заданные прямо в коде.
Для каждого типа данных литералы пишутся в разном формате (например, у литералов вещественного типа данных есть точка и дробная часть после неё)

Литералы

5	целочисленный литерал — int
5l	l или L означает long
true	логический литерал — bool
5.0	литерал с плавающей точкой, понимается как double
5.0f	f или F — с плавающей точкой, понимается как float
0.3e–2	литерал с плавающей точкой double, e или E отделяют экспоненциальную часть
‘d'	символьный литерал
«Visual»	строковый литерал — это набор произвольных символов, заключенный в кавычки. Компилятор воспринимает его именно как набор символов и никак обрабатывать его не собирается, даже если в кавычках окажутся какие-то ключевые слова и операции.

А теперь поработаем с вещественными числами!

```
// Заголовок
#include <iostream>
// определение пространства имен, в котором есть cout<<
using namespace std;
// Главная функция
int main()
{
     // Объявляем переменную Discount
    float Discount=0.05;

    // Объявляем переменную
    Cost float Cost=10.50;
    // Объявляем переменную
    Count int Count=5;
    // Объявляем переменную
    Price float Price;
    // Вычисляем значение переменной Price
    Price=Count*Cost-Count*Cost*Discount;
    // Выводим итоговую стоимость товара со скидкой
    cout<<"Please, pay:"<<Price<<"\n";

    return 0;
}
```

Константы

Кроме переменных есть ещё и константы. Константы – это тоже коробочки, у которых есть размер (то есть тип), но в них можно только один раз положить значение – при объявлении. Если попытаться положить в них значение второй раз – компилятор выдаст ошибку. Перед объявлением константы пишется ключевое слово «const» затем тип данных, затем (имя как у переменной), а затем знак равенства и какое-то конкретное значение:

const Тип_данных Имя = Значение;

Например, целочисленная константа со значением 5:

const int a = 5;

Но если потом попытаться поменять значение константы, то будет ошибка:

a = 10; // Ошибка!

До следующего урока! Домашнее задание делаем!