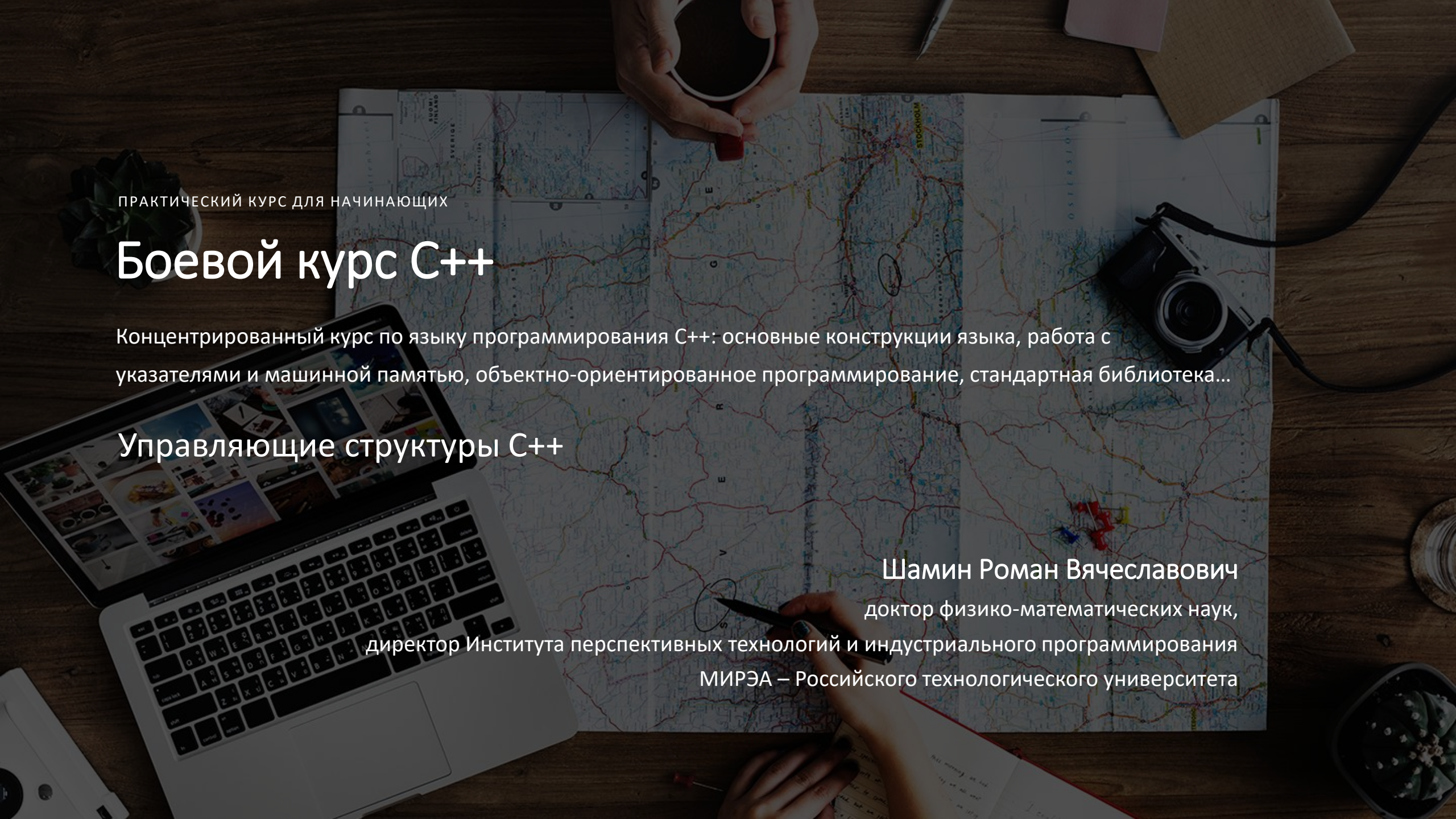

ПРАКТИЧЕСКИЙ КУРС ДЛЯ НАЧИНАЮЩИХ

Боевой курс C++

Концентрированный курс по языку программирования C++: основные конструкции языка, работа с указателями и машинной памятью, объектно-ориентированное программирование, стандартная библиотека…

Управляющие структуры C++

Шамин Роман Вячеславович

доктор физико-математических наук,

директор Института перспективных технологий и индустриального программирования

МИРЭА – Российского технологического университета

Язык C++ является строго типизированным языком, но тем не менее в новом стандарте есть так называемый синтаксический сахар – это автоматическое определение типа данных при инициализации переменных. Например:

```
auto a = 648; // int
auto x = 3.0 / 2.0; // double
auto b = a; // int
```

Использование auto не означает отхода от строгой типизации – при ее использовании тип данных фиксируется и в дальнейшем не меняется.

С помощью cout можно выводить значения переменных на печать, а с помощью cin можно вводить значения переменных с терминала. Вот как это делается:

```
int a; // создаем переменную
cout << "Введите число > "; // выводим запрос
cin >> a; // запрашиваем значение с терминала, выполнение программы приостанавливается
cout << "квадрат числа " << a << " равен " << a * a << endl; // выводим результат
```

Заметьте, что когда мы запрашиваем значение, то ход выполнения программы будет приостановлен до тех пор, пока вы не введете значение.

Если Вы введете в терминале текст, который не будет соответствовать типу данных, который ожидается в cin, то переменная получит значение по умолчанию, например, ноль, но может возникнуть и ошибка выполнения программы – это будет зависеть от компилятора. Если Вы хотите проводить проверку вводимых пользователем данных, то нужно запрашивать строку, а потом ее анализировать. О том как работать со строками мы поговорим несколько позже.

Посмотреть видео

Важнейшим элементом любого языка программирования является оператор условного перехода. Простейший оператор в C++ – это оператор if(). Покажем на практике как он работает:

```
int a; // создаем переменную
cout << "Введите число > "; // выводим запрос
cin >> a; // запрашиваем значение с терминала, выполнение программы приостанавливается
if (a > 0) // проверяем условие
{
    cout << "Это положительное число" << endl; // выполнится только, если условие - истинно
}
```

В скобках оператора if () может быть любое выражение. Это выражение будет приведено к целому типу и если получится значение отличное от нуля, то оператор if сочтет, что условие выполнено, в противном случае – нет. В родительском языке C логических типов не было, поэтому использовались числовые значения в качестве логических переменных. Для обратной совместимости – это, к сожалению, поддерживается и в C++. Старайтесь пользоваться только логическими выражениями в операторе if ().

Оператор if имеет вариант if – else, когда можно указать какой код будет выполнен при истинном условии, а какой при ложном условии:

```
if (a > 0) // проверяем условие
{
    cout << "Это положительное число" << endl; // выполнится только, если условие - истинно
}
else
{
    cout << "Это число отрицательное или равно нулю"; // выполнится только, если условие - ложно
}
```

Еще условные операторы перехода

В языке C++ как в языке C есть тернарный условный оператор, который имеет три операнда:

```
(условие) ? [код, если верно условие] : [код, если условие неверно];
```

Пример использования этого оператора:

```
int a = 7;
int b = 8;
char c; // создаем переменную
cout << "Введите + или - > "; // выводим запрос
cin >> c; // запрашиваем значение с терминала, выполнение программы приостанавливается
int res = (c == '+') ? a + b : a * b; // проверяем введен ли +
cout << "Результат = " << res << endl;
```

Использование тернарного оператора оправдано, когда нужно быстро реализовать альтернативу.
Оператор switch..case реализует возможность выбора из нескольких условий:

```
int a = 7, b = 8, res;
char c;
cout << "Введите + или * > "; // выводим запрос
cin >> c; // запрашиваем значение с терминала
switch (c) // проверяем значение
{
case '+': // если +
    res = a + b;
    break; // нужно, чтобы не выполнялся код дальше
case '*': // если -
    res = a * b;
    break;
default:
    res = 0; // будет выполнено, если нет совпадений ранее
}
cout << "Результат = " << res << endl;
```

Обратите внимание, если убрать break, то код будет выполняться дальше, включая все case!

Старайтесь не использовать этот оператор.

Цикл – это еще одно фундаментальное понятие в программировании. Цикл позволяет выполнять (повторять) код нужное количество раз.

Основным оператором цикла в C/C++ является оператор for(;;). Этот оператор содержит три выражения в скобках, разделенных точкой с запятой:

```
for([инициализация]; [условие]; [код после прохода цикла])
{
    … код цикла…
}
```

Сначала один раз выполняется код инициализации, потом проверяется условие, если оно верное, то выполняется один раз код цикла, потом код после прохода цикла, снова проверяется условие…

Вот пример цикла, который печатает заданное количество звездочек:

```
int N;
cout << "Введите количество звездочек > ";
cin >> N; // запрашиваем количество
for (int i = 0; i < N; i++) // цикл повторяем N раз
{
    cout << "*"; // выводим одну звездочку за раз
}
```

Заметим, что мы в цикле определяем переменную i, которая будет иметь область видимости внутри цикла. В новом цикле можно будет снова определить переменную с таким именем.

Внутри цикла можно выполнять и другие циклы – это называется «вложенные циклы». Вложенность циклов может быть произвольной. Вот пример вложенного цикла

Объявим переменные различного типа:

```
int N;
cout << "Введите количество звездочек > ";
cin >> N; // запрашиваем количество
for (int i = 0; i < N; i++) // цикл повторяем N раз
{
    for (int j = 0; j < i + 1; j++) // вложенный цикл
    {                               // количество итераций зависит от i
        cout << "*"; // выводим одну звездочку за раз
    }
    cout << endl; // новая строка
}
```

Вот еще пример вложенного цикла:

```
int N;
cout << "Введите количество звездочек > ";
cin >> N; // запрашиваем количество
for (int i = 0; i < N; i++) // цикл повторяем N раз
{
    for (int j = 0; j < N; j++) // вложенный цикл
    {
        cout << i + j << "\t"; // в конце строки - знак табуляции
    }
    cout << endl; // новая строка
}
```

Другие операторы цикла

Еще одним оператором цикла является оператор

```
while([условие])
{
    … код …
}
```

Этот цикл работает очень просто – если условие истинно, то выполняется код, потом снова проверяется условие.

```
int x = 0;
while (x != 7) // пока x не равно 7 продолжать цикл
{
    cout << "Введите число > ";
    cin >> x; // ввести число
}
```

Второй цикл имеет вид do..while, когда первый раз тело цикла всегда выполняется, а потом проверяется условие, в зависимости от которого цикл продолжается или нет.

```
int n;
do
{
    cout << "Введите число ноль > ";
    cin >> n;
}
while (n == 0); // продолжать до тех пор, пока пользователь вводит 0
```

Прерывание цикла

В теле любого цикла могут встретиться оператор break; в результате которого выполнение цикла немедленно прекращается и continue; когда завершается выполнение текущего прогона цикла и переход к проверке условия. В случае for перед проверкой условия будет выполнен код после прохода цикла.

Операторы break; и continue; во вложенном цикле действуют только на ближайший цикл.

```
while (true) // продолжать пока не будет break;
{
    double x;
    cout << "Введите число > ";
    cin >> x;
    if (x == 0) // если ноль, то не делить
    {
        cout << "Делить на ноль не могу!" << endl;
        continue; // прервать выполнение тела цикла и начать заново
    }
    if (x < 0)
    {
        break; // если введено отрицательное число, то выйти из цикла
    }
    cout << "1 / " << x << " = " << 1.0 / x << endl;
}
```

Еще есть конструкция goto, которую мы даже рассматривать не будем.

Скачать текст программы

Посмотреть видео