УВАЖАЕМЫЕ СТУДЕНТЫ! Изучите приведенную лекцию, законспектируйте основные понятия контроля версий программного обеспечения.

Ответы на вопросы, фотоотчет, предоставить преподавателю на e-mail: r.bigangel@gmail.com до 13.02.2023.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)111-37-59, (Viber, WhatsApp), vk.com: https://vk.com/daykini

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция № 5
Тема: «Типы данных, переменные в языке C++. Основные операторы языка C++»

План лекции:

1. Типы данных и переменные C++

2. Операторы C++

Типы данных

В языке C++ *все переменные* имеют определенный тип данных. Например, переменная, имеющая целочисленный тип, не может содержать ничего кроме целых чисел, а переменная с плавающей точкой — только дробные числа.

Тип данных присваивается переменной при ее объявлении или инициализации. Ниже приведены основные типы данных языка C++, которые нам понадобятся.

Основные типы данных в C++

- **int** — целочисленный тип данных.
- **float** — тип данных с плавающей запятой.

- **double** — тип данных с плавающей запятой двойной точности.
- **char** — символьный тип данных.
- **bool** — логический тип данных.

Объявление переменной

Объявление переменной в C++ происходит таким образом: сначала указывается тип данных для этой переменной а затем название этой переменной.

Пример объявления переменных

```
int a; // объявление переменной a целого типа.
float b; // объявление переменной b типа данных с плавающей запятой.
double c = 14.2; // инициализация переменной типа double.
char d = 's'; // инициализация переменной типа char.
bool k = true; // инициализация логической переменной k.
```

- Заметьте, что в C++ **оператор присваивания (=)** — не является знаком равенства и не может использоваться для сравнения значений. Оператор равенства записывается как «двойное равно» — ==.
- Присваивание используется для сохранения определенного значение в переменной. Например, запись вида a = 10 задает переменной a значение числа 10.

Простой калькулятор на C++

Сейчас мы напишем простую программу-калькулятор, которая будет принимать от пользователя два целых числа, а затем определять их сумму:

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

int main()
{
    setlocale(0, "");
    /*7*/ int a, b; // объявление двух переменных a и b целого типа данных.
    cout << "Введите первое число: ";
    cin >> a; // пользователь присваивает переменной a какое-либо значение.
    cout << "Введите второе число: ";
    cin >> b;
    /*12*/  int c = a + b; // новой переменной c присваиваем значение суммы введенных
пользователем данных.
    cout << "Сумма чисел = " << c << endl; // вывод ответа.
    return 0;
}
```

Разбор кода

В 7-й строке кода программы мы объявляем переменные «a» и «b» целого типа int. В следующей строке кода выводится сообщение пользователю, чтобы он ввел с клавиатуры первое число.

В 9-й строке стоит еще незнакомая вам конструкция — cin >>. С помощью нее у пользователя запрашивается ввод значения переменной «a» с клавиатуры. Аналогичным образом задается значение переменной «b».

В 12-й строке мы производим инициализацию переменной «c» суммой переменных «a» и «b». Далее находится уже знакомый вам оператор cout, который выводит на экран строку и значение переменной «c».

- При выводе переменных, они *не заключаются в кавычки*, в отличие от строк.

Операторы языка C++

Операторы управляют процессом выполнения программы. Набор операторов языка C++ содержит все управляющие конструкции структурного программирования.

Составной оператор ограничивается фигурными скобками. Все другие операторы заканчиваются точкой с запятой.

- **Пустой оператор – ;**

Пустой оператор – это оператор, состоящий только из точки с запятой. Он может появиться в любом месте программы, где по синтаксису требуется оператор. Выполнение пустого оператора не меняет состояния программы.

- **Составной оператор – {...}**

Действие составного оператора состоит в последовательном выполнении содержащихся в нем операторов, за исключением тех случаев, когда какой-либо оператор явно передает управление в другое место программы.

- **Оператор обработки исключений**

try { <*операторы*> }
catch (<*объявление исключения*>) { <*операторы*> }
catch (<*объявление исключения*>) { <*операторы*> }
...
catch (<*объявление исключения*>) { <*операторы*> }

- **Условный оператор**

if (<*выражение*>) <*оператор 1*> [**else** <*оператор 2*>]

- **Оператор-переключатель**

switch (<*выражение*>)
{
  case <*константное выражение 1*>: <*операторы 1*>
  case <*константное выражение 2*>: <*операторы 2*>
  ...
  case <*константное выражение N*>: <*операторы N*>
  [**default**: <*операторы*>]
}

Оператор-переключатель предназначен для выбора одного из нескольких альтернативных путей выполнения программы. Вычисление оператора-переключателя начинается с вычисления *выражения*, после чего управление передается *оператору*, помеченному *константным выражением*, равным вычисленному значению *выражения*. Выход из оператора-

переключателя осуществляется оператором **break**. Если значение *выражения* не равно ни одному *константному выражению*, то управление передается *оператору*, помеченному ключевым словом *default*, если он есть.

- **Оператор цикла с предусловием**

while (<*выражение*>) <*оператор*>

- **Оператор цикла с постусловием**

do <*оператор*> **while** <*выражение*>;

В языке C++ этот оператор отличается от классической реализации цикла с постусловием тем, что при истинности *выражения* происходит продолжение работы цикла, а не выход из цикла.

- **Оператор пошагового цикл**

for ([<*начальное выражение*>]; [<*условное выражение*>]; [<*выражение приращения*>]) <*оператор*>

Тело оператора **for** выполняется до тех пор, пока *условное выражение* не станет ложным (равным 0). *Начальное выражение* и *выражение приращения* обычно используются для инициализации и модификации параметров цикла и других значений. *Начальное выражение* вычисляется один раз до первой проверки *условного выражения*, а *выражение приращения* вычисляется после каждого выполнения *оператора*. Любое из трех выражений заголовка цикла, и даже все три могут быть опущены (не забывайте только оставлять точки с запятой). Если опущено *условное выражение*, то оно считается истинным, и цикл становится бесконечным.

Оператор пошагового цикла в языке C++ является гибкой и удобной конструкцией, поэтому оператор цикла с предусловием **while** используется в языке C++ крайне редко, т.к. в большинстве случаев удобнее пользоваться оператором **for**.

- **Оператор цикла для диапазона**

for (<*элемент*>: <*массив*>) <*оператор*>

Оператор цикла для диапазона предоставляет способ итерации по массиву (или другой структуре). Выполняется итерация по каждому элементу

массива, при этом значение текущего элемента массива присваивается переменной, объявленной как *элемент*. В целях улучшения производительности объявляемый элемент должен быть того же типа, что и элементы массива. Можно использовать автоматический вывод типа. *Элемент* может быть изменён, но это не влияет на массив. Для того, чтобы изменить массив, *элемент* надо объявлять как ссылку. В цикле обрабатываются все элементы массива.

- **Оператор разрыва**

break;

Оператор разрыва прерывает выполнение операторов **while**, **do**, **for** и **switch**. Он может содержаться только в теле этих операторов. Управление передается оператору программы, следующему за прерванным. Если оператор разрыва записан внутри вложенных операторов **while**, **do**, **for**, **switch**, то он завершает только непосредственно охватывающий его оператор.

- **Оператор продолжения**

continue;

Оператор продолжения передает управление на следующую итерацию в операторах цикла **while**, **do**, **for**. Он может содержаться только в теле этих операторов. В операторах **do** и **while** следующая итерация начинается с вычисления условного выражения. В операторе **for** следующая итерация начинается с вычисления выражения приращения, а затем происходит вычисление условного выражения.

- **Оператор возврата**

return [<*выражение*>];

Оператора возврата заканчивает выполнение функции, в которой он содержится, и возвращает управление в вызывающую функцию. Управление передается в точку вызывающей функции, непосредственно следующую за оператором вызова. Значение *выражения,* если она задано, вычисляется, приводится к типу, объявленному для функции, содержащей оператор

возврата, и возвращается в вызывающую функцию. Если *выражение* опущено, то возвращаемое функцией значение не определено.

С формальной точки зрения операторы **break**, **continue** и **return** не являются операторами структурного программирования. Однако их использование в ограниченных количествах оправдано, когда они упрощают понимание программы и позволяют избегать больших вложенных структур. Например, мы проверяем входные данные на аномалии. Если не использовать эти операторы, то всю обработку придется вложить в условный блок, что ухудшает читабельность программы. Вместо этого можно написать небольшой условный блок, который организует выход из функции при неверных исходных данных.

Ввод/вывод не является частью языка C++, а осуществляется функциями, входящими в состав стандартной библиотеки.

1. Инструкции (стейтменты)

Стейтмент (англ. ***«statement»***) — это наиболее распространенный тип инструкций в программах. Это и есть та самая инструкция, наименьшая независимая единица в языке C++. Стейтмент в программировании — это то же самое, что и «предложение» в русском языке. Мы пишем предложения, чтобы выразить какую-то идею. В языке C++ мы пишем стейтменты, чтобы выполнить какое-то задание. **Все стейтменты в языке C++ заканчиваются точкой с запятой.**

Есть много разных видов стейтментов в языке C++. Рассмотрим самые распространенные из них:

```
1 int x;
2 x = 5;
3 std::cout << x;
```

`int x` — это **стейтмент объявления** (англ. ***«statement declaration»***). Он сообщает компилятору, что `x` является переменной. В программировании

каждая переменная занимает определенное число адресуемых ячеек в памяти в зависимости от её типа. Минимальная адресуемая ячейка — байт. Переменная типа int может занимать до 4-х байт, т.е. до 4-х адресуемых ячеек памяти. Все переменные в программе должны быть объявлены, прежде чем использованы. Мы детально поговорим о переменных на следующих уроках.

`x = 5` — это **стейтмент присваивания** (англ. ***«assignment statement»***). Здесь мы присваиваем значение `5` переменной `x`.

`std::cout << x;` — это **стейтмент вывода** (англ. ***«output statement»***). Мы выводим значение переменной `x` на экран.

2. Выражения

Компилятор также способен обрабатывать выражения. **Выражение** (англ. ***«expression»***) — это математический объект, который создается (составляется) для проведения вычислений и нахождения соответствующего результата. Например, в математике выражение `2 + 3` имеет значение `5`. **Выражения в языке C++ могут содержать**:

– отдельные цифры и числа (например, `2`, `45`);

– буквенные переменные (например, `x`, `y`);

– операторы, в т.ч. математические (например, `+`, `-`);

– функции.

Выражения могут состоять как из единичных символов — цифр или букв (например, `2` или `x`), так и из различных комбинаций этих символов с операторами (например, `2 + 3`, `2 + x`, `x + y` или `(2 + x) * (y - 3)`). Для наглядности разберем простой корректный стейтмент присваивания `x = 2 + 3;`. Здесь мы вычисляем результат сложения чисел `2 + 3`, который затем присваиваем переменной `x`.

3. Функции

В языке C++ стейтменты объединяются в блоки — функции. **Функция** — это последовательность стейтментов. Каждая программа, написанная на языке C++, должна содержать главную **функцию main()**. Именно с первого стейтмента, находящегося в функции main(), и начинается выполнение всей программы. Функции, как правило, выполняют конкретное задание. Например, функция max() может содержать стейтменты, которые определяют большее из заданных чисел, а функция calculateGrade() может вычислять среднюю оценку студента по какой-либо дисциплине.

4. Библиотеки

Библиотека — это набор скомпилированного кода (например, функций), который был «упакован» для повторного использования в других программах. С помощью библиотек можно расширить возможности программ. Например, если вы пишете игру, то вам придется подключать библиотеки звука или графики (если вы самостоятельно не хотите их создавать).

Язык C++ не такой уж и большой, как вы могли бы подумать. Тем не менее, он идет в комплекте со **Стандартной библиотекой C++**, которая предоставляет дополнительный функционал. Одной из наиболее часто используемых частей Стандартной библиотеки C++ является **библиотека iostream**, которая позволяет выводить данные на экран и обрабатывать пользовательский ввод.

5. Пример простой программы

Теперь, когда у вас есть общее представление о том, что такое стейтменты, функции и библиотеки, давайте рассмотрим еще раз программу «Hello, world!»:

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, world!";
    return 0;
}
```

Строка №1: Специальный тип инструкции, который называется **директивой препроцессора**. Директивы препроцессора сообщают компилятору, что ему нужно выполнить определенное задание. В этом случае мы говорим компилятору, что хотели бы подключить содержимое заголовочного файла <iostream> к нашей программе. Подключение заголовочного файла <iostream> дает нам возможность использовать функционал библиотеки iostream, что, в свою очередь, позволяет выводить нам данные на экран.

Строка №2: Пустое пространство, которое игнорируется компилятором.

Строка №3: Объявление главной функции main().

Строки №4 и №7: Указываем компилятору область функции main(). Всё, что находится между открывающей фигурной скобкой в строке №4 и закрывающей фигурной скобкой в строке №7 — считается содержимым функции main().

Строка №5: Наш первый стейтмент (заканчивается точкой с запятой) — стейтмент вывода. `std::cout` — это специальный объект, используя который мы можем выводить данные на экран. `<<` — это оператор вывода. Всё, что мы отправляем в `std::cout`, — выводится на экран. В этом случае, мы выводим текст `"Hello, world!"`.

Строка №6: **Оператор возврата return**. Когда программа завершает свое выполнение, функция main() передает обратно в операционную

систему значение, которое указывает на результат выполнения программы: успешно ли прошло выполнение программы или нет.

Если оператор return возвращает число 0, то это значит, что всё хорошо! Ненулевые возвращаемые значения указывают на то, что что-то пошло не так и выполнение программы было прервано. Об операторе return мы еще поговорим детально на соответствующем уроке.

6. Синтаксис и синтаксические ошибки

Как вы, должно быть, знаете, в русском языке все предложения подчиняются правилам грамматики. Например, каждое предложение должно заканчиваться точкой. Правила, которые регулируют построение предложений, называются **синтаксисом**. Если вы не поставили точку и записали два предложения подряд, то это является нарушением синтаксиса русского языка.

Язык C++ также имеет свой синтаксис: правила написания кода/программ. При компиляции вашей программы, компилятор отвечает за то, чтобы ваша программа соответствовала правилам синтаксиса языка C++. Если вы нарушили правила, то компилятор будет ругаться и выдаст вам ошибку.

Например, давайте посмотрим, что произойдет, если мы не укажем в конце стейтмента точку с запятой:

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Hello world!"
6     return 0;
7 }
```

Результат:

E0065: требуется точка с запятой ";"

C2143: синтаксическая ошибка: отсутствие ";" перед "}"

Допущена синтаксическая ошибка в строке №6: мы забыли указать точку с запятой перед оператором return. В этом случае ошибка на самом деле в конце строки №5. В большинстве случаев компилятор правильно определяет строку с ошибкой, но есть ситуации, когда ошибка не заметна вплоть до начала следующей строки.

Синтаксические ошибки нередко совершаются при написании программ. К счастью, большинство из них можно легко найти и исправить. Но следует помнить, что программа может быть полностью скомпилирована и выполнена только при отсутствии ошибок.

Контрольные вопросы:

1. В чём разница между стейтментом и выражением?

2. В чём разница между функцией и библиотекой?

3. Чем заканчиваются стейтменты в языке C++?

4. Что такое синтаксическая ошибка?

5. Типы данных и переменные C++

6. Операторы C++