Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

Introduction to C++

MAS Foundations Course SS-19

September 19, 2019

Original Author: Sushant Vijay Chavanan & Modifications by: Ethan Oswlad Massey

About the content

- This contents of these slides are based on a very good tutorial available at Tutorials point
- We cover the following basic topics of C++ today:
  - Setup a baisc development environemnt
  - A "hello world" program to demonstrate the basic syntax.
  - Variable declarations and basic data types provided by C++.
  - Basic operators (Arithmetic, Relational, Logical, Assignment and Conditional)
  - Decision making (if-else and switch) and looping (for, while and do-while)
  - Using functions
  - Pointers and references

Topics for Advanced C++ (next session)

- Dynamic memory allocation.
- Writing basic classes and crating objects.
- Using object oriented programming concepts like Inheritance, Overloading, Polymorphism.
- Optional: Using STL (Standard Template Library).
- Optional: Additional Tools for Development

Setting up the development environment

- Check if you have a proper g++ installed on your system using:
  `g++ --version`
- If you don't have it already, install it using:
  `sudo apt-get install g++`
- Clone the C++ training repository using:
  `git clone`
- Test if everything is working fine using:
  `cd src/testSetup/`
  `./testBuildSetup.sh`
  You should get the following message: **Test successful!**

Hello World!

- A basic C++ code which simply prints **Hello World** to the console is provided in the *helloWorld* folder.
- Since C++ is a compiled language, every time you make any change to the code, you need to recompile it to see the changes.
- We will use g++ to compile our hello world program. The output will be a binary file (something like a *.exe* file you might have seen on windows) which can be run using the terminal.
- To compile the program, *cd* to the helloWorld directory and use the command: `g++ -o helloWorld helloWorld.cpp`
- Run the compiled program using: `./helloWorld`

Exercise - Self Introduction

- Write a program to print your name and nationality to the screen **on separate lines**. Hint: Use the Hello World program as an example.
- Create a folder called SelfIntroduction (in linux avoid using spaces in folder or file names like "Self Introduction")
- Create a C++ file called SelfIntroduction.cpp in the newly created folder and write your code in it. **Do not forget the semicolons after each statement.**
- Compile it using g++ and check your output.
- Output should look like:

```
Hello, my name is Sushant
I come from India
```

Variables

- Suppose you want to ask the user of your program to input two integer numbers so that you can display their sum. You will need to store this information somehow in your code. We use variables for this purpose.
- Simply put, a variable is just a name given to a part of the memory (RAM) that we will use to store information.
- C++ provides some basic predefined data types (primatives) for variables:
  - **int** : to store integer values.
  - **char** : to store a **single** character. For example: **a**
  - **float** : to store floating point (or fractional) values
  - **double** : to store fractional values with very high precision
  - **bool** : to store binary values (True/False)
  - **void** : to indicate no datatype (we will see more on this when we look at functions)
- To store strings of character's in C++, we need to include the header called string.

Example - Sum of two integers

- cd to the folder *sumOfTwoIntegers*
- Compile and run the provided example.
- Create three more variables in the same file to store the difference, product and fractional part.
- Compute and print the results of the mathematical operations on the input numbers using the `-` `*` and `/` operators respectively.
- Use the two inputs as 17 and 3. The output should look like:

```
Sum of 17 and 3 is: 20
Difference of 17 and 3 is: 14
Product of 17 and 3 is: 51
Division of 17 by 3 is: 5.66667
```

- Is the result of your division proper?

Exercise - Print User details

- Write a program to request the user for his first name, last name, nationality and **year of birth** (not age).
- Compute the user's age using just the current year and year of birth.
- The program should then print the user details as:
  `Sushant Chavan is a 28 year old Indian`

Operators

Arithmetic

- We already saw 4 arithmetic operators `+`, `-`, `*` and `/` used for addition, subtraction, multiplication and division respectively. There are three more arithmetic operators:
  - `%` : the modulo operators, used to compute the remainder of a division.
  - `++` : increment operator. Increases a variable by 1.
  - `--` : decrement operator. Decreases a variable by 1.

Operators

Relational

These operators are used to compare two variables.

- == : check if two variables are equal.
  For example: `bool result = (num1 == num2);`
- != : check if two variables are not equal.
- > : check if left variable is greater than right variable.
  For example: `bool result = (num1 > num2);`
- < : check if left variable is less than right variable.
- >= : check if left variable is greater than or equal to right variable.
- <= : check if left variable is less than or equal to right variable.

Operators

Logical

These operators operate on boolean variables or boolean results of operators

- `&&` : logical AND operator. This operation returns true if both the input variables to this operator are true.
- `||` : logical OR operator. This operation returns true if atleast one of the input variables to this operator is true.
- `!` : logical NOT operator. Takes only one input and returns the negation of its value.

Operators

Comments

- There are many more types of operators available but we cannot go through all of them due to time constraints.
- Please click here to get a better understanding of them.
- Consider the below example where multiple operators are used in a single expression.
  ```
  int x = a + b * 25 + c / 2
  ```
  Here the decision of applying the operators in the right order is done using the operator precedence.
- The link mentioned in the previous bullet point also describes this.

Example - Operators

- A sample code to demonstrate some operators is available in the folder *operators*.
- This is a very brief example. It is highly recommended that you try many other combinations in your free time.
- Try out how operator precedence affects evaluation of the expressions.
- Think about (or Google) PEMDAS

Decision making

- Used to execute a part of the code only if a certain condition is met.
- C++ provides 3 types of decision making constructs:
  - The simple *if* statement
  - The *if-else* statement
  - The *switch* statement
- The if and if-else statements use a boolean expression to determine if a block of code should be executed.
- The switch statement on the other hand compares a variable against a list of values.

Decision making - Example

- cd to the directory *decisionMaking*
- There are two examples to mimic the voting process of an election: one with an if-else and another that uses a switch instead of an if-else.
- In the example for if-else,
  - What is **const** used for?
  - Until when do you think the variable partyID is valid? Or in other words what is the **scope of the variable** partyID?
- In the example for switch,
  - Is it possible to replace the outer if-else statement with a switch statement?
  - What will happen if we remove the break keywords?

Decision making - Exercise

- Write a simple program to check if a visa is needed to enter Germany based on the user's nationality.
- Assume that only citizens of USA, Japan and UK are exempt from requiring a Visa. (keep in mind when entering the user's nationality that C++ is case sensitive. Therefore 'Japan' is different than 'japan').
- The program should print `Visa Required` or `Visa Not Required` based on the nationality of the user.

Looping

- Facilitates execution of a piece of code multiple times.
- Keeps repeating the code block until a control condition becomes false.
- There are three basic types of loops:
  - The *for* loop
  - The *while* loop
  - The *do-while* loop
- Depending on the type of loop used, either at the start of each iteration or at the end of it, the control expression is evaluated and checked if it is true.
- If the expression evaluates to true, the piece of code is repeated again. Otherwise, the execution is stopped and the statement that follows the loop is executed.
- It is possible to loop forever (called an **infinite loop**) if the control statement never evaluates to false.

Looping

Control statements

- We have two very common control statements used in conjunction with loops:
  - The **break** statement.
    - » This will immediately terminate the loop unconditionally.
    - » We do not check if the control expression evaluates to true or false.
    - » All statements in of the loop code following the break statement will be skipped.
  - The **continue** statement.
    - » This control statement does not stop the loop instantaneously.
    - » It is used to stop just the current iteration of the loop.
    - » All statements of the loop body that follow this statement will be skipped for this iteration.
    - » The loop however continues with its next iteration.

Looping - Examples

- for loop:
  - cd to the folder *loops* and open the file *forloop.cpp*
  - How can we modify this to compute sum of only even numbers?
  - How can we modify this code to stop execution if the sum exceeds 1000?
- while loop : open the file *whileloop.cpp*
- do-while loop : open the file *dowhileloop.cpp*

Looping - Exercise

- Modify the previous program that you wrote to check the Visa requirements for Germany to use loops. Choose any loop type which you think is the best for this use-case.
- The program should not stop after each input and should start over until the user enters `END` as an input.
- The output should look something like:

```
Enter your nationality: Japan
Visa Not Required.
⋮
Enter your nationality: China
Visa Required.
Enter your nationality: END
```

Functions

- It is a group of statements that perform a specific task. For example, determining the maximum of two numbers.
- Typically the code is split into many logical functions which helps in easier understanding of the code and better maintainability.
- The below pesudo-code shows the typical structure of a function
- The first line of the function is called the function signature.

```
return_type function_name( parameter list )
{
    body of the function
}
```

Functions

Example

```
int add(int a, int b) // This is a function
{
    int sum = a + b;
    return sum;
}
int main()
{
    int num1 = 10;
    int num2 = 20;
    int sum = add(num1, num2); // This is a function call
    cout << "Sum = " << sum << endl;
    return 0;
}
```

Functions - Example

- cd to the *functions* directory and open the file *maxValue.cpp*
- Comments:
  - It is possible to define more than one function with the same function name as long as the parameters differ.
  - It is possible to call a function inside another function. Example the max of two number is called inside the function max of three numbers.
  - It is not necessary to take the result of a function into a variable.
  - It is also possible to call a function within its own body. This is called a recursive function.

Functions

What is the result of this code?

```
int swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}
int main()
{
    int num1 = 10, num2 = 20;
    swap(num1, num2);
    cout << "Swapped numbers: " << num1 << " " << num2 << endl;
    return 0;
}
```

Functions

Pass by value, reference or pointer

- There are three ways in which a variable can be passed as a parameter to a function:
  1. Pass by value.
     For example: `int swap(int a, int b)`
  2. Pass by reference.
     For example: `int swap(int& a, int& b)`
  3. Pass by pointer.
     For example: `int swap(int* a, int* b)`
- *Pass by value* **creates a copy** of the values that are passed to it and therefore any changes to these variables do not affect the original variables that were used in the function call.
- The other two allow modifying the original variables used during the function call.

Functions - Example

Swapping two numbers

- A reference in the simplest terms means giving the memory location associated with a variable another name (like an alias).
- Therefore, since there is no copy created, any changes done to the alias will reflect in the original variable too.
- We use the `&` symbol after a datatype to indicate that it is a reference (or alias) to another variable.
  For example: `int& num1Alias;`
- **Question:** How can you return more than one return value from a function?

Functions - Exercise

- Write a simple calculator program that can add, subtract, multiply and divide.
- Each of these operations must be computed in a different function.
- Ask the user to input two integers and then ask for the choice of operation he wants to perform.
- Then call the appropriate function written by you based on the user's choice of operation and display the result.

Pointers

- They simply represent the address in the memory where the variable is stored.
- For example, every shopping mall has an address. And who ever knows the address can access this mall.
- A pointer datatype is identified by pre-pending any datatype with a $*$ symbol. For example: `int* num1Ptr;`
- Since it is an address, we need to first go to the memory location to access the data stored there. This is called de-referencing.
- We use the $*$ symbol to dereference a pointer and access its data. For example:

```
int num1 = 10;
int* num1Ptr = &num1; // & is used to get the address of the variable
cout << *num1Ptr << endl;
```

Pointers - Example

- cd to *pointers* directory.
- There are two example's for pointers.
- The *simpleExample.cpp* demonstrates how to initialize and use pointers.
- The *userDetails.cpp* demonstrates how parameters can be passed by pointers to functions. It also shows how to set default values for function parameters.