COMP 322: Introduction to C++

Chad Zammar, PhD
Jan 15, 2021

Lecture 2
(C++ basics)

- Quick recap

- Data types

- Flow Control

- Standard input & output

- Namespaces

- Functions

```cpp
/*
=============================
 Name        : helloWorld.cpp
 Author      : Chad
 Description : Hello World in C++
=============================
*/

#include <iostream>

using namespace std;

// main function
int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

- /* … */ block comment

- // single line comment

- # … preprocessor command

  - #include <iostream> dumps in the content of iostream

- using namespace std

  - Means using namespace std :)

- main() is entry point function

- Operator << to write to cout object

- endl: end line and flush stream

- return 0; to signal that code has completed successfully

Data Types

- **Same as C**
    - **int :** sizeof(int) = 4 (4 at least, on 64-bit system)
    - **float :** sizeof(float) = 4 (usually 4, which is a 32-bit floating point type)
    - **char :** sizeof(char) = 1
    - **double :** sizeof(double) = 8 (which is a 64-bit floating point type)

    Sizeof results may vary depending on compiler and operating system (32-bit vs 64-bit)

Data Types

- **C++ only**
  - **string :** sizeof(string) = 8 in general on a 64-bit system
  - **bool :** sizeof(bool) = 1 in general but it is implementation dependent so might differ from 1
  - **auto (since C++11):** type automatically deduced from initializer
    - Do not confuse with C auto modifier which is the default for all local variables

  Sizeof results may vary depending on compiler and operating system (32-bit vs 64-bit)

Operator review

- Assignment operator (=)
  - To assign (from right to left) a value to a variable
  - int x = 42;
- Mathematical operators (+, -, *, /, %)
  - Arithmetical operations: add, subtract, multiply, divide, modulo
  - int x = 13%3;
- Relational operators (==, !=, <, <=, >, >=)
  - Test based on comparison
- Logical operators (&&, ||, !)
  - AND, OR, NOT
- Bitwise operators (&, |, ~, ^, <<, >>)
  - AND, OR, NOT, XOR, left shift, right shift

Flow control

- Conditional execution
    - if (*condition*) … else …
    - switch (*expression*) case *constant* …
- Loops (iterate over the same code multiple times)
    - for (*initialization; condition/termination; increment/decrement*)
    - for (element:array)
    - while (*condition*) { … }
    - do { … } while (*condition*)

Relational or logical operators can be used to evaluate conditions

Flow Control (conditional)

if else

```
1  if (x==25)
2  {
3      // do something with 25
4  }
5  else if (x==50)
6  {
7      // do something with 50
8  }
9  else
10 {
11     // do something else
12 }
13
14
```

- Expression can be anything
- Condition can be anything (>, <, =, etc)
- Condition values can be placed in any order
- Can be nested

switch case

```
1  switch (x)
2  {
3      case 25:
4          // do something with 25
5          break;
6      case 50:
7          // do something with 50
8          break;
9      default:
10         // do something else
11 }
12
```

- Expression must be int or char
- Condition is restricted to =
- Case values can be placed in any order
- Can be nested

Flow control

Conditional operator ?: (also called ternary operator because it takes 3 operands)

condition ? expression : expression

max = (x > y) ? x : y;

Equivalent to the following *if else* statement:

```
if (x>y)
    max = x;
else
    max = y;
```

Flow Control (looping)

for

```
for (int x=0; x<10; x++)
{
    cout<<x<<", ";
}
```

- Check condition before executing the body

while

```
int x=0;
while(x<10)
{
    cout<<x<<", ";
    x++;
}
```

- Check condition before executing the body

do while

```
int x=0;
do
{
    cout<<x<<", ";
    x++;
} while(x<10);
```

- Executes body at least once before checking the condition

Flow Control (looping)

More on for loops

- If x was already initialized, you can:

```cpp
int x=0;
for (;x<10; x++)
{
    cout<<x<<", ";
}
```

- Range for loops (since C++11):

```cpp
int a[] = {0,1,2,3,4,5,6,7,8,9};
for (auto x : a) // for each x in a
cout << x << endl;
```

Quiz

- What would the following for loop do?
  - for(; ;)

Quiz

- What would the following for loop do?
  - for(; ;)

Infinite loop

Standard input/output

- C++ uses "streams" for reading from (input) and writing to (output) a media
  - Media can be a keyboard, screen, file, printer, etc.
- Input and output streams are provided by the iostream header file
  - #include <iostream>
- cout stream object is used to print on screen
  - cout << "some message";
  - <<: insertion operator
- Default standard output is the screen
- Similar to printf() in c, system.out.println() in java

Standard input/output

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello";
    cout << "Class";
}
```

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello" << "Class";
}
```

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello" << endl << "Class";
}
```

Output:
HelloClass

Output:
HelloClass

Output:
Hello
Class

Standard input/output

```cpp
#include <iostream>

using namespace std;

int main()
{
    string var = "Hello Class";
    cout << var << endl;
}
```

Output:
Hello Class

Standard input/output

- **cin stream object is used to read from the keyboard**
  - **cin >> x;**
  - **>>: extraction operator**
- **Cin can read strings but limited to one word**
  - **cin >> stringVariable;**
- **Use getline function to read a full sentence**
  - **getline(cin, stringVariable);**
- **Similar to scanf() in c, scanner class in java**

Standard input/output

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

int main()
{
    string var;
    cout << "Please enter your name" << endl;
    cin >> var;

    cout << "your name is: " << var;
}
```

Namespaces

- A name can represent only one variable within the same scope
- Large projects consists of multiple modules of code provided by different programmers
    - What happens if one module has a variable name that is the same as another variable in different module? Name conflict (also called name collision)
- Namespaces solve the name conflict problem

Namespaces

QuebecTemp.h

```cpp
namespace QC
{
    double getTemp()
    {
        return -30.7;
    }
}
```

main.cpp

```cpp
#include <iostream>
#include "QuebecTemp.h"

int main() {
    std::cout << "Temperature is: " << QC::getTemp() << std::endl;
    return 0;
}
```

Or also: main.cpp

```cpp
#include <iostream>
#include "QuebecTemp.h"

using namespace QC;

int main() {
    std::cout << "Temperature is: " << getTemp() << std::endl;
    return 0;
}
```

Functions

- **Same as in C and java**
- **Should be declared before being used**
- **Declaration should include the name, return type and arguments type**
  - **Also called prototype or signature of a function**
- **If the function doesn't return a value, its return type should be declared *void***
- **Functions can be recursive**

Recursive Function: example

```cpp
 9  #include <iostream>
10  using namespace std;
11
12  // function declaration
13  int factorial(int nbre);
14
15  // main function
16  int main()
17  {
18      cout<<factorial(5);
19      return 0;
20  }
21
22  // function definition
23  int factorial(int nbre)
24  {
25      if (nbre<=1)
26          return 1;
27      else
28          return nbre*factorial(nbre-1);
29  }
```

The factorial function in this example is not optimal because it is not "tail-recursive". Can you rewrite it in a more optimal way?

Factorial is the number of permutations for a set of objects.

Quiz

- **Rewrite the factorial function but in an iterative (non-recursive) fashion.**

Quiz

- **Rewrite the factorial function but in an iterative (non-recursive) fashion.**

```cpp
#include <iostream>

int factorial(int i);

int main()
{
  std::cout << factorial(4);
}

int factorial(int i)
{
    int fact = i;
    for (int j=i-1; j>1; j--)
    {
        fact = fact*j;
    }
    return fact;
}
```

Function overloading

- **What's the output of the following code?**

```cpp
#include <iostream>

int absValue(int i);

int main()
{
  std::cout << absValue(-4.3);
}

int absValue(int i)
{
    if (i>=0)
        return i;
    else
        return -i;
}
```

Function overloading

- **What's the output of the following code? (answer is 4 because of implicit conversion from double to int)**

```cpp
#include <iostream>

int absValue(int i);

int main()
{
  std::cout << absValue(-4.3);
}

int absValue(int i)
{
    if (i>=0)
        return i;
    else
        return -i;
}
```

Function overloading

- **Multiple functions may have the same name but different number of arguments**
  - **Int max(int i, int j);**
  - **Int max(int i, int j, int k);**
- **Multiple functions may have the same name and same number of arguments but different types**
  - **Int max(int i, int j);**
  - **float max(float i, float j);**
- **Changing only the return type is not enough**

Function overloading

```cpp
int absValue(int i);
double absValue(double i);

int main()
{
  std::cout << absValue(-4.3);
}

int absValue(int i)
{
    if (i>=0)
        return i;
    else
        return -i;
}

double absValue(double i)
{
    if (i>=0)
        return i;
    else
        return -i;
}
```

Quiz

- **Rewrite the absolute value function from previous example using the ternary operator ?:**

Quiz

- **Rewrite the absolute value function from previous example using the ternary operator ?:**

```cpp
int absValue(int i);
double absValue(double i);

int main()
{
  std::cout << absValue(-4.9);
}

int absValue(int i)
{
    return i>=0?i:-i;
}

double absValue(double i)
{
    return i>=0?i:-i;
}
```