Week 2: Introduction to C++

CSCI 2100 Data Structures
Fall 2017

Tae-Hyuk (Ted) Ahn
Department of Computer Science
Program of Bioinformatics and Computational Biology
Saint Louis University
For Loops

```
for (int count = 10; count > 0; count--)
    cout << count << endl;
cout << "Blastoff!" << endl;
```

Notes:
- int declaration isn’t required if the variable was already declared before.
#include <iostream>
using namespace std;

int main ()
{
    for (int n=0; n<10; ++n) {
        cout << n << "", ";
    }
    cout << "Done!" << endl;
    for (int n=0; n<10; n++) {
        cout << n << "", ";
    }
    cout << "Done!" << endl;

    int n1=1;
    int n2=++n1;
    int n3=n1++;

    cout << "n1=" << n1 << ", n2=" << n2 << ", n3=" << n3 << endl;

    return 0;
}
Defining a Function

- Every C++ program has at least one function, which is `main()`, and all the most trivial programs can define additional functions.

Defining a Function

The general form of a C++ function definition is as follows:

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

A C++ function definition consists of a function header and a function body.

**Return Type** - A function may return a value. The return type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return type is the keyword `void`. 
// function returning the max between two numbers
int max(int num1, int num2) {
    // local variable declaration
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
Calling a Function

// function returning the max between two numbers
int max(int num1, int num2) {
    // local variable declaration
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}

int main () {
    // local variable declaration:
    int a = 100;
    int b = 200;
    int ret;

    // calling a function to get max value.
    ret = max(a, b);
    cout << "Max value is : " << ret << endl;

    return 0;
}
Function Declaration

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts:

```
return_type function_name( parameter list );
```

Example:

```
int max(int num1, int num2);
```

Function declaration is required when you define a function in one source file and you call that function in another file.
Necessary libraries

- Technically, streams are not automatically available in C++. Rather, they are included from one of the standard libraries.

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

<table>
<thead>
<tr>
<th>Class</th>
<th>Purpose</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>istream</td>
<td>Parent class for all input streams</td>
<td>&lt;iostream&gt;</td>
</tr>
<tr>
<td>ostream</td>
<td>Parent class for all output streams</td>
<td>&lt;iostream&gt;</td>
</tr>
<tr>
<td>istream</td>
<td>Parent class for streams that can process input and output</td>
<td>&lt;iostream&gt;</td>
</tr>
<tr>
<td>ifstream</td>
<td>Input file stream</td>
<td>&lt;fstream&gt;</td>
</tr>
<tr>
<td>ofstream</td>
<td>Output file stream</td>
<td>&lt;fstream&gt;</td>
</tr>
<tr>
<td>fstream</td>
<td>Input/output file stream</td>
<td>&lt;fstream&gt;</td>
</tr>
<tr>
<td>istringstream</td>
<td>String stream for input</td>
<td>&lt;sstream&gt;</td>
</tr>
<tr>
<td>ostringstream</td>
<td>String stream for output</td>
<td>&lt;sstream&gt;</td>
</tr>
<tr>
<td>stringstream</td>
<td>String stream for input and output</td>
<td>&lt;sstream&gt;</td>
</tr>
</tbody>
</table>
Input and Output

- The `cout` identifier represents a special output stream used to display information to the user console.
- The `<<` symbol is an operator for inserting data into that output stream.
- We use the `cin` object to read input from the user console.
- The `>>` operator is used to extract information from the stream into a variable.
Output

Python

1  print "Hello"
2  print  # blank line
3  print "Hello," first
4  print first, last  # automatic space
5
6  print str(total) + "."  # no space
7  print "Wait...",  # space; no newline
8  print "Done"

C++

1  cout << "Hello" << endl;
2  cout << endl;  // blank line
3  cout << "Hello, " << first << endl;
4  cout << first << " " << last << endl;
5  cout << total << endl;
6  cout << total << "." << endl;
7  cout << "Wait... ";  // no newline
8  cout << "Done" << endl;

Formatting

```
cout << "pi is " << fixed << setprecision(3) << pi << endl;
```

This command would result in the output pi is 3.142.

```
cout << left << setw(10) << item << " " << right << setw(5) << quantity << endl;
```

we get a result of

<table>
<thead>
<tr>
<th>Pencil</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pen</td>
<td>100</td>
</tr>
</tbody>
</table>
Console Input

```python
person = raw_input('What is your name?')
```

```c
int number;
cout << "Enter a number from 1 to 10: "; // prompt without newline
cin >> number; // read an integer from the user
```

- Notes:
  - inputs are separated by any white space
  - type of input must match type of variable

```c
string person;
cout << "What is your name? "; // prompts the user (without a newline)
getline(cin, person); // stores result directly in variable 'person'
```
Console Input

- **Problem:**

```cpp
string person;
cout << "What is your name? "; // prompt the user (without a newline)
cin >> person; // input the response
```

```
What is your name? Ted Ahn
```

- After executing the C++ code, the variable person will be assigned the string "Ted", while the subsequent characters ("Ahn\n") remain on the stream.

- **getline** : function to save the string up to the next new line

```cpp
string person;
cout << "What is your name? "; // prompts the user (without a newline)
getline(cin, person); // stores result directly in variable 'person'
```
The problem is that after executing the above code, the variable food will be set to the empty string "".

How to solve this problem? Add a line “cin.ignore();” after “cin >> age;”
File Streams

If the name of an existing file is known:

```cpp
ifstream mydata("scores.txt");
```

If the name of an existing file is unknown:

```cpp
ifstream mydata;
string filename;
cout << "What file? ";
cin >> filename;
mydata.open(filename.c_str()); // parameter to open must be a C–style string
```

Note: Writing a file default: overwrite.
To append:

```cpp
ofstream datastream("scores.txt", ios::app);
```
String Streams

- Casting between numbers and strings

```cpp
int age(42);
string displayedAge;
stringstream ss;
ss << age; // insert the integer representation into the stream
ss >> displayedAge; // extract the resulting string from the stream
```
Classes

● A **class** in C++ is a user defined type or data structure declared with keyword class that has data and functions (also called methods) as its members whose access is governed by the three access specifiers *private, protected or public* (by default access to members of a class is private).
  - The private members are not accessible outside the class; they can be accessed only through methods of the class.
  - The public members form an interface to the class and are accessible outside the class.

● Instances of a class data type are known as objects and can contain member variables, constants, member functions, and overloaded operators defined by the programmer.
#include <iostream>
#include <string>
using namespace std;

class person
{
    public:
        string name;
        int age;
};

int main()
{
    person a, b;
    a.name = "Jason";
    b.name = "John";
    a.age = 22;
    b.age = 20;
    cout << a.name << ": " << a.age << endl;
    cout << b.name << ": " << b.age << endl;
    return 0;
}
class Point {
private:
  double _x; // explicit declaration of data members
  double _y;

public:
  Point( ) : _x(0), _y(0) { } // constructor

  double getX( ) const { // accessor
    return _x;
  }

  void setX(double val) { // mutator
    _x = val;
  }

  double getY( ) const { // accessor
    return _y;
  }

  void setY(double val) { // mutator
    _y = val;
  }
}; // end of Point class (semicolon is required)
Explicit declaration of data members

- The issue of static typing arises prominently in a class definition as all data members must be explicitly declared.
- Recall that in Python, attributes of a class were simply introduced by assignment statements within the body of the constructor.
- In our C++ example, we explicitly declare the two data members at lines 3 and 4.
Access Control

- **public** aspects are those that we expect other programmers to rely upon, while private ones are considered to be internal implementation details that are subject to change.

- The compiler enforces these designations within the rest of the project, ensuring that the **private** members are not directly accessed by any code other than our class definition.
Other Issue

- What about data that main can’t have, but child classes should?
  ⇒ Protected

If the inheritance is **public**, everything that is aware of Base and Child is also aware that Child inherits from Base.

If the inheritance is **protected**, only Child, and its children, are aware that they inherit from Base.

If the inheritance is **private**, no one other than Child is aware of the inheritance.
Constructor

- The line begins with the name of the class itself (i.e., Point) followed by parentheses.
- The constructor is a function, with this particular example accepting zero parameters.
- No designated return value in the signature (not even void).
- The next piece of syntax is the colon followed by _x(0), _y(0). This is what is known as an initializer list in C++.
- It is the preferred way to establish initial values for the attributes (we are not allowed to express initial values on lines 3 and 4).
- Finally, we see the syntax { }. This is technically the body of the constructor.
- Some classes use the constructor body to perform more intricate initializations. In this case, having already initialized the two variables, there is nothing else for us to do. So the { } serves syntactically as a placeholder for the function body (somewhat like pass in Python).

```
Point(): _x(0), _y(0) { }

Point(double initX=0.0, double initY=0.0): _x(initX), -y(initY) { }
```
Example: Square Class

class Square : public Rectangle {

public:
    Square(double size=10, Point center=Point( )) :
        Rectangle(size, size, center)  // parent constructor
    {}  

    void setHeight(double h) { setSize(h); }
    void setWidth(double w) { setSize(w); }

    void setSize(double size) {
        Rectangle::setWidth(size);  // make sure to invoke PARENT version
        Rectangle::setHeight(size);  // make sure to invoke PARENT version
    }

    double getSize() const { return getWidth(); }
};  // end of Square
Implicit Self Reference

- No Self! Can just use _x or _y immediately scopes to the class attributes
- There is a “this”, but its usage is a bit more complex
- Technically, this is a pointer variable.
- Access control: public versus private (enforced by compiler)
Designating accessors versus mutators

- Access vs Mutator
- In C++ by explicitly placing the keyword `const` for accessors at the end of the function signature but before the body.

```cpp
double getX() const {return _x;}
double setX(double val) {_x = val;}
```
Inheritance

● What is inheritance?
  ▪ Creating a “child” object class that steals data/methods from parent class

● Ex: Fillable shape – Circle
  – Rectangle – Square
  – Triangle
Object Oriented Concept

- Objects of the program interact by sending messages to each other
C++ Data Types

C++ Data Types

simple

integral
cchar  short  int  long  bool
enum
floating
float  double  long double
structured
array  struct  union  class
address
pointer  reference
Recall that

char str[8];

- **str** is the base address of the array.
- We say **str** is a pointer because its value is an address.
- It is a pointer constant because the value of **str** itself cannot be changed by assignment. It “points” to the memory location of a char.

```
6000

<table>
<thead>
<tr>
<th>'H'</th>
<th>'e'</th>
<th>'l'</th>
<th>'l'</th>
<th>'o'</th>
<th>'\0'</th>
</tr>
</thead>
<tbody>
<tr>
<td>str[0]</td>
<td>[1]</td>
<td>[2]</td>
<td>[3]</td>
<td>[4]</td>
<td>[5]</td>
</tr>
</tbody>
</table>
```
Addresses in Memory

- When a variable is declared, enough memory to hold a value of that type is allocated for it at an unused memory location. This is the address of the variable.

```c
int x;
float number;
char ch;
```
The address of a non-array variable can be obtained by using the `address-of operator` &

```cpp
int x;
float number;
char ch;

cout << "Address of x is " << &x << endl;
cout << "Address of number is " << &number << endl;
cout << "Address of ch is " << &ch << endl;
```
What is a pointer variable?

- A pointer variable is a variable whose value is the address of a location in memory.

- To declare a pointer variable, you must specify the type of value that the pointer will point to, for example,

```c
int* ptr; // ptr will hold the address of an int
char* q;   // q will hold the address of a char
```
Using a Pointer Variable

```
int x;
x = 12;
int* ptr;
ptr = &x;
```

**NOTE:** Because ptr holds the address of x, we say that ptr "points to" x
int x;
x = 12;

int* ptr;
ptr = &x;

cout << *ptr;

**NOTE:** The value pointed to by ptr is denoted by *ptr
Reference Variables

Reference variable = alias for another variable
- Contains the address of a variable (like a pointer)
- No need to perform any dereferencing (unlike a pointer)
- Must be initialized when it is declared

```cpp
int x = 5;
int &z = x;    // z is another name for x
int &y;       // Error: reference must be initialized

cout << x << endl; -> prints 5
cout << z << endl; -> prints 5

z = 9;        // same as x = 9;

cout << x << endl; -> prints 9
cout << z << endl; -> prints 9
```
Pointer vs Reference variable

- Difference in Reference variable and pointer variable
- References are generally implemented using pointers. A reference is same object, just with a different name and reference must refer to an object. Since references can’t be NULL, they are safer to use.
- A pointer can be re-assigned while reference cannot, and must be assigned at initialization only.
- Pointer can be assigned NULL directly, whereas reference cannot.
- Pointers can iterate over an array, we can use ++ to go to the next item that a pointer is pointing to.
- A pointer is a variable that holds a memory address. A reference has the same memory address as the item it references.
- A pointer to a class/struct uses ‘-‘(arrow operator) to access it’s members whereas a reference uses a ‘.’(dot operator)
- A pointer needs to be dereferenced with * to access the memory location it points to, whereas a reference can be used directly.