2100: Computing Foundations I

From C to C++ : Recap and practical examples

Tamara Schneider
Jorge Reyes-Silveyra

Fall 2012

Hello World!

C

```c
#include <stdio.h>

int main()
{
  printf("Hello World in C \n");
  return 0;
}
```

hello_world.c

C++

```cpp
#include <iostream>

using namespace std;

int main()
{
  cout << "Hello World in C++" << endl;
  return 0;
}
```

hello_world.cpp

Compiling C++ Programs vs. C Programs

C

- Compile
  gcc hello_world.c -o helloworld
- Execute
  ./helloworld
  >> Hello World in C

C++

- Compile
  g++ hello_world.cpp -o helloworld
- Execute
  ./helloworld
  >> Hello World in C++

Namespaces

```cpp
#include <iostream>

using namespace std;

namespace space1 {
  int x = 5;
}

namespace space2 {
  int x = 6;
}

int main()
{

  cout << space1::x << endl;
  cout << space2::x << endl;

}
```

- Structure a program into logical units
  - Classes, objects, functions, etc.
- "using namespace" sets namespace
- If namespace "std" is not used, it has to be referenced explicitly
  - std::cout
  - std::endl

C-Strings

```cpp
#include <iostream>
#include <string>

int main()
{
   using namespace std;

   string string1 = "first string";
   string string2 = "second string";

   string string3 = string1 + " " + string2; // concatenation
   cout << string3 << endl;

// string4 = "hello " + "world"; // COMPILATION ERROR!

   string string4;
   string4 += "Hello ";
   string4 += "World!";

   cout << string4 << endl;

   // get a character of a string; remember that we start counting at 0
   cout << "Character 3 of " << string4 << " is " << string4[2] << endl;
   cout << "The size of " << string4 << " is " << string4.size() << endl;
}
```

Input and Output: Command Line

```cpp
#include <iostream>
#include <string>

int main()
{

using namespace std;

    string my_text;
    cout << "Type something ";
    cin >> my_text;
    cout << "You typed the following: " << my_text <<endl;

    getline(cin, my_text); // avoids that space cuts off string
}
```

- Use cout for output and cin for input.
- endl ends the current line.
- "Regular" cin only reads up to the first space.

Input and Output : Files

```
#include <iostream>
#include <fstream>

/* Copy one file to another */
using namespace std;

int main()
{
  ifstream input; // input stream
  ofstream output; // output stream
  input.open("from_file.txt", ios::in); //open input stream
  output.open("to_file.txt", ios::out);  //open output stream
  output << "This is a copy of the original file" << endl; //add line
  output << input.rdbuf();   // get buffer object
  input.close();   // close input stream
  output.close();  // close output stream

}
```

Object-Oriented Programming

- Objects contain attributes and functions, which belong just to the objects themselves.
  - Example:
    - A rectangle can have "length" and "width" as attributes.
    - It might have "computeArea" as a function. It can only compute its own area, not that of any other rectangle object.
- Classes can **inherit** from other classes
- Classes can be **friends** with other classes
  - A friend class can directly access private attributes and functions.

C++ Classes [1]

Geometry.h

```
#include <string>

using namespace std;

class Geometry
{
  private:
        int area;
        int circumference;
        string name;

  public:
        Geometry(string name);
        int get_area();
        int get_circumference();
        void print_area();
        void print_circumference();
};
```

header: It is good practice to declare members of class in a **separate** header file.

private: Members in this section can only be accessed from within the class, unless another class is declared as a friend. Then the friend class can access private attributes and functions.

Constructor: It must have same name as the class and is used to create an object of type Geometry.

C++ Classes [2]

Geometry.h

```
#include <string>

using namespace std;

class Geometry
{
  private:
        int area;
        int circumference;
        string name;

  public:
        Geometry(string name);
        int get_area();
        int get_circumference();
        void set_area(int area);
        void print_area();
        void print_circumference();
};
```

Geometry.cpp

```
#include "Geometry.h"

Geometry::Geometry(string name)
{
  this->name = name;
};

Geometry::print_area()
{
  cout << "The area of the geometry ";
  cout << name << " is " + area << endl;
};
```

constructor: called when an object generated, e.g. Geometry(string name)

"this" pointer corresponds generally to the class.

C++ Classes [3]

Geometry.h

```
#include <string>

using namespace std;

class Geometry
{
  private:
        int area;
        int circumference;
        string name;

  public:
        Geometry(string name);
        int get_area();
        int get_circumference();
        void set_area(int area);
        void print_area();
        void print_circumference();
};
```

Main.cpp

```
#include "Geometry.h"

int main(){

   Geometry my_geom("my new geometry");
   my_geom.set_area(10);
   my_geom.print_area();
}
```

```
> g++ *.cpp
> ./a.out
> The area of the geometry my new geometry is 10
```

C++ Classes - Inheritance

```
class Rectangle: public Geometry
{
...
}
```

- Rectangle inherits the members of Geometry.
- We can add additional members.
- That way some functions can be implemented in Geometry and used by inheritance by multiple classes, e.g. Rectangle, Triangle, Circle, etc.
  - May need to define some functions in parent class as "virtual".

Useful Examples – Using Classes [1]

Geometry.h

```
#include <string>

using namespace std;

class Geometry
{
   private:
          int area;
          int circumference;
          string name;

   public:
          Geometry(string name);
          int get_area();
          int get_circumference();
          void set_area(int area);
          void print_area();
          void print_circumference();
};
```

Main.cpp

```
#include "Geometry.h"
#include <sstream>

const int MAX = 10;

int main(){

   /* Create a geometry object and set its area */
   Geometry my_geom1("First Geometry");
   my_geom1.set_area(10);

   /* Create another geometry object */
   Geometry my_geom2("Second Geometry");
   my_geom2.set_area(234);

   /* Print areas. Each object only "knows" its
      own area */
   my_geom1.print_area();
   my_geom2.print_area();

   /* But what if we need to use pointers…? */
```

Useful Examples – Using Classes [2]

Geometry.h

```
#include <string>

using namespace std;

class Geometry
{
   private:
          int area;
          int circumference;
          string name;

   public:
          Geometry(string name);
          int get_area();
          int get_circumference();
          void set_area(int area);
          void print_area();
          void print_circumference();
};
```

Main.cpp

```
   /* But what if we need to use pointers?
      Note how member variables and functions are
      accessed! */

   Geometry* my_geom3 = new Geometry("Third Geometry");
   my_geom3->set_area(83);
   my_geom3->print_area();


   /* How about an array of geometries? */
```

14

Useful Examples – Using Classes [3]

Geometry.h

```
#include <string>

using namespace std;

class Geometry
{
   private:
          int area;
          int circumference;
          string name;

   public:
          Geometry(string name);
          int get_area();
          int get_circumference();
          void set_area(int area);
          void print_area();
          void print_circumference();
};
```

Main.cpp

```
   /* How about an array of geometries? */

   // This will not work, since the Geometry constructor
   // requires a string argument. It can only work if we
   // add a constructor Geometry().
   // Geometry geom_array[MAX];

   /* Instead create an array of pointers to Geometry
      objects. Since the compiler knows the size of a
      pointer, it can allocate space appropriately. */

   Geometry* geom_array1[MAX];

   for(int i = 0; i<MAX; i++){

    // crate a string stream (in sstream)
    std::ostringstream name;

    // concatenate integer and text
    name << i << " in array 1";

    // .str() will convert the string stream to a string
    geom_array1[i] = new Geometry(name.str());

    geom_array1[i]->set_area(i);
   }
```

Useful Examples – Using Classes [4]

Geometry.h

```
#include <string>

using namespace std;

class Geometry
{
   private:
          int area;
          int circumference;
          string name;

   public:
          Geometry(string name);
          int get_area();
          int get_circumference();
          void set_area(int area);
          void print_area();
          void print_circumference();
};
```

Main.cpp

```
   /* For dynamic memory allocation of the array you may
      want to create a pointer to an array of Geometry
      pointers. */

   Geometry **geom_array2 = new Geometry *[MAX];

   for(int i = 0; i<MAX; i++){

    // crate a string stream (in sstream)
    std::ostringstream name;

    // concatenate integer and text
    name << i << " in array 2";

    // .str() will convert the string stream to a string
    geom_array2[i] = new Geometry(name.str());
    geom_array2[i]->set_area(i*i);
   }

   /* Print out contents of both arrays */
   for(int i = 0; i<MAX; i++){
    geom_array1[i]->print_area();
    geom_array2[i]->print_area();
   }
}
```

Useful Examples – Accessing Private Class Members

- Public members can be accessed directly via "." or "->" depending on whether we are using a pointer to the object or not.
- For private members, we can implement methods to access or modify them. These functions must be public.
- OR: Sometimes it may be adequate to define a "friend class"

Geometry.h

```
class Geometry
{
   private:
          int area;
          int circumference;
          string name;

   public:
          Geometry(string name);
          int get_area();
          int get_circumference();
          void set_area(int area);
          void print_area();
          void print_circumference();

          friend class SomeOtherClass;
};
```

Geometry.cpp

```
Geometry::print_area()
{
   cout << "The area of the geometry ";
   cout << name << " is " + area << endl;

};

int Geometry::get_area()
{
   return area;
}
```

17

Useful Examples – Testing Input

```
#include <iostream>

using namespace std;

/* Example for "isdigit" to test if a character
   is a digit.
   Analogously the following can be used:
   isalpha – test if characeter is alphabetic
   isalnum – test if character is alphanumeric
*/

int main(){

    /* Some string to test on */
    string test = "abc123a";

    /* Loop through the string character by
       character and test if it is a digit */

    for(int i=0; i<test.size(); i++)
    {
          if(isdigit(test[i]))
                cout << "true" << endl;
          else
                cout << "false" << endl;
    }
    return 0;
}
```

18

Useful Examples – Splitting Strings

```
#include <iostream>
#include <sstream>

using namespace std;

/* Example for tokenizing a string, i.e. splitting a
   string with white spaces into its word components
*/

int main(){

    /* Some string to test on */
    string test = "This is a test sentence";

    /* A buffer to store the tokens */
    string buffer;

    /* The string is placed into a string stream */
    stringstream stream(test);

    /* Read word by word into buffer. Now you can
       deal with the individual words. */
    while (stream >> buffer)
            cout << buffer << endl;

    return 0;
}
```

19

Summary of Concepts

- Namespaces
- Strings
- Object-oriented programming
- Classes
- Visibility (private, public)
- Constructor
- Inheritance