An Overview of C++ and Object-Oriented Programming

Sven Chilton
The Hacker Within
UC Berkeley
Wednesday, April 22, 2015

What is C++?

- A powerful, general purpose programming language
- An offshoot of C, designed to be an improvement of it
- Compiled
- Multi-paradigm

Why care about C++?

- Performance
- System architecture
- Multi-platform

Built-in data types

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

Built-in operators

- Arithmetic (+, -, *, /, %, ++, --)
- Relational (==, !=, >, <, >=, <=)
- Logical (&&, ||, !)
- Bitwise (&, |, ^)
- Assignment (=, +=, -=, *=, /=, etc.)

C standard libraries

- Math (cmath; functions and fundamental constants)
- Strings (cstring)
- I/O (iostream, cstdio)
- Date/time (ctime)
- Memory allocation
- Many more…

Standard template library

- The STL provides a set of pre-made common classes for C++, in four categories:

- Containers (vectors, lists, sets, queues, maps…)

- Iterators

- Algorithms (sorting, searching, logical operations on data sets…)

- Functors (overloading)

Object-oriented programming

- Fundamentally, an object is a location in a computer's memory

- Equivalently, an object is an abstract data type containing data and code

- In object-oriented rather than object-based languages, objects also exhibit *polymorphism* and *inheritance*

Classes

- User defines classes for more complicated data types and functions

- Class contains *attributes* (data) and *methods* (functions), which may be public, protected, or private

- Any object which is an *instance* of a given class has access to the functions and data defined in that class

Encapsulation

- Bundling of data with methods that operate on that data
- Hiding of information within a class to prevent unauthorized access (public vs. protected vs. private)
- By encapsulating, one reduces the chances of identically named variables or functions colliding

Inheritance

- A new class may be derived from a *base class* or *parent class*. It then *inherits* the public and protected member data and functions of the base class.

- User typically defines new data and functions specific to the derived class

- Virtual functions in the base class may be overwritten in the derived class

- C++ supports multiple inheritance, i.e. classes may be derived from multiple base classes

Polymorphism

- Ad hoc, aka overloading: defining operators or functions to accept multiple types of arguments

- Parametric, aka template classes

- Subtyping, aka inheritance

Pointers

- A pointer is the memory address of some variable
- User declares a pointer with an asterisk:
  `int* p1; int *p2;`
- User obtains a variable's address with the address-of operator (&):
  `int i1 = 1; p1 = &i1;`
- To obtain the value to which a pointer points, use the dereference operator (*):
  `*p1 == 1; // This would evaluate to true`
- Demonstration: `pointerTest.cpp`

References

- A reference is an alias, another name for an existing variable

- Useful for function arguments; passing in a value (as is the default) rather than a reference to it will create a copy of that value, which can be costly in both memory and performance.

- In contrast to pointers, once your reference refers to an object, it can’t refer to another object, though the value of the original object can be changed.

The driver file

```
#include “baseClass.H”
#include “derivedClass.H”
#include <standard_header>
// This is common, but not necessary
using namespace std;

int main()
{
  // Do some stuff
  /* Do some more stuff */
  return 0;
}
```

Header file for a base class (baseClass.H)

```
class baseClass
{
  public:
    baseClass(argType arg); // Constructor
    virtual ~baseClass(); // Destructor
    virtual mDataType getData(); // Accessor
    virtual dataType1 func1(argType arg);
    virtual dataType2 func2(argType arg) = 0;
  protected:

  private:
    mDataType m_data;
};
```

Header file for a derived class (derivedClass.H)

```
class derivedClass: public baseClass
{
  public:
    derivedClass(argType arg); // Constructor
    ~derivedClass(); // Destructor
    mDataType getData(); // Accessor
    dataType1 func1(argType arg);
    dataType2 func2(argType arg);
  protected:

  private:
    mDataType m_data;
};
```

Source file for a base class (baseClass.cpp)

```
#include “baseClass.H”

baseClass::baseClass(argType arg)
{
  // Do something with the argument here
}
baseClass::~baseClass()
{
}
mDataType baseClass::getData()
{
  return m_data;
}
// etc.
```

Header file for a template class

```
template <class T>
class tempClass
{
  public:
    outputType1 func1(T& tArg);
    outputType2 func2(argType& arg);
    // etc.
};
outputType1 tempClass::func1(T& tArg)
{
  // Do some stuff with tArg
}
```

For loops

```
for (initialization; condition; increment)
  {
    // Carry out appropriate operations here
  }
```

```
for (int i=1; i<=10; i++)
  {
    cout << pow(i,2) << endl;
  }

// Above assumes using namespace std and
// #include <cmath>
```

The const flag

- Before a parameter or argument: code can’t change that quantity (within that scope)
- Before a function declaration/definition: return object can’t be changed
- After a function declaration/definition: function isn’t allowed to change any (non-mutable) class members

Compiling

On unix-like systems, g++ is the go-to compiler

```
$ g++ driver.cpp source.cpp
(Yields the executable a.out)
```

```
$ g++ driver.cpp source.cpp -o executable.exe
```

Many, MANY other options for g++

To avoid typing in a long string of options every time you compile, use a make file

Another demonstration: Sudoku

- Sit back, relax, and enjoy the show
- Alternatively, you can play around with the code on your own computer
- If you really want, you can give me unsolved sudoku grids to feed into the solver

Resources and acknowledgements

- www.cplusplus.com
- www.tutorialspoint.com/cplusplus
- Many O'Reilly books

- Thanks to Min Ragan-Kelley and Lynne Salameh for helping me prepare my demos