CSE 230
Intermediate Programming in C and C++
C++ as a Better C: Introduction to OOP

Fall 2017

Stony Brook University

Instructor: Shebuti Rayana

http://www3.cs.stonybrook.edu/~cse230/

Ref. Book: C How to Program, 8th edition by Deitel and Deitel

Introduction

- C++, an extension of C, was developed by Bjarne Stroustrup in 1979 at Bell Laboratories.
- C++ provides a number of features that "spruce up" the C language
- C++ was originally called "C with classes"
- The increment operator ++ indicates that C++ is an enhanced version of C
- The latest version is C++11 standardized through ANSI and ISO

Programming Paradigms

- C++ provides Two important programming paradigms:

1. Object Oriented Programming (OOP)
   - Classes
   - Encapsulation
   - Objects
   - Operator overloading
   - Inheritance
   - Polymorphism
2. Generic Programming
   - Function templates
   - Class templates

C++ Basics: File name in C vs C++

- In C, file names have the .c (lowercase) extension is used
- In C++, file names can have one of several extensions, such as .cpp, .cxx or .C (uppercase)
- We will use .cpp

How to run your C++ code in gcc?

g++ test.c –o test

./test

- For C++11:

g++ -std=c++11 test.c –o test

./test

Your First Code in C++: Adding Two Integers

Tells the C++ preprocessor to include the content of input/output stream header

```
1  // Fig. 15.1: fig15_01.cpp
2  // Addition program that displays the sum of two numbers.
3  #include <iostream> // allows program to perform input and output
4
5  int main()
6  {
7     int number1; // first integer to add
8
9     std::cout << "Enter first integer: "; // prompt user for data
10    std::cin >> number1; // read first integer from user into number1
11
12    int number2; // second integer to add
13    int sum; // sum of number1 and number2
14
15    std::cout << "Enter second integer: "; // prompt user for data
16    std::cin >> number2; // read second integer from user into number2
17    sum = number1 + number2; // add the numbers; store result in sum
18    std::cout << "Sum is " << sum << std::endl; // display sum; end line
19 } // end function main
```

Standard output stream object and Stream insertion operator <<

Standard input stream object Stream extraction operator >>

Stream manipulator Outputs a newline

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

std:: is required before cout, cin, endl. It specifies that we are using names that belongs to "namespace" std

Your First Code in C++: Adding Two Integers

```
1  // Fig. 15.1: fig15_01.cpp
2  // Addition program that displays the sum of two numbers.
3  #include <iostream> // allows program to perform input and output
4
5  int main()
6  {
7     int number1; // first integer to add
8
9     std::cout << "Enter first integer: "; // prompt user for data
10    std::cin >> number1; // read first integer from user into number1
11
12    int number2; // second integer to add
13    int sum; // sum of number1 and number2
14
15    std::cout << "Enter second integer: "; // prompt user for data
16    std::cin >> number2; // read second integer from user into number2
17    sum = number1 + number2; // add the numbers; store result in sum
18    std::cout << "Sum is " << sum << std::endl; // display sum; end line
19 } // end function main
```

Concatenated stream outputs, Cascade outputs of different types

```
std::cout << "Sum is " << number1 + number2 << std::endl;
```

```
Sum is 117
```

Your First Code in C++: Adding Two Integers

```
1  // Fig. 15.1: fig15_01.cpp
2  // Addition program that displays the sum of two numbers.
3  #include <iostream> // allows program to perform input and output
4
5  int main()
6  {
7     int number1; // first integer to add
8
9     std::cout << "Enter first integer: "; // prompt user for data
10    std::cin >> number1; // read first integer from user into number1
11
12    int number2; // second integer to add
13    int sum; // sum of number1 and number2
14
15    std::cout << "Enter second integer: "; // prompt user for data
16    std::cin >> number2; // read second integer from user into number2
17    sum = number1 + number2; // add the numbers; store result in sum
18    std::cout << "Sum is " << sum << std::endl; // display sum; end line
19 } // end function main
```

No return statement in main().
If execution reaches the end of main(), it automatically returns 0

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Your First Code in C++: Adding Two Integers

```
1  // Fig. 15.1: fig15_01.cpp
2  // Addition program that displays the sum of two numbers.
3  #include <iostream> // allows program to perform input and output
4
5  int main()
6  {
7     int number1; // first integer to add
8
9     std::cout << "Enter first integer: "; // prompt user for data
10    std::cin >> number1; // read first integer from user into number1
11
12    int number2; // second integer to add
13    int sum; // sum of number1 and number2
14
15    std::cout << "Enter second integer: "; // prompt user for data
16    std::cin >> number2; // read second integer from user into number2
17    sum = number1 + number2; // add the numbers; store result in sum
18    std::cout << "Sum is " << sum << std::endl; // display sum; end line
19 } // end function main
```

Operator Overloading

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

C++ Standard Library

- C++ programs consist of pieces called classes and functions.
- Most C++ programmers take advantage of the rich collections of classes and functions in the C++ Standard Library.
- Two parts to learning the C++ "world."

1. The C++ language itself, and
2. How to use the classes and functions in the C++ Standard Library.

- Many special-purpose class libraries are supplied by independent software vendors.

C++ Header Files

C++ Standard Library header file	Explanation
`<iostream>`	Contains function prototypes for the C++ standard input and standard output functions. This header file replaces header file `<iostream.h>`. This header is discussed in detail in Chapter 23, Stream Input/Output.
`<iomanip>`	Contains function prototypes for stream manipulators that format streams of data. This header file replaces header file `<iomanip.h>`. This header is used in Chapter 23.
`<cmath>`	Contains function prototypes for math library functions. This header file replaces header file `<math.h>`.
`<cstdlib>`	Contains function prototypes for conversions of numbers to text, text to numbers, memory allocation, random numbers and various other utility functions. This header file replaces header file `<stdlib>`.
`<ctime>`	Contains function prototypes and types for manipulating the time and date. This header file replaces header file `<time.h>`.
`<vector>`, `<list>`, `<deque>`, `<queue>`, `<stack>`, `<map>`, `<set>`, `<bitset>`	These header files contain classes that implement the C++ Standard Library containers. Containers store data during a program's execution.

C++ Header Files

`<cctype>`	Contains function prototypes for functions that test characters for certain properties (such as whether the character is a digit or a punctuation), and function prototypes for functions that can be used to convert lowercase letters to uppercase letters and vice versa. This header file replaces header file `<ctype.h>`.
`<cstring>`	Contains function prototypes for C-style string-processing functions. This header file replaces header file `<string.h>`.
`<typeinfo>`	Contains classes for runtime type identification (determining data types at execution time).
`<exception>`, `<stdexcept>`	These header files contain classes that are used for exception handling (discussed in Chapter 24).
`<memory>`	Contains classes and functions used by the C++ Standard Library to allocate memory to the C++ Standard Library containers. This header is used in Chapter 24.
`<fstream>`	Contains function prototypes for functions that perform input from files on disk and output to files on disk. This header file replaces header file `<fstream.h>`.
`<string>`	Contains the definition of class `string` from the C++ Standard Library.

And more …

Advantage of C++

- Creating own functions and classes
  - You will know exactly how they work
- Using existing collections of functions and classes from C++ Standard Library
  - By including suitable header files, containing the prototypes of functions and definitions of various classes
  - You can also create your custom header file
- Inline Function
  - C++ provides inline function to reduce the overhead associated with function calls
  - It is used specifically for small functions

Defining an Inline Function

```
1  // Fig. 15.3: fig15_03.cpp
2  // inline function that calculates the volume of a cube.
3  #include <iostream>
4  using std::cout;
5  using std::cin;
6  using std::endl;
7
8  // Definition of inline function cube. Definition of function appears
9  // before function is called, so a function prototype is not required.
10 // First line of function definition acts as the prototype.
11 inline double cube( const double side )
12 {
13    return side * side * side; // calculate the cube of side
14 } // end function cube
15
16 int main()
17 {
18    double sideValue; // stores value entered by user
19
20    for ( int i = 1; i <= 3; i++ )
21    {
22       cout << "\nEnter the side length of your cube: ";
23       cin >> sideValue; // read value from user
24
25       // calculate cube of sideValue and display result
26       cout << "Volume of cube with side "
27          << sideValue << " is " << cube( sideValue ) << endl;
28    }
29 } // end main
```

using namespace std;

C++ Keywords

C++ keywords

Keywords common to the C and C++ programming languages

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

C++-only keywords

and	and_eq	asm	bitand	bitor
bool	catch	class	compl	const_cast
delete	dynamic_cast	explicit	export	false
friend	inline	mutable	namespace	new
not	not_eq	operator	or	or_eq
private	protected	public	reinterpret_cast	static_cast
template	this	throw	true	try
typeid	typename	using	virtual	wchar_t
xor	xor_eq			

References and Reference Parameters

- Function arguments are passed in two ways
  - Call-by-value
- A copy of the arguments value is made
  - Call-by-reference
- Caller gives called function the ability to access caller's data directly
- Pro: No copying overhead like call-by-value for large data
- Con: called function can corrupt the caller's data
  - To indicate that a function parameter is passed by reference, we write

```
type &var_name;
```

Passing by pointer vs Passing by reference

```
void swap(int *a, int *b){
        int temp;
        temp = *a;
        *a = *b;
        *b = temp;
}

int a = 10, b = 20;
swap(&a, &b);
```

```
void swap(int &a, int &b){
        int temp;
        temp = a;
        a = b;
        b = temp;
}

int a = 10, b = 20;
swap(a, b);
```

Difference in Reference variable and pointer variable

- A reference must refer to an object. Since references can’t be NULL, they are safer to use.
- 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.

References as Alias within a function

```
int count = 1;
int &cRef = count;
cRef++;// increment count
```

- Taking address of a reference and comparing references do not create an error, each operation occurs on the variable for which the reference is an alias.
- `const int &i` is a reference to a constant integer type
- Unless it is reference to a constant a reference is an lvalue (e.g., variable name)
- It is not a constant or expression that returns an rvalue (e.g., the result of a calculation)

Returning a reference from a function

- Returning reference from a function is dangerous
- When returning reference of a variable from a function, that variable must be static
- If returned reference is for an automatic variable then the behavior is undefined
- Such references are called dangling references

Unary scope resolution operator : :

```
1  // Fig. 15.9: fig15_09.cpp
2  // Using the unary scope resolution operator.
3  #include <iostream>
4  using namespace std;
5
6  int number = 7; // global variable named number
7
8  int main()
9  {
10    double number = 10.5; // local variable named number
11
12    // display values of local and global variables
13    cout << "Local double value of number = " << number
14       << "\nGlobal int value of number = " << ::number << endl;
15 } // end main
```

Accessing the global variable

Function Overloading

- Several functions with same name, different parameter list

```
1  // Fig. 15.10: fig15_10.cpp
2  // Overloaded square functions.
3  #include <iostream>
4  using namespace std;
5
6  // function square for int values
7  int square( int x )
8  {
9     cout << "square of integer " << x << " is ";
10    return x * x;
11 } // end function square with int argument
12
13 // function square for double values
14 double square( double y )
15 {
16    cout << "square of double " << y << " is ";
17    return y * y;
18 } // end function square with double argument
19
20 int main()
21 {
22    cout << square( 7 ); // calls int version
23    cout << endl;
24    cout << square( 7.5 ); // calls double version
25    cout << endl;
26 } // end main
```

```
square of integer 7 is 49
square of double 7.5 is 56.25
```

Function Overloading

- Several functions with same name, different parameter list
- Compiler selects the proper function by number, type and order of parameters
- A common error: Creating overloaded function with identical parameter lists and different return types [gives compilation error].

Introduction to Object Technology

- *Objects*, or more precisely the classes objects come from, are essentially *reusable* software components.
  - There are date objects, time objects, audio objects, video objects, automobile objects, people objects, etc.
  - Almost any *noun* can be reasonably represented as a software object in terms of *attributes* (e.g., name, color and size) and *behaviors* (e.g., calculating, moving and communicating).
- Using a modular, object-oriented design-and-implementation approach can make software-development groups much more productive than was possible with earlier techniques
  - object-oriented programs are often easier to understand, correct and modify.

Introduction to Object Technology: Example

- The Automobile as an Object
  - Let's begin with a simple analogy.
  - Suppose you want to *drive a car and make it go faster by pressing its accelerator pedal.*
  - Before you can drive a car, someone has to *design* it.
  - A car typically begins as engineering drawings, similar to the *blueprints* that describe the design of a house.
  - Drawings include the design for an accelerator pedal.
  - Pedal *hides* from the driver the complex mechanisms that actually make the car go faster, just as the brake pedal hides the mechanisms that slow the car, and the steering wheel hides the mechanisms that turn the car.

Introduction to Object Technology: Example

- Enables people with little or no knowledge of how engines, braking and steering mechanisms work to drive a car easily.
- Before you can drive a car, it must be *built* from the engineering drawings that describe it.
- A completed car has an *actual* accelerator pedal to make the car go faster, but even that's not enough—the car won't accelerate on its own (hopefully!), so the driver must *press* the pedal to accelerate the car.

Introduction to Object Technology

Member Functions and Classes

- Performing a task in a program requires a member function
- Houses the program statements that actually perform its task.
- Hides these statements from its user, just as the accelerator pedal of a car hides from the driver the mechanisms of making the car go faster.

Introduction to Object Technology

- In C++, we create a program unit called a class to house the set of member functions that perform the class's tasks.
- A class is similar in concept to a car's engineering drawings, which house the design of an accelerator pedal, steering wheel, and so on.

Introduction to Object Technology

Instantiation

- Just as someone has to *build* a car from its engineering drawings before you can actually drive a car, you must *build an object* from a class before a program can perform the tasks that the class's methods define.
- An object is then referred to as an instance of its class.

Introduction to Object Technology

Reuse

- Just as a car's engineering drawings can be *reused* many times to build many cars, you can *reuse* a class many times to build many objects.
- Reuse of existing classes when building new classes and programs saves time and effort.

Introduction to Object Technology

Messages and Member Function Calls

- When you drive a car, pressing its gas pedal sends a *message* to the car to perform a task—that is, to go faster.
- Similarly, you *send messages to an object.*
- Each message is implemented as a member function call that tells a member function of the object to perform its task.

Introduction to Object Technology

Attributes and Data Members

- A car has *attributes*
- Color, its number of doors, the amount of gas in its tank, its current speed and its record of total miles driven (i.e., its odometer reading).
- The car's attributes are represented as part of its design in its engineering diagrams.
- Every car maintains its *own* attributes.
- Each car knows how much gas is in its own gas tank, but *not* how much is in the tanks of other cars.

Introduction to Object Technology

- An object has attributes that it carries along as it's used in a program.
- Specified as part of the object's class.
- A bank account object has a *balance attribute* that represents the amount of money in the account.
- Each bank account object knows the balance in the account it represents, but *not* the balances of the *other* accounts in the bank.
- Attributes are specified by the class's data members.

Introduction to Object Technology

Encapsulation

- Classes encapsulate (i.e., wrap) attributes and member functions into objects—an object's attributes and member functions are intimately related.
- Objects may communicate with one another, but they're normally not allowed to know how other objects are implemented—implementation details are *hidden* within the objects themselves.
- Information hiding is crucial to good software engineering.

Introduction to Object Technology

Inheritance

- A new class of objects can be created quickly and conveniently by inheritance—the new class absorbs the characteristics of an existing class, possibly customizing them and adding unique characteristics of its own.
- In our car analogy, an object of class "convertible" certainly *is an* object of the more *general* class "automobile," but more *specifically*, the roof can be raised or lowered.

Introduction to Object Technology

Object-Oriented Analysis and Design (OOAD)

- How will you create the code (i.e., the program instructions) for your programs?
- Follow a detailed analysis process for determining your project's requirements (i.e., defining *what* the system is supposed to do)
- Develop a design that satisfies them (i.e., deciding *how* the system should do it).
- Carefully review the design (and have your design reviewed by other software professionals) before writing any code.

Introduction to Object Technology

- If this process involves analyzing and designing your system from an object-oriented point of view, it's called an object-oriented analysis and design (OOAD) process.
- Languages like C++, Java are object oriented.
- Object-oriented programming (OOP) allows you to implement an object-oriented design as a working system.