Review materials are here: goo.gl/Zz2JG2

Chapter 1 - Basics

Advantages and disadvantages of C++

Advantages: vs **Disadvantages:**

--Object Oriented and Procedural
--Programs can compute very quickly!
--More user control
--Powerful STL (Standard Template Libraries)
--Code reuse

--Easily Maintained
--Lots of Jobs using C++

--Widely supported

--Steep learning curve
--It is not portable
--Requires more programmer work
--For simple programs, procedural languages are better
--Pointers and memory management
→(you need to worry about Garbage Collection)
--Compiler allows the programmer to make mistakes!
--C++ can be dangerous
→ no security/protection against overrunning the buffer and overwriting memory in adjacent locations
--Requires a strict top to bottom order

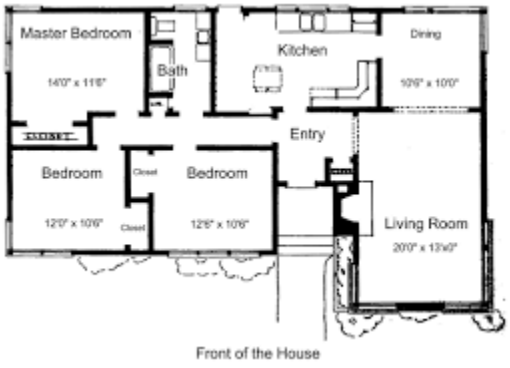
What is the difference between a class and an object?

Class

- ➔ Class is the blueprint
- ➔ Declare and define all variables/functions here

Object

- → object is the instantiation of the blueprint
- → Access functions through an object



Inheritance – use to write **reusable** code

Hierarchies – child classes inherit from the parent class

→A toyota is a type of car, therefore toyota inherits from the car class

→Triangles and rectangles are both polygons, therefore they inherit from the polygon class

Polymorphism → means having many forms

C++ allows multiple functions to have the same name:

"refers to the ability to associate many meanings to one function name"– W. Savitch

Eclipse Example – Overloading.cpp

Encapsulation (or Data Abstraction or Information Hiding)

Keep all member variables private, i.e. they are hidden from the user
- Private data cannot be accessed mistakenly by methods outside of the class
- A user or another class cannot directly change the private data; user must go through a public interface to access private data
- Details of how operations work are not known to the user

"The details of the implementation of a class are hidden from the programmers who uses the class" – W. Savitch

Input and output

```
#include <iostream>                    //used with cin and cout and fixed
Using namespace std;                   // used to replace std::cin or std::cout
```

cin >> [user input];

cout << [variable or object] << endl;

Eclipse Example – Print Decimal Places (in Review.cpp)

Chapter 3 and 4 – Functions, Paramenters and Overloading

Eclipse Example – Make Random Numbers (in Review.cpp)

What is a Function Signature?

Only includes:

1. Name of Function
2. Number and type of parameters
3. Order of parameters

What it does NOT include:

1. Return type
2. Pass by value vs pass by reference
3. Keyword const
4. Default arguments

Do these functions have the same signature?

```
 int Divide (int n, float m) ;
double Divide (float n, int m) ;
```

Function Prototypes and Declarations - Declare these at the TOP

- [Return value] [name of function] (list type of parameters - only list these if needed);

Function Definitions

- Includes the code for the function – describes what the function is supposed to do

Function Call

- Tells the function to run; it won't do anything unless called

Eclipse Example – Functions.cpp

Eclipse Example – Switch functions (in Review.cpp)

Default arguments in a function

1. Default values for the arguments are listed at the **END** of your parameter list

2. int myFunc(int a, int b=6, int c=12);

    a. Are these valid calls?
        i. myFunc(1,2,3) //Valid
        ii. myFunc(1) //Valid
        iii. myFunc() //Invalid!

Eclipse Example – Default arguments (in Review.cpp)

Passing an array as a function parameter

```
void arrayFunc(int arg[]);          //function prototype

int myArray[40];                    //declaring the array

arrayFunc(myArray);                 //calling the function with an array argument
```

Eclipse Example – Passing arrays and pointers to functions (in Review.cpp)

Reference variables

1. Reference variables must be initialized when you declare them
    a. int &refScore ; // illegal statement

int x = 5 ;
int z = x; **x→** [5] **z→** [5]

z = 6; **x→** [5] **z→** [6]

int &y = x; **x→** [5] **← y** **//y can only be used to refer to an existing integer**

y = 56 **x→** [56] **← y** **//now x=56**

Eclipse Example – reference variable (in Review.cpp)

Enumeration:

The enum is declared as: **enum** enum-type-name enum-variable;

By default, the first enumerator is assigned the integer value 0, and each subsequent enumerator has a value one greater than the previous enumerator:

```
Enum Color {
COLOR_BLACK,        // assigned 0
COLOR_RED,          // assigned 1
COLOR_BLUE,         // assigned 2
COLOR_GREEN,        // assigned 3
COLOR_WHITE,        // assigned 4
COLOR_CYAN,         // assigned 5
COLOR_YELLOW,       // assigned 6
COLOR_MAGENTA }     // assigned 7
```

Eclipse Example – enumeration (in Review.cpp)

Chapter 6 and 7 – Classes and Constructors

What do Constructors do?
- Assign data to member variables and any other 'setup' that is required such as allocating resources
- Similar to an init function

Are Constructors defined in a similar way as member functions?
- Yes, except they must have the same name as the class (Uppercase is convention for class and CTOR)
- Cannot have a return value (not even void)
- Must be public

When do Constructors get called?
- When instantiating an object, the object gets 'constructed'
    - Call to the constrcutor is made every time a new object is created

Can constructors be overloaded?
- Yes, similar to function overloading (but again, no return type)
    - Define a default constructor which takes no arguments
    - Define other constructors which take any number of arguments

Will the default constructor always be called?
- If (there are no constructors defined }{
    - -> then yes }
- else {
    - at least one constructor has been defined -> then no and you should define a default constructor}

If there is no default constructor defined, but a constructor taking 1 argument is defined as follows:

```
ThisClass {
private string response = " is the best class";
public: ThisClass (string semester) { response = semester + response;  };
```

So, is this a valid call? -- ThisClass rocks;

How is the default constructor called?
- With no parameters as an implicit call or empty paranthesis when using an explicit constructor call

What happens in the Constructor Initialization list? – See CTORInitialization.cpp
- Member variables are assigned prior to the constructor body

What does a destructor do? – Example included with static variables under WorldPeople.cpp
- "Destroys" an object as it goes out of scope
- Commonly used to de-allocate memory such as when using pointers

When are destructors called?
- It will be called automatically when an object is no longer in scope (i.e. program/function ends)
- Or when you call delete (not needed unless working with pointers)

What is a copy constructor?
- Creates an object by initializing it with an object of the same class (other object created previously)
    - Used to initialize one object from another of the same type
    - Or copy an object to pass it as an argument to a function
    - Or copy an object to return it from a function

What is the difference between an Explicit vs Implicit Constructor call?

- Refer to Jackie's slides
  - Implicit call -> examples Box b1; Box ex(1,2,3); -- both of these are implicit
  - Explicit call -> ex = Box(2,5,8);

When to use the dot . operator ?

After creating an object, you can access any of its member functions by using the dot operator

```
Car audi;
audi.getMileage();                    // now you can use the dot operator
```

When to use the scope resolution operator :: ?

1. Used in place of *using namespace std* – i.e. std::cout
   a. float f = 5.6289;
   b. std::cout << std::setprecision(5) << f << std::endl;

2. Used when functions are defined outside of the class
   a. double Box::getVolume(void) { return length*width*height; }

3. Resolve ambiguity between functions with same template which are derived from different classes
   a. See Scope_ResolveAmbiguity.cpp

4. To override the overridden function
   a. See Override_Function.cpp

5. To access static data members
   a. See StaticMemberFunctions.cpp

6. Unary scope resolution operator - Can refer to global variable within a local function
   a. int x = 5;
   b. int main() {

```
int x = 1;
cout << "The variable x: " << ::x << endl;   //  prints 5
cout <<"The variable x: " << x << endl;  // prints 1
return 0; }
```

**const modifier – used for variables when you don't want them to be changed/modified

Eclipse example for static variables – these must be initialized outside of the class

See WorldPeople.cpp and Static Members.cpp for examples of static variables

Chapter 7 –Vectors

Vectors – expandable, dynamic array

1. Good for quick random access
2. Slow to insert & erase in the middle or beginning
3. Quick to insert and erase at end

Common Vector functions – all public member functions

size() :	Return size -- there's also resize(n) which resizes vector to n
capacity() :	Returns largest # of items that can be currently stored (is a function of 2^n)
empty():	Test whether vector is empty – returns boolean value
at(index):	Access data at any position
assign():	Assign vector content – can take 1 or 2 arguments depending on version used
push_back(value):	Adds element value at the end
pop_back():	Delete last element
insert(index,value):	Insert element at specified index and puts in value
erase(position):	Erase elements at position specified (or range provided)
clear():	Clears all content; size() = 0 after executed

Example – Vectors.cpp and copy_algorithm.cpp

Chapter 8 –Operator Overloading and Friends

Operator Overloading – used to make code easier to read/maintain -- ***syntactic sugar***

Operator overloading can be done on everything except for

1. Dot operator -> .
2. Scope resolution -> ::
3. Pointer to member -> .*
4. Ternary -> ?:

The following set of operators is commonly overloaded for user-defined classes:

=	(assignment operator)
+ - *	(binary arithmetic operators)
+= -= *=	(compound assignment operators)
== !=	(comparison operators)
<< >>	(stream operators)

See Jackie's powerpoint slides on **Operator Overloading and Friends.** Make sure you understand:

1. When and how to use **friends** – (when operator overloading is a non-class member function)
    a. **friends** is *only* in function declaration or inline functions (not repeated again outside of class)
    b. If declared as a friend function, then you must have 2 arguments listed
        i. If not a a friend function, then it is assumed the **this* is on the lhs and the object you want to do something to is on the rhs (See short syntax example below)
2. Why and where to use **const** – protect parameters and return type from being modified
    a. Use when you don't want an object to be accidentally modified/overwritten
3. Must use keyword **operator** and then symbol(s) that are beng overloaded
    a. ALWAYS used at beginning of function definition (after listing the class type)
4. Why overload an operator?
    a. Because objects are user-defined data types and operators will only work on primitive or built-in data types

Syntax for overloading += operator

```
// if declared as an inline friend function
// lhs = left-hand side and rhs = right-hand side
// const was used to protect MyClass objects from being modified

friend MyClass operator+=( const MyClass& lhs, const MyClass& rhs ) {
        MyClass result;
        // code here
        return result; }

// same inline function as above, but not a friend function this time
MyClass operator+=( const MyClass& rhs ) const {
        MyClass result;
        // code here
        return result; }
```

Syntax for overloading << (insertion operator) and >> (extraction operator)

```
ostream & operator << (ostream &out, const SomeClass &someExample) {
      statements;
      return out;
}
istream & operator >> (istream &in, SomeClass &someExample) {
      statements;
      return in;
}
```

Example – OperatorOverloading.cpp

Chapter 9 –Strings

Advantages of C++ vs C strings

- C++ strings are mutable! (mutable means modifiable/can be changed)
- Copying, comparing, and assigning using = is much easier in C++
- Default constructor initializes a string object to an empty string **and** uninitialized strings are okay
- Being out of bounds will not cause a problem, but you will access garbage
    - Do not end with the null terminator '\0' like they do with C Strings

Syntax
#include <string>
string lastName; // constructs an empty string

3 ways to assign
lastName = "Davidson";
string lastName("Davidson");
string lastName = "Davidson"; // use without declaring variable above

String manipulation
lastName[0] = 'M'; // lastName = Mavidson

String concatentation

```
string firstName("Alex");
string lastName("Perez");
string fullName = firstName + lastName;        // + operator combines strings objects together
string error = "Michelle" + "!";               // cannot combine 2 C strings like this
```

More string manipulations and comparing

str[i]	Returns the ith character from the string
s1 + s2	Returns a new string of s1 concatenated with s2
s1 = s2;	Copies contents of string s2 into string s1
s1 += s2	Appends contents of s2 to end of s1
s1 == s2	Compares two strings lexographically (similar for <, <=, >, >=, !=)

String member functions

s1.length()	Returns the number of characters in the s1
s1.at(index)	Returns the character at position index
s1.substr(pos, len)	Returns substring of s1 from position pos to length len (not including value at len)
s1.find(ch, pos)	Returns first index greater than or equal to pos, containing char ch
s1.find(text, pos)	Same as above, except searches for string text instead of character ch
s1.insert(pos, text)	Inserts the characters from text before index position pos
s1.replace(pos, count, text)	Replaces count characters from text starting at position pos
s1.erase(n)	Deletes everything after nth character
s1.erase(i,j)	Deletes j number of characters starting at index i

How do we get delimited inputs from a user at once?

cin uses whitespace as a delimiter between inputs, can be a problem when reading string input from user

- Use getline() function
- getline(source, destination) where source is cin and destination is the variable to store string into
- Can grab multiple lines at once by using a delimiter character with getline
    - getline(cin, str, '?') where ? is the delimiter for end of input
    - OR getline(cin, str, ',') for comma separated values

C++ strings should usually be passed by reference (string &s1) and if not modified within the function they should be passed as a const (const string &s1)

Example – Strings.cpp

Data Types

char 1 byte	**bool** 1 byte	**short** 2 bytes	**int** 4 bytes	**unsigned int** 4 bytes
float 4 bytes	**long** 8 bytes	**double** 8 bytes	**long double** 16 bytes	

Access Control:

Private	Protected	Public
Class members declared as **private** can be used only by member functions and friends (classes or functions) of the class.	Class members declared as **protected** can be used by member functions and friends (classes or functions) of the class. Additionally, they can be used by classes derived from the class.	Class members declared as **public** can be used by any function.

Review materials are here: goo.gl/Zz2JG2