CMSC 341

C++ Review

Overview

- Big 4
- Extra Constructor Syntax and Accessors
- Call-by-Value vs. Call-by-Reference
- Using Const
- Dynamic Allocation
- Templates
- Object Relationships
- Inheritance
- Overloading vs. Overriding
- Exceptions
- Standard Template Library
- Makefiles

Big 4

- Compiler automatically provides the following methods
  - Destructor
  - Copy Constructor
  - Assignment Operator
  - Default Constructor (unless you explicitly write any constructor)
- Sometimes you can use the default behavior, sometimes you can not

Big 4

- Destructor
  - Automatically called when object goes out of scope
  - Typically frees up resources
    - For your projects: free dynamically allocated memory
    - In the real world: close open files & close network and database connections
- Copy Constructor
  - Constructs an object which is a copy of the same type of object
  - Called transparently when :
    - An object is passed by value
    - An object is returned by value

Big 4

- Assignment Operator (`operator=`)
  - Assigns one object equal to another after they have both been previously constructed
  - Applies `operator=` to each data member which may or may not be what you want
    - What about pointers/dynamically allocated memory
- Default (zero argument) Constructor
  - Provided if and only if you do not explicitly provide a constructor of your own
  - Useful so that you can treat Object as if it were a primitive
    - What if we wanted an array of Objects, but provided no default constructor?

Basic Class Syntax

```
class IntCell
{
    public:
        IntCell( ) {
            storedValue = 0;
        }

        IntCell( int initialValue ) {
            storedValue = initialValue;
        }

        int read( ) {
            return storedValue;
        }

        void write( int x ) {
            storedValue = x;
        }

    private:
        int storedValue;
};
```

Extra Constructor Syntax and Accessors

- Default Parameters
  - Can be used create multiple constructors of a method yet writing it once
  - Used to provide default values in the event parameter is not provided
- Initializer List
  - Used to directly initialize data members directly
  - Some cases it is required
  - Order needs to match order of declarations to avoid compiler errors
- `explicit` Constructors
  - Good habit to make all 1 argument constructors explicit to avoid behind the scenes type conversion

Call-by-Value vs. Call-by-Reference

- Call-by-Value:
  - Passes a **copy** of the parameter to the function as if declared as a local variable
  - Changes made in function are local to function only – you are modifying a copy
  - Can be an expression that is a parameter (i.e. 5+5)
- Call-by-Reference:
  - Passes an alias or handle to parameter to the function – references to the parameter are to the **original** variable in the calling scope
  - Changes made are in the function are on the original variable
  - Can not have an reference to something that is anonymous (i.e. not explicitly stored in a variable such as an expression)

Using Const

- Parameters and objects which are designated as `const` cannot be changed
- If a parameter doesn’t need to change prepend the parameter with `const`
- Use `const` with methods that do not need to modify any part of the class
  - i.e. accessors

Dynamic Allocation

- Objects can be dynamically allocated at run-time using `new`
  - Referenced via a pointer to the type
  - Used when things need to change size dynamically at run-time
- C++ does not have garbage collection – that means everything that is allocated using `new` needs to be freed using `delete`
  - If you allocate an array using: `foo = new int[n]`
  - then it needs to be freed using: `delete[ ] foo`

Templates

- Used heavily for container classes
  - i.e. classes that hold collections of objects
- Used to make a class or function generic
  - Don’t rewrite the same code over for different types
- For the g++ compiler, the source code and the prototypes must be in the same file
  - The easiest solution to accomplish this (and still have separate .h and .cpp files) is to `#include` the .cpp file at the bottom of the .h file
- **Never manually compile template classes**
  - It is automatically compiled by code that references it
- See IntCell / MemCell slides

Object Relationships

- "Uses a"
  - An object uses another object by calling a public method of that object
- "Has a"
  - Implemented using composition (aggregation)
  - i.e. object Foo has an object Bar as a data member
- "Is a"
  - An object builds off of a base object to extend its functionality (inheritance)
  - Typically derived class is a specialized version of its base class

Inheritance

- Single Inheritance
  - – Use when multiple objects are specific versions of some generic thing
  - – Base class / Derived class
- Multiple Inheritance
  - – Debate over worth of Multiple Inheritance
    - Some newer object oriented languages such as Java & C# for example ditched the idea (although they both support multiple interfaces)
  - – The “Diamond Problem”

Overloading vs. Overriding

- Overloading is when multiple versions (distinguishable by the parameter list – a.k.a. signature) of a method/function exist
  - foo(), foo(int), foo(char), foo(int*, string, float)
- Overriding is when a method in a base class is shadowed by a method with the same name in the subclass
  - Assuming ColorBox extends Box: then ColorBox::paint() overrides Box::paint()

Exceptions

- The **author** of a library/class can detect run-time errors, but does not in general know what to do with them
- The **user** of a library/class can cope with such errors, but can not detect them (otherwise they would have been handled in the users code and not left to the library to find)

Exceptions

- Notion of an **exception** is provided to deal with such problems
- General idea is that when a function/method encounters a problem it can not cope with, it *throws* an exception, hoping that its caller (indirectly or directly) can handle the problem

Exception Alternatives

- Terminate the program
- Return a value representing an error
  - Author does this excessively
- Return a legal value and leave the program/object in an illegal state
- Call a function to be supplied in case of an error

Exception Benefits

- Removes error handling code from the code that caused the error (less clutter)
- Makes it possible to catch all kinds of errors, errors of a certain type, or errors of related types
- Usually used in situations in where the system can recover
- Used when the error will be dealt with by a different part of the program (i.e., different scope) from that which detected the error

Exception Examples

- Throwing / Catching exceptions
  - General Form
- Grouping of exceptions
- Order of catching
- Complex exceptions

```
try {
    // code to be tried that throws an exception;
} catch (type  exception) {
    // code to be executed in case of exception
}
```

Standard Template Library (STL)

- The Standard Template Library (STL) is a general-purpose C++ library of algorithms and data structures
  - Well tested and documented
- You will most likely need to use 2 of the most common ones for this class
  - STL string improves and simplifies strings from C
  - vector acts as a dynamic array supporting operations that are a pain in C
    - vector is a template class – can use it to store anything

STL String

- size( ) – get size of string
- c_str( ) – convert from string class to array of chars
- insert( ) and erase( ) methods
- Various find( ) methods
- Various find_last/first_of( ) methods
- substr(pos, n) method – gets portion of string
- Overloaded operators
  - Assignment, equality, concatenation, subscript, etc…

STL Vector

- size( ) – returns number of elements in vector
- empty( ) – is the vector empty?
- begin( ) and end( ) – get iterators (we'll learn more about iterators as the semester progresses)
- clear( ) – empty out a vector
- Overloaded operators for equality, assignment and subscripting