Random useful stuff for C++

Structs

* usually passed by reference
* can be used to return multiple values
* example of way to use a struc and function
  + genericStruct = genericFunction(struct1, struct2)

Class

* defines a new type that can group data and functions to form an object
* public member functions
  + indicate all operations a class user can perform on the object
* declaring a variable of a class type creates an object of that type
* member access operator
  + “.” operator
  + is used to invoke a function on an object
* private data members
  + variables that functions can access but class users cannot
  + 99% of data members should be private
* function declaration
  + provides the function’s name, return type, and parameter type, but not the function’s statements
  + declare member functions after ther word “public”
* function definition
  + provides a class name, return type, parameter names and types, and the function’s statements
  + className::functionName
  + can access private data members
* Implicit parameter
  + in the function call object.Function(), the “object” part is called the implicit parameter
  + the compiler converts the call syntax object.Function(…) into a function call with a pointer to the object implicitly passed as a parameter
    - ex. Function(object, …)
  + the implicitly passed object pointer is accessible via the name *this.*
    - this->member
    - this-> is the member access operator for a pointer, similar to the “.” operator for non-pointers
* mutator
  + a function that may modify (“mutate”) a class’ data members
  + setter function
    - name starts with set
* accessor
  + a function that accesses data members but does not modify a class’ data members
  + usually defined as const
  + getter function
    - name starts with get
* private helper functions
  + is a private member function
    - any member function can call a private member function
    - user cannot call private member functions
* constructor
  + a special class member function that is called automatically when a variable of that class type is declared
  + can initialize data members
  + a constructor callable without arguments is a default constructor
  + a constructor has the same name as the class along with no return type
    - ex. GenericClass::GenericClass() {…}
  + if a class has no programmer-defined constructor, then the compiler implicitly defines a default constructor having no statements
  + you can explicitly add a default constructor
    - GenericClass() = default
* constructor initializer list
  + an alternative approach for initializing data members in a constructor
  + coming after a colon and consisting of a comma-separated list of variableName(initValue) items
  + you should always use constructor initializer lists for data members of class type that you want initialized with arguments
  + Member objects requiring constructor arguments should ALWAYS appear in the initializer list (and not in the constructor body).
  + SampleClass::SampleClass() : field1(100), field2(200) {}
* static data members
  + static
    - indicates a variable is allocated in memory only once during a program’s execution
    - reside in the program’s static memory region and have a global scope
    - can be accessed from anywhere in a program
  + a data member of the class instead of a datamember of each class object
  + are independent of any class object, and can be accessed without creating a class object
  + must be defined outside of class declaration
    - is declared with static, not defined with static
    - int genericClass::genericVariable = value
  + accessed outside of class using :: notation
    - genericClass::genericVariable
* static member function
  + is a class function that is independent of class objects
  + the this-> parameter is not passed to a static member function
  + cannot access non-static data members
* files
  + there are usually two files per class
  + ClassName.h
    - contains the class definition, including data members and member function declarations
  + ClassName.cpp
    - contains member function definitions
    - must include ClassName.h

Operator overloading

* redfining the functionality of built-in operators like +, -, \*, to operate on programmer-defined objects
* create a member function named operator+
  + ex. TimeHrMn TimeHrMn::operator+(TimeHrMn rhs)
* operator can be overloaded multiple times
  + the compiler determines which operation to invoke based on operand types
* Comparison operator overloading
  + Create function named operator== that returns bool and takes two const reference arguments of the class type for the left-hand-side and right-hand-side
  + ex. bool operator==(const Review& lhs, const Review& rhs)

Namespace

* There are three ways to access items defined in a namespace
  + using namespace std;
    - This makes all declared names in the #included headers defining that namespace visible in the program being compiled. Never do this in a header file
  + using std::cout;
    - this only makes visible the individual names specified
  + not using the using keyword and always fully qualify names

Vector

* vector< “value type”> vectorName(“vector size”)
* sort
  + #include <algorithm>
  + overload the < operator for the programmer-defined class
  + call sort() function as sort(myVector.begin(), myVector.end())

Exception-handling

* always catch exceptions by reference
* try
  + a block that surrounds normal code, which is exited immediately if a throw statement executes
* throw
  + a statement that appears within a try block; if reached, execution jumps immediately to the end of the try block.
  + provides an object of a particular type, such as an object of type “runtime\_error”
  + similar to a return statement
* catch
  + a clause that immediately follows a try block; if the catch was reached due to an exception thrown of the clause’s parameter type, the clause executes
* if an exception is thrown and not caught within the function then the function is exited
* catch(…)
  + catches all exceptions

Inheritance

* derived class
  + is a class that is derived from another class called a base class(superclass)
  + inherits the properties of the base class
  + has access to all of the public members of the base class
  + ex.
    - class DerivedClass: public BaseClass {…}
  + classes can be derived from a derived class
* Protected
  + provides access to derived classes but no one els

Overriding

* override
  + you can use the base class’s function by using override keyword

Polymorphism

* compile-time polymorphism
  + when the compiler determines which function to call at compile-time
  + overloading
* Runtime polymorphism
  + when the compiler is unable to determine which function to call at compile-time, so the determination is made while the program is running
  + derived classes
* pure virtual function

Tips&Tricks

* converting minutes to hours
  + divide minutes by 60
  + mod minutes by 60
* using const in functions
  + a function name along with its parameter list is called its signature
  + you place const after a member function’s signature
  + if you separate the declaration of a member function from its definition (you don’t declare it inline), you must use const after the singnature in both plaaces
  + const comes before the opening brace of the function body in its definition
  + a function’s signature does not include its return type
  + only member functions cane be called const
  + A const function can only call another const function
* in-class initialization occurs before the constructor runs
* Initialize variables when declared
* when overloading all equality and relational operators
  + A common approach is to first overload the == and < operators and then overload other comparison operators using == and <.
  + Overloading != using ==:
    - bool operator!=(const Review& lhs, const Review& rhs) { return !(lhs == rhs); }
  + Overloading >, <=, and >= using <:
    - bool operator>(const Review& lhs, const Review& rhs) { return rhs < lhs; }
    - bool operator<=(const Review& lhs, const Review& rhs) { return !(lhs > rhs); }
    - bool operator>=(const Review& lhs, const Review& rhs) { return !(lhs < rhs); }