Lecture 1: Creating a Class

* Objects are characterized by their behavior and state
  + Behavior: how the object reacts and what it reacts to
    - Member functions
  + State: Current state of object (ex. My grocery cart contains only apples)
    - Member data
* Example:

*class Account {*

*public:*

*void Display() { … } // member function*

*int getBalance() { return balance; } // getter*

*int setBalance(int b) { balance = b; } // setter*

*private: // only member functions can access “private” data*

*std::string number; // member data (state)*

*int balance; // member data (state)*

*}*

* Create an object like: *Account myAccount;*
  + Can access data using myAccount.number and myAccount.balance if they were public, but we use getters and setters and make data private to make it safer
* Constructors allow the class designer to guarantee that every object is properly initialized

*class Account {*

*public:*

*Account(std::string n, int b) : number(n), balance(b) {*

*…*

*}*

*Account(std::string n) : number(n), balance(0) {*

*…*

*}*

*…*

*private:*

*const std::string number;*

*int balance;*

*}*

* Implementer – How I code the class?
  + Correctness
  + Performance
  + Natural Interface
* Client – how do I use a class?
  + Interfaces
  + Correct usage protocol
* Separate the implementation from the interface
  + Use .h files which is the declaration of the function elements

Lecture 2: Introduction to classes, objects, and strings

* Classes begin with upper case letter
* Forgetting the semicolon at the end of a class definition is a syntax error
* The access-specifier label *public:* contains the keyword *public* as an access specifier

Lecture 3: Creating a Class with a Method and Parameter

* getline(cin, nameOfCourse) reads characters from the standard input stream object cin until the newline character is encountered, places the characters in the string variable *nameOfCourse* and discards the newline character

Lecture 4: Creating a Class with Data and Getters and Setters

* Attributes are represented as variables in a class definition
  + These variables are called data members and are declared inside a class definition but outside the bodies of the class’s member function definitions
  + Each object of a class maintains its own attributes in memory
* Data member: a variable that is declared in. a class definition outside the bodoes of the class’s member function definitions
  + Every instance (object) of a class contains each of the class’s data members
* All attributes of the class should be accessed with *this->*
* Variables or functions declared after the access specifier *private* are accessible oly to member functios of the class for which they are declared
* Setters = mutators
* Getters = accessors
* Always try to localize the effects of changes to a class’s data members by accessing and manipulating the data members through their corresponding *get* and *set* functions

Lecture 5: Instantiating Multiple Objects of a Class

* Constructors
  + A class can have multiple constructors since they override each other based on input
  + Constructors must have the same name as the class
  + Constructors cannot return values
  + Are normally public
* Constructor initializer list is after the : in the constructor header
  + This initializes the variable instead of doing it in the body
  + Ex. explicit GradeBook(std::string name) : courseName(name) {}
    - This initialized the variable courseName to name
* Default constructor takes no arguments
* If you define any constructor with arguments, C++ will not implicitly create a default constructor for that class, so always make default constructor’

Lecture 6: Separating a class from driver

* Programs use the *#include* preprocessing directive to include header files and take advantage of reusable software components
* Do not use “using namespace std;” inside any header file
* Compiler creates only one copy of the class’s member function
  + Shares the copy among all the class’s objects
* Each object needs tis own data members
  + Contents vary among objects
* Member-function code is not modifiable since it is shared among all objects of the class
* The size of an object depends on the amount of memory required to store the class’s data members
* Header should be in double quotes
  + Ex. #include “GradeBook.h”
* Placing a class definition in a header filer reveals the entire implementation of the class to the class’s client

Lecture 7: Separate Interface from Implementation

* Interface is .hpp file
  + Defines and standardized the ways in which things such as people and systems interact with one another
* Interface is .cpp file
  + Describes what services a class’s clients can use and how to request those services but not HOW the class carries out the services
* Better to define member functions outside the class definition
  + Hiding implantation details form the client code
  + Ensures you do not write client code that depends on the class’s implementation details
* Member function definitions are placed in a source code file
* Only put function prototypes describing the class’s interface without revealing the class’s member functions implementation in the header file
* Although parameter names in function prototypes are optional (they are ignored by the compiler), many programmers use these names for documentation purposes
* Source code file GradBook.cpp defines class GradeBook’s member functions
  + Each member function name is preceded by the class name and :: which is known as the class scope resolution operator
  + This “ties” each member function to the GradeBook class definition, which declares the class’s member functions and data members
* When compiling a .cpp file, the compiler uses the information in the associated .h file to ensure that:
  + The first line of each member function matches its prototype in the .h file
  + Each member function knows about the class’s data members and other member functions
* Order of functions in .h file do not matter

Lecture 8: Validate Data in a Setter

* Setters do validity checking (aka validation)
* Making data members private and controlling access, especially write access, to those data members through public member functions helps ensure data integrity
* X = variableName.substr(0,25) sets X to be the first 25 characters of variableName
  + Use std::cerr or throw exception if needed

Lecture 9: Intro to Control Statements

* Algorithm: procedure for solving a problem in terms of the action to execute and the order in which the action executes
* Program control: specifies the order in which statements (actions) execute in a computer program
  + Control statements provide program control
* Types of execution: sequential, selection, and repetition
  + Selection statements: if, if/else, switch
  + Repetition statements: while, do/while, for
* Activities diagrams help develop and represent the algorithms
  + Defined by UML standards
  + Transition arrows: indicate order in which the actions