CSCI 3110
ADT and C++ Class

In many cases, the solution to a problem requires operation on data, for example add, sort, print, and remove operations on data. In these cases, it is better to put the focus on data, and think in terms of what we can do with a collection of data, INDEPENDENTLY of HOW we do it. This is called **data abstraction**. Data abstraction is the fundamental idea of Object Oriented Programming.

Abstract Data Type (ADT) – a collection of data together with a set of operations on that data

ADT is different from Data Structure – data structure is a construct within a programming language that stores a collection of data. It defines how data is stored in the memory for a program.

When constructing an ADT, there are two steps involved:

- Specification – indicate precisely what each operation of an ADT does.
- Implementation –specify how each operation is implemented. This includes selecting a data structure for data involved in ADT using programming language → data structure is part of an ADT's implementation

The client program is the program that uses the ADT, ie the C++ class type defined in the specification and implementation files. In C++, an instance of a C++ class is referred to as an object. OOP emphasis creating objects in programming.

Object Oriented Programming (OOP)

Three main characteristics:

- **Encapsulation**:
  - The ability to provide users with a well-defined interface to a set of functions in a way which hides their internal workings
  - Practical rules of implementation which enhance the encapsulation of your objects:
    - Never put data in the public interface of your class
    - Create accessor methods (e.g., Get and Set methods) for all the data in your class
    - Keep helper methods out of the public interface
- **Inheritance:**
  - A way to form new classes using classes that have already been defined.
  - The new classes, known as derived classes, take over (or inherit) attributes and behavior of the pre-existing classes, which are referred to as base classes.
  - It is intended to help reuse existing code with little or no modification
- **Polymorphism**
  - The characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a function to have more than one form.
  - The essence of polymorphism is to combine encapsulation with inheritance and thus hide the details of specialization

C++ Class Basics

class, object

- ADT implemented in C++ → class
- instance of a class → object

public, private, protected, const

- **public** -- methods declared in the public section can be accessed by all client programs of the class. Do not declare data in public section

- **private** -- Data and methods declared in the private section cannot be accessed by the client program of the class. They can only be accessed by the member functions of the class itself. This is how data hiding is achieved.
- **protected** -- Protected access modifier is similar to that of private access modifiers, the difference is that the class member declared as Protected are inaccessible outside the class but they can be accessed by any subclass(derived class) of that class. (more details to come when we discuss class inheritance)
- **const** -- A const method may not modify the data of a class
  - Non-constant member functions cannot be invoked within a constant member function since they may change member data.
  - For an object passed into a user defined function as a constant parameter, only const method of the object may be invoked in the user defined function

Constructor (default constructor, other constructors)

- Using class name as its name NO return type, even void Can have parameters
- Activated automatically and implicitly when an object of the class is created
- used to initialize data of the object
- A class can have more than one constructor
  - Typically, one should define at least the default constructor, one or more value constructor, and the copy constructor

- **Default Constructor:** the constructor taking no parameters
  - The compiler generates a default constructor if NO constructor is defined
  - A compiler-generated default constructor may not initialize data members to values that you will find suitable.

```
class Date {
        Date(int year=2007, int month=1, int date=1);
        ...
};
```

Default parameter

- A default parameter is a function parameter that has a default value provided to it.
- If the user does not supply a value for this parameter, the default value will be used.
- If the user does supply a value for the default parameter, the user-supplied value is used.
- If a parameter is a default parameter, all parameter following it must be default parameters too.
  - Make default parameters appear last in function parameter list
- If the declaration and definition is separated, no need to specify the default value in the definition.

  - Initializer list can be used and preferred (separated by ,)
    - A initializer list uses a functional notation that consists of a data member name followed by its initial value enclosed in parentheses.
    - Initializer list is more efficient
    - Initializer lists can only be used with constructor
  - Like any other member functions, constructors can be public or private.

Destructor

- Destructor provides a chance to release resources the object has, such as memory, files, locks, etc.

        - Destructor is necessary when data of the class has acquired dynamically allocated memory. In this case, destructor deallocates the dynamically acquired memory when the object exits the scope
    - Destructor properties:
        - Destructor name: followed by class name NO return type, not even void
        - CanNOT have parameters CanNOT be overloaded
            - a class can only have one destructor
        - Compiler generates a default destructor if no one is defined
    - Activated automatically and implicitly when an object of the class exits its scope

method -- member function of a class
- **accessor** – get method, retrieve the data value of an object
- **mutator** – set methods, change the data value of an object

- Use **#ifndef / #define / #endif** to prevent multiple inclusion of a class definition

Examples of defining and implementing a C++ class, and client program that declares objects, or array of objects, invoking methods using dot notation.

More on C++ Class

Class member data

- A member data can be
    - instance variables: Each object/instance of the class has its own copy
    - class variables: All objects/instances of the class *share the same copy*
        static int NumOfInstanceCopy;
    - class constants: Constants shared by all instances syntax:
        static const int MinDeposit;
- Instance variables should be initialized in constructors
- Class variables and constants should be initialized outside the class, typically in the .cpp file.
    int CLASSNAME::NumOfInstanceCopy = 0;
    const int CLASSNAME::MinDeposit = 500;
- Class variables and constants exist even before any object/instance is created

Static member functions

- A static member function is a member function following the keyword static
    Syntax: static Student create();
- A static member function cannot be declared with the keywords virtual, const.
- A static member function can access only the names of static members, enumerators, and nested types of the class in which it is declared.
- Static member functions exist even before any object is created.
- Static member functions can be invoked in the following form:
    ClassName::StaticMemberFuncName(func. argument list)
- The static member functions are not used very frequently in programs. But nevertheless, they become useful whenever we need to have functions which are accessible even when the class is not instantiated.

    For example: wrap a global function in a class, or provide a function instead of constructors to create an object/instance.

Both constructor and destructor can be public or private

- If all explicitly declared constructors are private, clients cannot create an object through these constructors
- If the destructor is private, you cannot create an object of this class without using new operator.
- If the destructor is private, no object of the class can be deleted.

C++ Class Usage (in the client program)

Create an object of a class

- A class is a user-defined data type. It can be used like any other built-in types like int.
- Different ways to create an object
  - Date today; // default constructor is called
  - Date anniversary(1997, 10, 23); // constructor taking 3 int parameters is called
  - Date commencement = today; // copy constructor is called
  - Date ACMMeeting(commencement); // copy constructor is called
  - Date *currentEventDay = new Date; // default constructor is called
  - Date * currentEventDay = new Date(); // equivalent to the previous one
  - Date *anniversary = new Date(1997, 10, 23); // constructor taking 3 int parameters is called

Access members of a class

- Access public member data/functions:
  - By . (dot) operator Example: today.getYear()
  - By → (selection operator) Example: currentEventDay→getYear()
    // currentEventDay must be a pointer
- Access class variables or constants
  - By :: (scope resolution operator) Example: Date::NumOfDateObjectCopy = 5;
    // NumOfDateObjectCopy is a class variable of class Date

    currentMonth = Date::Feb;
    // Feb is either a static constant or an enumerator of class Date
  - They can also be accessed like any other "normal" public member data
- Access static member functions
  - By :: (scope resolution operator) Example: Date::getNumOfCopy()
    // getNumOfCopy is a static function of class Date
  - They can also be accessed like any other "normal" public member functions

Using Array of objects

1. Declaring an array of objects
   - Like any other data type in C++, we can define arrays of class objects.
     Example: InventoryItem inventory[40];
     Defines an array of 40 InventoryItem objects. The name of the array is inventory
     The default constructor is called for each object in the array
     If no default constructor is defined in the class InventoryItem, a compile error occurs.

- If those constructors have special needs (large and expensive memory allocation), this can be a performance bottleneck and a waste of time if not all elements are to be used.
    - Solution: Use STL vector defined in <vector>

```
vector<InventoryItem> inventory;
inventory.reserve(100);
```

2. Initialize array objects
    - Define an array of objects and initialize them by a constructor requiring parameters

```
InventoryItem inventory[3] = { InventoryItem("Hammer"),
                               InventoryItem("Wrench"),
                               InventoryItem("Pliers")};
```

    - Define an array of objects and initialize them by different constructors

```
InventoryItem inventory[3] = { InventoryItem("Hammer"),
                               InventoryItem("Wrench", 9.3, 100),
                               InventoryItem()};
```

Passing objects as function parameters

- Pass an object by value.
    - If an object is passed by value, (default) copy constructor is invoked automatically to copy the object argument to the parameter
- Pass an object by reference
    - Syntax: Appending the symbol (&) to the class type
    - The address of the object is copied.
    - Any modification through the object parameter applies to the object argument.
- Pass an object by const reference
    - Syntax: Appending the keyword “const” and symbol (&) to the class type
    - The address of the object is copied,
    - The object is NOT to be modified within the function