Introduction to Software Development – CS 6010
Lecture 15 – Classes and Objects

Master of Software Development (MSD) Program

Varun Shankar

Fall 2023

Lecture 15 – Classes and Objects

- Topics
  - Classes
    - Methods
    - Accessibility
    - Constructors
    - Getters/Setters

What is the Purpose of std::vector?

- What is the purpose of the std::vector?
    - Gives us data of a dynamic size?
        - I can do that without a vector: `new int[ size ];`
    - Does it without us (the programmer) having to:
        - Manually resize
        - Copy data
        - Deal with the heap
        - Memory leaks
        - etc
    - In other words: Hides all the gory details from the user.

Classes and Objects

- An object is a variable that provides both:
  - Data
  - Functions on that data
  - And hides the details (abstracts them away)
- String:
  - Data: List of characters
  - Functions: length(), find(), push_back(), pop_back(), substr(), etc.
- Objects are instantiations (or simply instances) of Classes.
- Classes are the “blueprints” used to declare an object.
  - Similar to `struct` but with a few added features.

MyVector

```
struct MyVector {
        int * data;
        int size, capacity;
};
MyVector v = makeVector( 10 );
push_back( v, 10 );
int value = pop_back( v );
```

- What happens if, as a user of your MyVector, I do this:

```
v.capacity *= 2; // or:
v.data++;  // I’ve moved the data pointer to the 2nd element in the array.
```

- I’ve just broken your object – because I have access to its internals.

Object Internals

- We don’t need to know how an object works on the inside to use it.
- How exactly do strings store their data? How do they calculate their length?
  - As long as they do what they advertise, we mostly don’t care.
- As creators of Classes (in other words, as creators of the blueprints for objects), we must decide upon and control the internals of our objects.
  - But we can hide all that complexity from the user of our class.
  - We control the users ability to see into our object using *access modifiers.*
  - Access Modifiers
    - public – anyone can see this piece of our class
    - private – only accessible by the class itself*
    - protected – like private, but we’ll talk about this in more detail later in this course.

MyVector – (Slightly) Better

```
struct MyVector {
    private:  // After private: no one can access the member variables from “outside” the struct
            int * data;
            int size, capacity;
};
MyVector v = makeVector( 10 );
```

- What happens if, as a user of your MyVector, I do this:

```
v.capacity *= 2; // or:
v.data++;
```

- The compiler will produce a compiler error – thus disallowing me from doing this.

Classes vs Structs

- It turns out that Structs and Classes are exactly the same thing with one minor difference:
  - Struct – by default everything in the Struct is public.
  - Class – by default everything in a Class is private.

```
class MyClass {                  ==            struct MyClass {
        public:                                                 …
                …                                               };
};
```

- In general, structs are used as containers for “plain old data” – giving each piece of data they contain a name (field).
  - They are “dumb”.
- Classes are used for more complicated types.
  - Contain related data with “invariants”
    - Which means there are rules about how the data members are related to each other.

MyVector – Data Relationships

```
class MyVector {
    private:
              int * _data; // a member variable, or a “field”
             size_t _size, _capacity; // more fields
    public:
             size_t size()  // a function inside a class is called a method (or a member function)
            {
              return _size;
            }
};
```

- What relationships between MyVector’s data?
  - size must be less than capacity
  - if size == capacity we must reallocate
  - data must point to an array with capacity elements.

Classes – Methods

- The functions provided by a Class, and that operate on the Class' data, are called *methods*.
- We have seen these before:
  - vector<int> v;
  - int s = v.size(); // .size() is a method of the class vector
- Just like the structs that we have created in previous assignments, a class is declared in a .h file.
  - This includes both the data, and the methods associated with the data.
- The methods are then defined (implemented) in the corresponding .cpp file.

Declaring a Method

```
class MyWidget {
        // Data for MyWidget.  Also called member variables or fields.
        int number;
        int weight;
        int width;

       // Methods (functions on the data)
        float determineCost();
        bool needToReorder();
};
```

- Previously we would have declared determineCost() like this:
  - float determineCost(const MyWidget & theWidget);
- Note: you can actually add methods to your structs!

Implementing a Method

- #include “MyWidget.h”
- float MyWidget::determineCost()
- {
  - …
- }
- The “MyWidget::” tells the compiler that this function is associated with the MyWidget class.
- Inside the { } you can use the member variables of the class – which refer to the member variables inside the class object that is calling the function.
- For example, if you call theWidget.determineCost(), the data within the variable theWidget will be used.

MyVector – Making It Better

```
class MyVector {
    private:
        int * _data;
        size_t  _size, _capacity;
};
size_t size( const MyVector & v ) {
        return v.size; // ERROR: size is private and thus not available to this function.
}
```

- Must turn *size()* into a method for the MyVector class

MyVector – Making It Better

```
class MyVector {
    public:  // Allows “outside” users to access these methods.
            size_t  size();  // Looks the same as a normal function, but inside a class.
    private: // Outside users cannot touch anything marked private
            int * _data;
            size_t  _size, _capacity;
};

MyVector v;
size_t s = v.size();  // size() can access the v’s size field because it is a method.
```

MyVector Implementation

-
  ```
  size_t size() {
          return _size; // ERROR
     }
  ```

-
  ```
  size_t MyVector::size() {
              return _size; // Works
     }
  ```

The – this – pointer

- *this* is a pointer to the current object.
  - It only exists within methods.
-
  ```
  size_t MyVector::size() {
          return this->size; //Redundant, can just use size
      }
  ```

Constructors

- A *Constructor* is a “function” that is used to create (construct) the object.
  - Constructors usually take in, as parameters, data that will be used in the creation of the object.
  - A class can have multiple constructors, allowing objects (of the same class) to be (initially) created in different ways.
- Constructors:
  - Must have the same name as the class.
  - Do not have return types (not even void).
- Examples:
  - MyWidget(int weight, int width); // In the .h file
  - MyWidget::MyWidget(int weight, int width) { … } // In the .cpp file

Constructors

- Constructors:
  - Usually should initialize all member variables (to some initial value)
- Using a constructor:
  - Sometimes referred to as “calling the constructor”.
  - MyWidget theWidget; // This calls the constructor with 0 parameters (if one exists)
    - This is called the *default constructor*.
  - MyWidget theWidget{}; // Same as above
  - MyWidget theWidget(); // LOGIC ERROR: this is actually declaring a function!
  - MyWidget theWidget(10, 15); // Calls the 2-parameter (2 ints in this case) constructor

Getters and Setters

- In the MyVector example:
  - size() is called a *getter*, because it provides the ability to get the value of an internal class variable, without allowing an outside user to change it.
  - void setCapacity(int cap)
    - *setter* functions are used to change an internal value.
    - So an outside user can use this function to change MyVector.capacity… what would we need to do in this function?
      - new/delete/copy the data vector to make the rest of the internal data variables match this capacity.

Const Methods

- Methods that don’t change any data within the class should be declared *const*.
- In the .h file:

```
float determineCost() const;
```

- In the .cpp file:

```
float determineCost() const {
}
```

More Reading

- https://runestone.academy/runestone/books/published/thinkcpp/Chapter14/private_data_and_classes.html

HW – Write a MyVector class!

- void set(MyVector & vec, int value);
- What does this become in a class?
- void set(int value);
  - Where does the above line go?
  - Inside the class MyVector declaration.
- Hint: When you turn your struct into a class…
  - How do you do this?
    - Just replace *struct* with *class*
- …add in *public:*
  - Why?
  - If you don’t, all of your code will break as your current functions will not be able to access the interval data (size, data, capacity) of your vector.
  - This will allow you to update one function at a time.
  - Then change the member variables to private once everything is converted.

```
class Card {
private:
  // Member Variables
  int suit_; // 0-3: 0 C, 1-D, 2-H, 3-S
  int rank_; // value of the card 2-14 (14-Ace)
public:
  // Constructors
  Card(); // Default
  Card( int suit, int rank );
  // Card( int number ); // number ranges from 0-51... 0=>2C  51=>AS
  // Card( string name ); // Use like: Card( "Queen of Diamonds" );  rank_ = 12,
suit_ = 1
  // Member Methods
  void print() const; // Just displays card to the screen
  int getSuit() const; // Returns the card's suit
  int getRank() const; // Returns the card's rank
};
```

```
Card::Card() {
  this->suit_ = rand() % 4; // 0-3
  this->rank_ = (rand() % 13) + 2; // 0-12 + 2 => 2 - 14
}
void Card::print() const {
  vector<string> suits = { "Clubs", "Diamonds", "Hearts", "Spades" };
  vector<string> facecards = { "Jack", "Queen", "King", "Ace" };

  if( rank_ <= 10 ) {
    cout << rank_;
  }
  else {
    cout << facecards[ rank_-11 ];
  }
  cout << " of " << suits[ suit_ ];
}
```

Tuesday Assignment(s)

- Code Review
- Homework (Group) – DIY Vector