Classes: An Introduction

Thomas Merkh
tmerkh55@gmail.com

Section 2, Week 7

May 12, 2020
Overview

1 Classes

2 Fundamentals of Classes

3 Next Time
Last time we discussed scope and debugging. As a refresher:

- Scopes can be nested and exist between `{ }` markers
- Debugging is useful and VS 2019 has an intensive debugger
- F5 executes until next breakpoint, F11 executes exactly 1 line at a time, and setting “watches” can help understand how variables change over time
Last time we discussed scope and debugging. As a refresher:

- Scopes can be nested and exist between `{ }` markers
- Debugging is useful and VS 2019 has an intensive debugger
- F5 executes until next breakpoint, F11 executes exactly 1 line at a time, and setting “watches” can help understand how variables change over time

Today, we have at last arrived at the topic of classes, which in my personal opinion is perhaps the most important coding construct in c++ to understand.
1. Classes

2. Fundamentals of Classes

3. Next Time
Motivation

The previous couple weeks have been structured around learning the very fundamentals of C++, meaning the syntax, control flow, and some basic objects like strings. Interesting problems call for more complex and interesting objects.
Motivation

The previous couple weeks have been structured around learning the very fundamentals of C++, meaning the syntax, control flow, and some basic objects like strings. Interesting problems call for more complex and interesting objects.

- Consider the task of creating an online blackjack game.
- Doing this using single variables and strings seems... impossible?
- What if we had a “card-deck” object, consisting of a bunch of “card” objects? Associated with these objects would be operations like “shuffle”, “deal”, and “trash”.

Classes allow us to create user-defined objects which are basically their own type. Types have certain operations defined on them, and so we can define the functionality we need.
What Use are Classes?

Classes allow us to:

- Define our own data types
- Define custom containers to hold data in an organized way
- Define functionality between our custom objects
- Abstractify our programs and make them widely applicable/reusable
Example: The class Exclaimer

```cpp
#include <iostream>
#include <string>
using namespace std;

class Exclaimer {
public:
    Exclaimer() {}
    void sayHello() {
        cout << 'Hello Los Angeles' << endl;
    }

    void shoutHello() {
        cout << myMessage << endl;
    }

    string myMessage = "GOOD MORNING LOS ANGELES";
};

int main() {
    Exclaimer Me = Exclaimer();
    Me.sayHello();
    Me.shoutHello();

    return 0;
}
```
Let Break It Down

The main code block:

- `Exclaimer Me = Exclaimer();` is initializing the object “Me”, which has type `Exclaimer`
- `Me.sayHello();` is calling the function `sayHello` that belongs to object “Me”
- `Me.shoutHello();` is calling the function `shoutHello` that belongs to object “Me”

To define the class `Exclaimer`, we have new notation:

- `class Exclaimer` is indicating that we will have a new class `Exclaimer`
- `Exclaimer() {}`; is something called an `constructor`, or what function we need to call to initialize an instance of this class
string myMessage = "GOOD MORNING LOS ANGELES"; is indicating that each Exclaimer object will contain the string “GOOD MORNING LA”

The two void functions defined belong to each instance of the class Exclaimer, and can be called using the ‘.’ operator public: indicates that all of class member functions and data members can be accessed from outside of the class. Private members can be written in too.

As a last analogy, we could create a human class, where each human would have several data members associated with each, such as a name, height, weight, hair color, eye color, and so on.
Classes

Fundamentals of Classes

Next Time
How to Create a Class

To properly create a class, we need to include methods for:

- Initializing instances of a class. This is done by defining one or several *constructors*.
- Defining which data members and member functions should exist.
- Declaring which members within a class will be *private* or *public*.
- Delete a class object after it has gone out of scope. This is done by defining a *destructor*, which has its own syntax.

To make things concrete, I will walk us through the process of creating a *Deck* class which can be used to draw a card from.
In our main code block, we want to be able to instantiate one or more instances of the Deck class. We want to do this by

```java
Deck firstdeck = Deck();
```

Calling Deck() here initializes the variable firstdeck. Within our definition of the Deck class, we need to have the line:

```java
Deck(){ \\startup routine here }
```

Perhaps we want our class to contain 52 cards by default. If our class as the data member `int Ncards;`, then upon initialization, we should set this to be 52.
Here is what we have so far:

```cpp
class Deck {
public:
    Deck() {
        Ncards = 52;
    }

private:
    int Ncards;
};
```

Since I defined `Ncards` to be private, how can the user access this if they needed to know how many cards are in the deck? The answer is, right now they can not! We need to create an access function which returns the number of cards in the deck.
Here is what we have so far:

class Deck {
public:
    Deck() {
        Ncards = 52;
    }

private:
    int Ncards;
};

Since I defined Ncards to be private, how can the user access this if they needed to know how many cards are in the deck?

- The answer is, right now they can not! We need to create an access function which returns the number of cards in the deck.
This will look something like this:

class Deck {
public:
    Deck() {
        Ncards = 52;
    }

    int get_Ncards() {
        return Ncards;
    }

private:
    int Ncards;
};
This will look something like this:

```cpp
class Deck {
public:
    Deck() {
        Ncards = 52;
    }

    int get_Ncards() {
        return Ncards;
    }

private:
    int Ncards;
};
```

The function `get_Ncards()` is referred to as a *public member function*, since its accessible from outside of the class, it belongs to the class, and it is a function. Now from the main code block, we could do:

```cpp
cout << firstdeck.get_Ncards() << endl;
```

and the number of cards in the deck will print to the console.
Public Member Functions

What are some essential publicly accessible member functions every class should have?

- **Constructor(s) - Mandatory**
- **Access functions -** For returning the value(s) of data members of a class
- **Assignment functions -** For reassigning variables which belong to a class
- **More advanced classes often have overloaded operators, which give an additional meaning to +, −, <<, etc.**
- **Destructor - Optional at first, mandatory later for good coding**
Inside of our class, in our public: section, we can have arbitrarily many member functions. Furthermore, we can have arbitrarily many variables stored in the class, which are referred to as public data members.

- Sometimes you’ll hear class method instead of member function
- I personally use method to mean “a function which acts internally within a class”, i.e. a private member function
- There are likely thousands of c++ experts who will tell me why that is not correct, but thats a little lingo
Perhaps we want to have a function which only has purpose within a class, or is already defined for another purpose in the main code. In that case, we typically define the function to be private within a class.

- The next example will show the usage of a private member function `determine_suit()`

- If this operation has usage in multiple places within the class, it makes sense to create a private function out of it
Example

class Deck {
public:
    Deck() {
        Ncards = 52;
        currentcard = 0;
    }

    int get_Ncards(){ return Ncards; }
    int get_currentcard(){ return currentcard; }
    void set_currentcard(int t){currentcard = t;}
    void printsuit(){
        string suit = determine_suit();
        cout << suit << endl;
    }

private:
    int Ncards;
    int currentcard;

    string determine_suit(){
        if (currentcard < 13){ return "spades"; }
        else if (currentcard < 26){ return "clubs"; }
        else if (currentcard < 39){ return "hearts"; }
        else{ return "diamonds"; }
    }
}
1 Classes

2 Fundamentals of Classes

3 Next Time
At last... Things to do between now and then:

- Classes: overloading, default constructor, copy constructor, member initialization