CS 31 Discussion 1J

ABDULLAH-AL-ZUBAER IMRAN

WEEK 7: CLASSES AND PROJECT5

Recap

- C-Strings
  - Library functions
  - Conversion to and from strings
- Structs
  - Definitions, declarations, memory and usage
- Enumerations

Class

A **class** is a construct used to group related fields (variables) and methods (functions).

```
class Cat {
  int m_age;
  void meow();
};
```

```
Cat meowth;
```

One can access these members by using a dot.

```
meowth.m_age      meowth.meow()
```

Class – member functions

Two ways to define the member functions

```
class Cat {
 int m_age;
 void meow() {
   cout << "Meow~" << endl;
 }
};
```

1. Inside the class definition

Scope resolution operator

```
void Cat::meow() {
  cout << "Meow~" << endl;
}
```

2. outside of the class definition

Class – access specifiers

The output of this code?

```
class Cat {
 int m_age;
 void meow() {
   cout << "Meow~" << endl;
 }
};

int main(){
 Cat meowth;
 meowth.m_age = 3;
 meowth.meow();
}
```

The class members have the private access specifier by default and cannot be accessed by an outside class or function.

```
class Cat {
public:
 int m_age;
 void meow() {
   cout << "Meow~" << endl;
 }
};
```

Class – access specifiers

```
meowth.m_age = -3;
```

Public members

Private members

```
int main(){
 Cat meowth;
 meowth.m_age = 3;
 meowth.meow();
}
```

```
class Cat {
public:
void meow() {
   cout << "Meow~" << endl;
 };
private:
 int m_age;
};
```

Class

Accessors

```
class Cat {
public:
  int age();
  void meow();
private:
  int m_age;
};
int Cat::age() {
  return m_age;
}
```

- Modifiers (mutators)

```
class Cat {
public:
  int age();
  void setAge(int newAge);
  void meow();
private:
  int m_age;
};
void Cat::setAge(int newAge) {
  m_age = newAge;
}
```

Class vs. Structs

Technically, the only difference between a class and a struct is the default access specifier.

By convention, we use structs to represent simple collections of data and classes to represent complex objects.

- This comes from C, which has only structs and no member functions.

Class - Constructors

A **constructor** is a member function that has the same name as the class, no return type, and automatically performs initialization when we declare an object:

```
class Cat {
public:
  Cat();
  int age();
  void meow();
private:
  int m_age;
};
Cat::Cat() {
  setAge(0);
  cout << "A cat is born" << endl;
}
```

When a constructor has no arguments, it is called the **default** constructor.
The compiler generates an empty one by default.

```
Cat kitty;
// uses default constructor -- Cat()
Cat *p1 = new Cat;
// uses Cat()
Cat *p2 = new Cat();
// uses Cat()
```

Class - Constructors

We can also create multiple constructors for a class by overloading it.

```
class Cat {
public:
  Cat(); // default constructor
  Cat(int initAge); // constructor with an initial age void meow();
  int age();
  void setAge(int newAge);
private:
  int m_age;
};
Cat::Cat(int initAge) {
  Cat(); // I can call the default constructor,
  setAge(initAge); // and then set the age to initAge
}
```

```
Cat kitty1(3);
```

Class - Constructors

Using initialization lists to initialize fields

```
class Cat {
public:
  Cat();
  int age();
  void setAge(int newAge);
private:
  int m_age;
  int m_weight;
  int m_gender;
};

Cat::Cat(): m_age(0), m_weight(10), m_gender(1)
{ /* some code */ }
```

Class - Constructors

Dynamic allocation

```
Cat *pKitty = new Cat();
Cat *pKitty2 = new Cat(10);
pKitty->meow()
(*pKitty).meow()
```

Class - Constructors

Can this compile? If so, what's the output?

```
#include<iostream>
using namespace std;
class Cat {
public:
  Cat(int initAge);
  int age();
  void setAge(int newAge);
private:
  int m_age;
};
Cat::Cat(int initAge) {
  setAge(initAge);
}
```

```
int main(){
  Cat meowth;
  Cat meowth1(2);
}
```

If we declare a constructor, the compiler will no longer generate a default constructor.

Class - Constructors

Can this compile? If so, what's the output?

```
#include<iostream>
using namespace std;
class Cat {
public:
  Cat(int initAge);
  int age();
  void setAge(int newAge);
private:
  int m_age;
};
Cat::Cat(int initAge) {
  setAge(initAge);
}
int Cat::age(){
  return m_age;
}
```

```
void Cat::setAge(int newAge){
  m_age = newAge;
}
class Person {
private:
  Cat pet;
};
int main(){
  Person Mary;
}
```

Class - Constructors

A fixed solution

```
#include<iostream>
using namespace std;
class Cat {
public:
  Cat(int initAge);
  int age();
  void setAge(int newAge);
private:
  int m_age;
};
Cat::Cat(int initAge) {
  setAge(initAge);
}
int Cat::age(){
  return m_age;
}
```

```
void Cat::setAge(int newAge){
  m_age = newAge;
}
class Person {
public:
  Person():pet(1){
    cout << "Person initialized" << endl;
  };
private:
  Cat pet;
};
int main(){
  Person Mary;
}
```

Class - Constructors

Order of construction

When we instantiate an object, we begin by initializing its member variables *then* by calling its constructor. (Destruction happens the other way round!)

The member variables are initialized by first consulting the initializer list. Otherwise, we use the default constructor for the member variable as a fallback.

For this reason, member variable without a default constructor must be initialized through the initializer list.

Class - Constructors

Can this compile? If so, what's the output?

```
class Cat {
public:
  Cat(string name) {
    cout << "I am a cat: " << name << endl;
    m_name = name;
  }
private:
  string m_name;
};
class Person {
public:
  Person(int age) {
    cout << "I am " << age << " years old. ";
    m_cat = Cat("Alfred");
    m_age = age;
  }
private:
  int m_age;
  Cat m_cat;
};
int main() {
  Person p(21);
}
```

```
Person(int age) : m_cat("Alfred") { ... }
```

const

```
const int a1 = 3;

const int* a2

int const * a3;

int * const a4

Int const * const a5
```

- For member functions in class
- int Cat::age() const
- {
- /* code */
- }
- Ban age() in class Cat from being anything which can attempt to alter any member variables in the object.

Separate class definitions and declarations

- As classes get longer and more complicated, having all the member function definitions inside the class can make the class harder to manage.
- Separate the “declaration” portion of the class from the “implementation” portion.
- Class declarations can be put in a header file to facilitate reuse in multiple files or projects. And the member function definitions in a cpp file.

Cat.h

```
#ifdef CAT_H
#define CAT_H
class Cat {
public:
  Cat(); // default constructor
  Cat(int initAge);
int age();
  void setAge(int newAge);
private:
  int m_age;
};
#endif
```

Cat.cpp

```
#include “Cat.h”

Cat::Cat(int initAge) {
  Cat(); // I can call the default constructor,
// and then set the age to initAge
setAge(initAge);

}
```

Enumeration

- A user-defined data type consisting of multiple constants. An enumeration is defined using the keyword “enum”

enum <name> {<const1>, <const2>, <const3>};

Example: *enum quarter {winter, spring, summer, fall};*

By default, winter is 0, spring is 1 and so on. You can change the default values.

quarter current;

current = winter;

cout << “Next Quarter: ” << current + 1;

What will be printed?

Project 5

Time due: 9:00 PM Wednesday, February 26

All of the public operations of MegaMillionsLottery will return a MegaMilionsTicket.

The relationship between the class MegaMillionsLottery and MegaMillionsTicket referred to as "creational pattern.“

MegaMillionsLottery creates MegaMillionsTicket

Three class methods to implement:

1. MegaMillionsTicket MegaMillionsLottery::quickpick();
2. MegaMillionsLottery::WinningPossibility MegaMillionsLottery::checkTicket(MegaMillionsTicket Ticket);
3. String MegaMillionsLottery::whatHappened(MegaMillionsTicket Ticket);

Turn in

1. The text files named **MegaMillionsTicket.h** and **MegaMillionsTicket.cpp** that implement the MegaMillionsTicket class.
2. The text files named **MegaMillionsLottery.h** and **MegaMillionsLottery .cpp** that implement the MegaMillionsLottery class.
3. The text file named **main.cpp** which will hold your main program.
4. (Optionally) The text files named **RadnomNumber.h** and **RandomNumber.cpp** that implement the RandomNumber class.
5. A file named **report.doc** or **report.docx** (in Microsoft Word format) or **report.txt** (an ordinary text file) that contains in addition **your name** and **your UCLA Id Number**: brief description of notable obstacles and a list of test data.

Don’t forget: Your source code should have helpful comments that explain any non-obvious code!!!

Thanks!

Questions?

Today's discussion slides can be found at

https://github.com/zubaerimran/W20-CS31-1J/blob/master/week7/winter20_cs31_w7.pdf

Some of the materials presented have been taken from earlier TA discussions