CS 31 Discussion

ABDULLAH-AL-ZUBAER IMRAN

WEEK 8:STRUCT AND CLASS

Discussion Objectives

Review and practice things covered during lectures

- Struct
- Class
- Class vs struct
- Class constructors
- const
- Project 6

Time for you to ask questions!

Struct

Structs are objects in C++ that represent "data structures", or variables, functions, etc. that are organized under a categorizing identifier.

Data Members are variable components of a given struct; they can be of any variable type.

```
struct <structName> {
  <member1_type> <member1_name>;
  <member2_type> <member2_name>;
  // ...etc.
}; // Remember the semicolon!
```

```
struct Student {
  string name;
  int id;
  string email;
  char grade;
};
```

Struct

```
struct Student {
  string name;
  int id;
  string email;
  char grade;
};
```

```
const int NUM_STUDENTS = 32;
Student st
Student students[NUM_STUDENTS];
```

The element/member selection operation (.)

```
st.name = "Joe Bruin";
// st's name is set to "Joe Bruin"
students[10].id = 123456789;
// the 10-th Student in students array is assigned an ID
cout << st.grade << endl;
// print the grade of st
cout << students[0].email << endl;
// print the 0-th Stduent's email address
```

Member Functions (Method)

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

struct Student {
  string name;
  int id;
  string email;
  char grade;

  Student() {
    name = "Jane Doe";
    id = 0;
    email = "aa@a.com";
    grade = 'A';
  }
  void printName() {
    cout << name << endl;
  }
};
```

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

struct Student {
  string name;
  int id;
  string email;
  char grade;

  Student() {
    name = "Jane Doe";
    id = 0;
    email = "aa@a.com";
    grade = 'A';
  }
};

void Student::printName() {
    cout << name << endl;
}
```

```cpp
int main() {
  Student p;
  p.printName();
}
```

```
Jane Doe
```

What's the output?

```
#include <iostream>
#include <cstring>
#include <cctype>
#include <string>
using namespace std;
struct trickz {
  int i[5];
  trickz () {
    for (int j = 0; j < 5; j++) {
      // Assume ASCII encoding
      i[j] = j;
    }
  };
  void trickzInc (trickz*& t, int count) {
    for (int j = 0; j < count; j++) {
      t->i[j]++;
    }
  }
};

int main () {
  trickz t;
  trickz* ptr = &t;
  t.trickzInc(ptr, 5);
  for (int j = 0; j < 5; j++) {
    cout << t.i[j] << endl;
  }
  // True or false?
  cout << (ptr == &t) << endl;
}
```

```
1
2
3
4
5
1
```

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

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

```cpp
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 beging anything which can attempt to alter any member variables in the object.

Project6

- More like a homework
- Designed to help you master pointers (five problems)
- **Time due: 11:00 PM Tuesday, May 28**

The hypotenuse function is correct, but the main function has a problem. Explain why it may not work, and show a way to fix it. Your fix must be to the main function only; you must not change the hypotenuse function in any way.

```
#include <iostream>

#include <cmath>

using namespace std;

void hypotenuse(double leg1, double leg2, double* resultPtr)

  {  *resultPtr = sqrt(leg1*leg1 + leg2*leg2);    }

 int main()

   {  double* p;

       hypotenuse(1.5, 2.0, p);

       cout << "The hypotenuse is " << *p << endl;

   }
```

5. Write a function named deleteG that accepts one character pointer as a parameter and returns no value. The parameter is a C string. This function must remove all of the upper and lower case 'g' letters from the string. The resulting string must be a valid C string.

Your function must declare no more than one local variable in addition to the parameter; that additional variable must be of a pointer type. Your function must not use any square brackets. Do not use any of the <cstring> functions such as strlen, strcpy, etc.

```
int main()

{

    char msg[100] = "I recall the glass gate next to Gus in Lagos, near the gold bridge.";

    deleteG(msg);

    cout << msg;  // prints   I recall the lass ate next to us in Laos, near the old bride.

}
```

Thanks!

Questions?

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