Computer Programing (CP)

Lecture # 6 & 7

Struct vs Class

Struct

- A *struct* is a group of data elements grouped together under one name.
- These data elements, known as *members*
- Data elements have different types and different lengths.
- Struct can be declared in C++ using the following

```
struct type_name {
member_type1 member_name1;
member_type2 member_name2;
member_type3 member_name3;
.
.
} object_names;
```

Class

- Classes are an the user defined datatype
- A class definition begins with the keyword ***class***.
- A class consists of a set of variables called ***member variables*** and a set of functions called ***member functions***.
- The programmer can limit access to parts of the class using ***access modifiers.***

```
class class_name {
  access_specifier_1:
    member1;
  access_specifier_2:
    member2;
  ...
};
class_name object_names;
```

A Motivating Example

You declare a simple structure:

```
struct Robot {
  float locX;
  float locY;
  float facing;
};

Robot r1;
```

What if you never want locX or locY to be negative?

Someone unaware of your restriction could set:

r1.locX = -5;

and you have a problem!

General Class Definition in C++

```
class ClassName {
  access_modifier1:
    field11_definition;
    field12_definition;
    …
  access_modifer2:
    field21_definition;
    field22_definition;
  …
};
```

ClassName is a new type

Possible access modifiers:

- public
- private
- protected

Fields following a modifier are of that access until the next modifier is indicated

Fields: Member variables

Member functions

A Simple Class

```
class Robot {
  public:
    float getX() { return locX; }
    float getY() { return locY; }
    float getFacing() { return facing; }
    void setFacing(float f) { facing = f; }
    void setLocation(float x, float y);
  private:
    float locX;
    float locY;
    float facing;
};
```

`struct` versus `class`

- In C++ struct and class can be used interchangeably to create a class with one exception
- What if we forget to put an access modifier before the first field?

```
struct Robot {        OR        class Robot {
  float locX;                       float locX;
```

In a class, until an access modifer is supplied, the fields are assumed to be `private`

In a struct, the fields are assumed to be `public`

Access Modifiers

`private` Access Modifier

- Fields marked as `private` can only be accessed by functions that are part of that class
- In the Robot class, `locX, locY` and `facing` are `private` member variables, these fields can only be accessed by functions that are in class
- class Robot (getX, getY, getFacing, setFacing, setLocation)
- Example:

```
void main() {
  Robot r1;
  r1.locX = -5; // Error
```

`public` Access Modifier

- Fields marked as `public` can be accessed by anyone
- In the Robot `class`, the methods `getX, getY` etc. are public
- these functions can be called by anyone

- Example:

```
void main() {
  Robot r1;
  r1.setLocation(-5,-5); // Legal to call
```

`protected` Access Modifier

protected: can only be called by or accessed by functions that are members of the derived class

- C++ allows users to create classes based on other classes:
  - a FordCar class based on a general Car class
  - idea: the general Car class has fields (variables and functions that describe all cars)
  - the FordCar class then uses the fields from the general Car class and adds fields specific to FordCars

Class Methods

Types of Class Methods

Generally we group class methods into three broad categories:

accessors - allow us to access the fields of a class instance (examples: getX, getY, getFacing), accessors do not change the fields of a class instance

mutators - allow us to change the fields of a class instance (examples: setLocation, setFacing), mutators do change the fields of a class instance

manager functions - specific functions (constructors, destructors) that deal with initializing and destroying class instances (more in a bit)

Class Methods

- Functions associated with a class are declared in one of two ways:

  ReturnType FuncName(*params*) { *code* }

  - function is both declared and defined (code provided)

  ReturnType FuncName(*params*);

  - function is merely declared, we must still define the body of the function separately

- To call a method we use the . form:

  classinstance.FuncName(args);

  - FuncName is a public member function of the class

Defined Methods

```
class Robot {
  public:
    float getX() { return locX; }
};
Robot r1;
```

Implicitly Inline

- The function getX is defined as part of class Robot
- To call this method:
  - `cout << r1.getX() << endl; // prints r1's locX`
- Why define the method this way?
  - Implicitly *inline*

Inline Functions

- In an inline function, the C++ compiler does not make a function call, instead the code of the function is used in place of the function call
- Why?
  - Inline functions *don't have the overhead* of other functions
  - Things like accessing/changing fields of a class instance should be *fast*
- Not all requests to make a function inline are honored
  - generally C++ examines the complexity of the function and the use of parameters in the function
  - should only request inline definition for short functions
- Can also explicitly request to make class methods and other functions inline (add inline keyword before return type in function declaration and definition)

Defined Methods (Outside the Class)

- For methods that are declared but not defined in the class we need to provide a separate definition
- To define the method, you define it as any other function, except that the name of the function is

- *ClassName*::*FuncName*

  :: is the scope resolution operator, it allows us to refer to parts of a class or structure

A Simple Class

```
class Robot {
  public:
    void setLocation(float x, float y);
  private:
    float locX;
    float locY;
    float facing;
};

void Robot::setLocation(float x, float y) {
  if ((x < 0.0) || (y < 0.0))
    cout << "Illegal location!!" << endl;
  else {
    locX = x;
    locY = y;
  }
}
```

Explicitly Requesting Inline

```
class Robot {
  public:
    inline void setLocation(float x, float y);
};

inline void Robot::setLocation(float x, float y) {
  if ((x < 0.0) || (y < 0.0))
    cout << “Illegal location!!” << endl;
  else {
    locX = x;
    locY = y;
  }
}
```

Referring to Class Fields

```
void Robot::setLocation(float x, float y) {
  if ((x < 0.0) || (y < 0.0))
    cout << “Illegal location!!” << endl;
  else {
    locX = x;
    locY = y;
  }
}
```

- What are locX and locY in setLocation??
  - Since a class function is always called on a class instance, locX, locY are those fields of that instance
  - Example:

```
Robot r1;
Robot r2;
r1.setLocation(5,5);  // locX is from r1
r2.setLocation(3,3);  // locY is from r2
```