Intro to Classes in C++

Classes

Programmer-defined types

Made up of members

- Variables
- Functions – called methods when part of a class
- Constructors: Initialize the class
- Destructors: Clean up as the class is being removed/ deleted

Concept is the same as C#, Java, etc.

Where C-structs have only variables, C++ classes are complete objects with methods plus data

Still use .h files for ‘defining’; Use .cpp or .cc files for implementation

Syntax, semantics and concepts

While ‘C’ syntax can be used, C++ class and object syntax can be quite different

Similarities:

- Use of #include files

- Control Structure: if/ then/ else/ while control structures

- Manual Memory management (but now for objects as well as data)

- Can actually write typical C code (and not use C++ enhancements)

Differences:

-File Structure: (.h) for class structure, .cpp files implement the classes (sometimes .cc files)

-Memory Management : new/delete (vs. malloc and free)

Object oriented programming:

- Abstract classes as ‘templates’
- public/ private/ protected variables and methods
- Constructors and Destructors
- Override and virtual methods
- Inheritance and polymorphism

- I/O: cout/cin vs printf/scanf

- Namespaces prevent naming conflict

- Iterators for traversing containers

Compiled with g++ in Linux

Class Declaration – Point.h

A class declaration tells the compiler about a class's name and structure without defining its full behavior(methods are usually defined in a separate .cpp file). Generally declared in a header file (.h)

Syntax

- Starts with the class keyword.
- Class names are capitalized by convention.
- Semicolon at the end is mandatory.

```
// Point.h

class Point

{

public:

          int x;

          int y;

          void move(int dx, int dy);

}; // Notice the semicolon!
```

Class Access Specifiers

C++ has three main access specifiers:

- public
  - Members declared public are accessible from anywhere in the program.
  - Used for functions or variables that should be available to other parts of the code.
- private
  - Members declared private are only accessible within the class itself.
  - Used to hide internal data and protect it from accidental changes.
  - All members are private by default.
- protected
  - Members declared protected are accessible within the class and by derived(child) classes, but not outside.
  - Useful in inheritance.

Usage is different from C# / Java

- Define sections with specified access

Access Specifier Example

```
class Point
{
public:
  void setCoordinates(int a,int b){
          x=a;
          y=b;
}
private:
          int x;
          int y;
};
```

Usage :

```
Point p;
// p.x =5   // ERROR
p.setCoordinates(3,4);
```

Constructors

- A constructor is a special member function if a class that is automatically called when an object is created.
- Its main job is to initialize objects.
- No return type. Not even void.
- Usually has the same name as the class.
- Default constructor- no parameters, used to initialize objects with default values.
  - If no constructor is defined, the compiler provides a default constructor automatically.
- Parameterized constructor – takes arguments to initialize object with specific values . Can be overloaded.

Constructor Declaration

```
class Point
{
public:
      Point();      // Default Constructor
      Point(int x, int y); // Constructor

private:
      int x;         // Member variable
      int y;               // Member variable
};
```

Methods

- Function that belongs to a class. It can operate on objects of class, access private/protected data members, and perform actions related to that class.
- Defined inside or outside of a class.
- Methods have access to other class members
  - Other methods
  - Variables

Methods can use “this” keyword

- A *pointer* to this object
- More on pointers & objects soon

Method Example

```
class Point
{
public:
        Point();     // Default Constructor
        Point(int x, int y);        // Constructor
        int GetX();                 // Method
        int GetY();                 // Method

private:
        int x;                      // Member variable
        int y;                      // Member variable
};
```

Class Implementation – Point.cpp

Generally defined in a .cpp file

- #include associated header file

Should define code of all members

- Methods
- Constructors
- Destructors

No “class” keyword in .cpp file!

- Must use scope operator

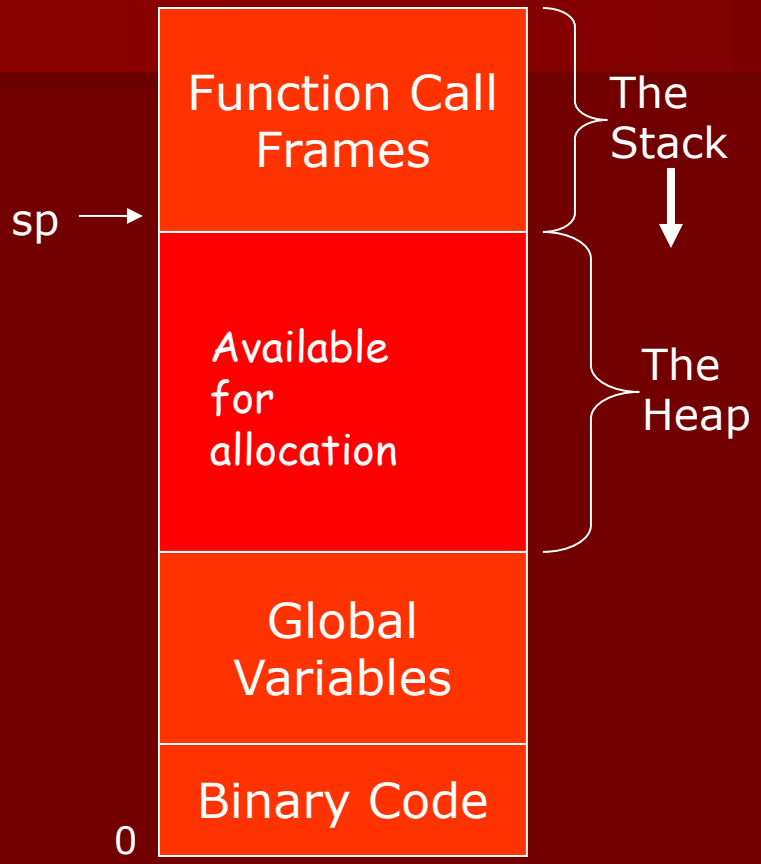
Point.cpp - Example Part 1

```
#include “Point.h"
// Constructors – Notice class name before ::
Point::Point()
{
        x = 0;
        y = 0;
}

Point::Point(int x, int y)
{
        // More on the “->” soon
        this->x = x;
        this->y = y;
}
```

Point.cpp - Example Part 2

```
// The rest of the file


// Methods – Again, class name before ::
int Point::GetX() { return x; }
int Point::GetY() { return y; }
```

Remember this? - Memory Organization



Similar in C++

The stack now holds local objects as well as data

Dynamic alloation for objects (and data) is from the heap

- The call stack grows from the top of memory down.
- Code is at the bottom of memory.
- Global data follows the code.
- What's left – the "heap" – is available for allocation.

When you declare a pointer

*int *pInt;*

It is pointing to NOTHING

You need to create a valid 'chunk' of memory, and assign the pointer to that valid memory

RAII

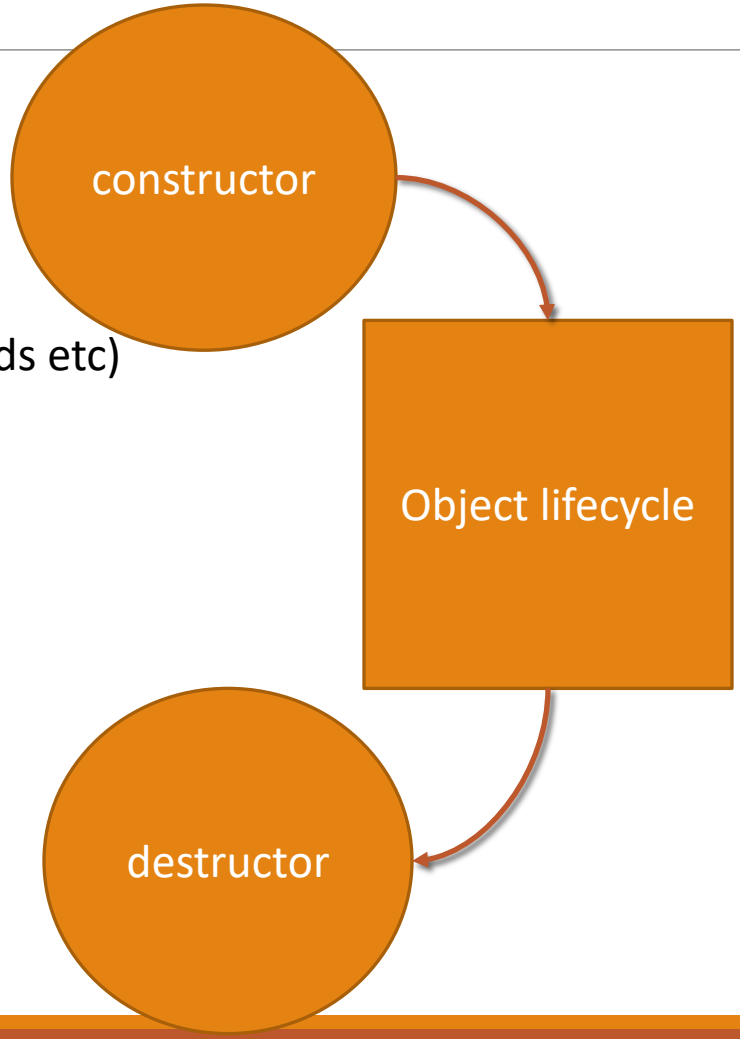
Resource **A**cquisition **I**s **I**nitialization- Tie the lifetime of a resource(memory, file etc.) to the lifetime of an object

- When the object is created(constructor), it acquires the resource.
  - Constructor is automatically called for initialization.
- When the object is destroyed(destructor), it releases the resources automatically.
- Prevents resource leaks(memory, file handles, sockets). Resources are released even if an exception occurs.
- Makes code cleaner and safer.

This is one of the keyways C++ manages resources safely without garbage collection.

Lifecycle

```
//create the object
MyClass *pObject = new MyClass();
….
…
//Do stuff with the object (call methods etc)
…
…
delete pObject; //destroy the object
```



Instantiating Objects of a Class

- Instantiation means creating an actual object of a class.
- A class is just a blueprint; an object is real entity in memory.
- When an object is created:
  - The constructor is automatically called.
  - Memory is allocated for the object.

Ways to Instantiate Objects:

- Stack Allocation- automatic memory ; objects created inside functions are on the stack. Destructor is automatically called when the object goes out of scope.

- Heap Allocation-dynamic memory using **new** ; for objects created on the heap, you must delete them manually, otherwise the destructor won't run and memory will leak;

Many syntax options for creating objects

Example

```
#include <iostream>

using namespace std;

Class Dog{

public :
 string name;
 int age;

 //constructor
  Dog(string n, int a){
          name = n; age= a;
          cout <<name << "is born !" <<endl;
 }
 // Method
 void bark(){ cout << name << "says Woof!"<<endl; }

 //Destructor: called when object is destroyed
 ~Dog() { cout << name << "is destroyed! "<< endl;}
 };
```

STACK ALLOCATION

```
int main()

{

Dog dog1("Joy",3);// constructor called

dog1.bark();

//Destructor will be automatically called
at end of main()

return 0;

}
```

```
Joy is born!

Joy says Woof!

Joy is destroyed!
```

HEAP ALLOCATION

```
int main()

{

Dog* dogPtr = new Dog("Jackie",3);//
constructor called

dogPtr->bark();

Delete dogPtr; //Destructor called
manually

return 0;

}
```

```
Joy is born!

Joy says Woof!

Joy is destroyed!
```

Example

```
class widget
{
private:
    int* data;// pointer to an int
public:
// allocates an array of integers of the specified size dynamically using new[]
    widget(const int size) { data = new int[size]; } // acquire
// deletes dynamically allocated array using delete[]
    ~widget() { delete[] data; } // release
    void do_something() {}
};

void functionUsingWidget() {
    widget w(1000000);
   //lifetime automatically tied to enclosing scope constructs w, including the w.data
member
   w.do_something();

} // automatic destruction and deallocation for w and w.data
```

Objects on the Stack

```
int main()
{
   // Call constructors, but not saved in variables
   Point();                                  // Default constructor
   Point(5,5);                         // Parameterized

   // Variables
   Point p1;                           // Default
   Point p2();                         // NEITHER!
   Point p3(5, 5);                     // Parameterized
   Point p4 = Point();                 // Default
   Point p5 = Point(5, 5);             // Parameterized
}
```

What's Wrong With This?

What's wrong with this line:

```
Point p2();
```

Looks like it should call default constructor

Technically a function declaration

- A function named p2, which returns a Point
- Yes, even though it's in the main!

Objects on the Heap

```
// Remember: new returns a pointer!
int main()
{
	// Call default constructor
	Point* p6 = new Point;
	Point* p7 = new Point();

	// Call parameterized constructor
	Point* p8 = new Point(5,5);
}
```

Destructors – time to clean up

```
// Called when the object is destroyed (stack or delete)!
class Point
{
public:
        Point();     // Default Constructor
        ~Point(); // Destructor
private:
        MyObject* pObject;
};
```

```
Point::~Point()
{
 if (pObject)
 {
    delete pObject;
    pObject = nullptr;
  }
};
```

Calling Methods – Local Variables

Can call methods once you have an object

Local variable method syntax is simple:

```
// Create a point
Point p = Point(5, 5);

// Get the x value
int x = p.GetX( );
```

Calling Methods - Pointers

Not as straight-forward

Can’t call a method on a memory address

Must dereference first, then call method

- Or use the arrow operator: ->

Calling Methods - Pointers

```
// Create a new Point, get a pointer to it
Point* p = new Point(5, 5);

// Dereference and call
int x = (*p).GetX( );

// Or use “->” syntax
// Essentially “dereference and call”
int y = p->GetY( );
```

Creating Classes/ Building

Basically, you need two files

- .h file: Sets up basic class declaration & definition
- .cpp (or .cc) implementation/ code

Compiling
- Same as “C” i.e. gcc or g++ and Makefile