Structures and Classes
Lecture 4

University of Anbar

College of Computer Science and Information Technology

Department of Computer Science

Object Oriented Programming

Second Class

Dr. Ruqayah R. Al-Dahhan

Outlines:

- Structures
- Classes

Introduction:

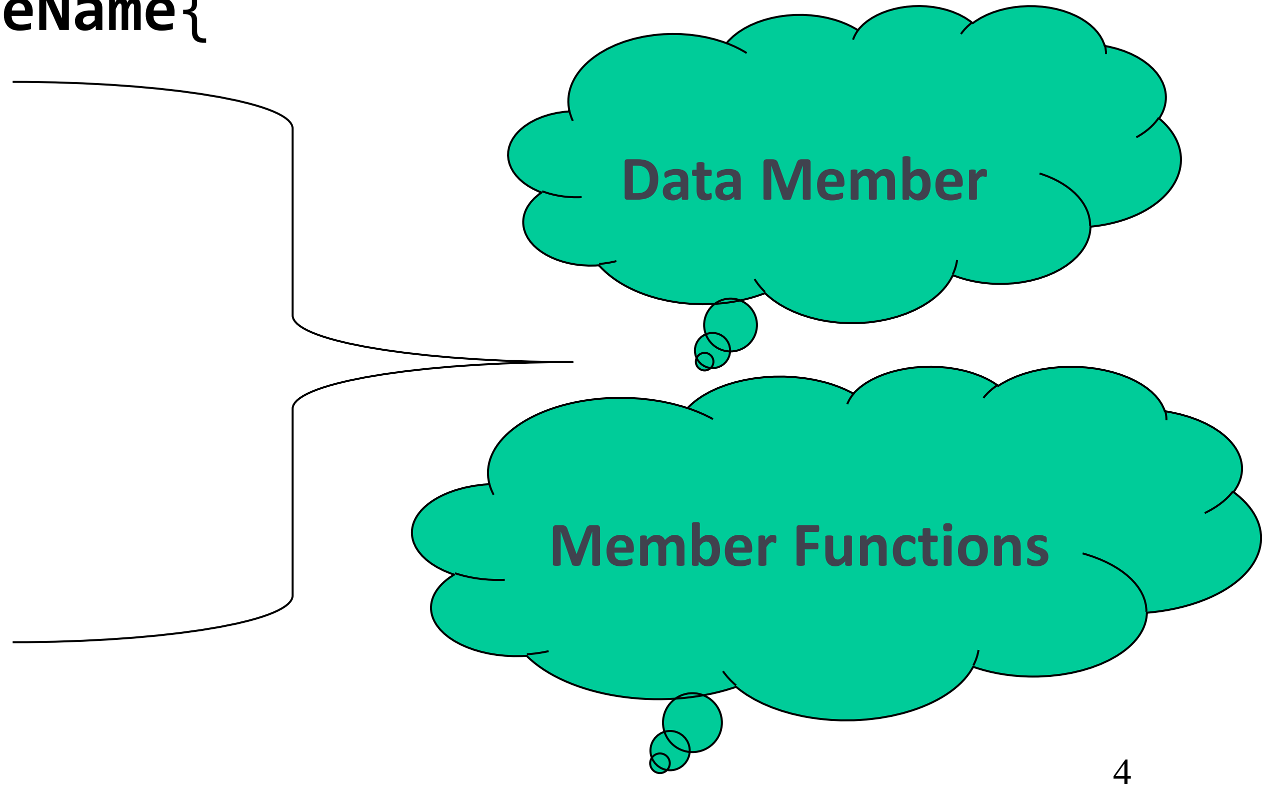
- **Arrays in C++:** are used to store set of data of similar data types at contiguous memory locations.
- **Structures in C++** are **<u>user-defined data types</u>** which are used to store group of items of non-similar data types.
- A **structure** creates a data type that can be used to group items of possibly different types into a single type.

How to create a structure?

The ‘**struct’** keyword is used to create a structure. The general syntax to create a structure is as shown below:

```
struct structureName{
member1;
member2;
member3;
. . .


memberN; };
```



Data Member

Member Functions

Structures

Structures in C++ can contain two types of members:

- **Data Member**: These members are normal C++ variables. We can create a structure with variables of different data types in C++.

- **Member Functions**: These members are normal C++ functions. Along with variables, we can also include functions inside a structure declaration.

```
#include <iostream>
using namespace std;
struct Point{
  int x;
  int y;
};
void outputAPoint( Point  );  // function prototype
main(){
  Point  one, two;
  one.x = 1;
  one.y = 2;
  two.x = 3;
  two.y = 4;
  outputAPoint(one);
  outputAPoint(two);

}

void outputAPoint( Point p ){
  cout << "Point :" <<  p.x << "," << p.y << endl;
}
```

- C++ Struct syntax is simpler

- Example Output:

```
Point : 1,2
Point : 3,4
```

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

struct Point{
  int x;
  int y;
  void outputAPoint(  ){
  cout << "Point :" <<  x << "," << y << endl;
}

};

 main(){
  Point  one, two;
  one.x = 1;
  one.y = 2;
  two.x = 3;
  two.y = 4;
  one.outputAPoint();
  two.outputAPoint();

}
```

- Example Output:

```
Point : 1,2
Point : 3,4
```

Classes

A **class** is a ***user-defined type*** that contains ***data*** as well as the set of ***functions*** that manipulate that data.

Classes

```
struct Point {
   int x,y;
};

...

Point w;
```

C++ implements **classes** by extending the idea of structures.

The name of a `struct` is automatically a new type.

We can use the keyword **class** instead of **struct -** they are almost the same

Classes

In C++ a structure not only groups **data**, it also groups **operations** that can be performed on data.

```
struct Point {
   int x,y;
   void print(){
     cout << “(” << x  << “,” << y <<“)”;
   }
};
```

We describe `print` as being a member function of the class `Point`

Classes

`w.print()` invokes the `print` function of the `Point` structure (or class)

```
int main(){
    Point w;
    w.x = 2;
    w.y = 5;
    w.print();
}
```

Classes

C++ limits the **visibility** of data and functions by allowing **public** and **private** parts to a structure.

By default all elements of a struct are **public**.
Programs that use variables of this type are allowed to access all data and all functions of the structure.

```
    w.y = 5;  // accessible to the calling code
    w.print();
```

Sometimes we do not want all the innards of a class to be accessible by calling code - we may want to hide part or all of it.

Classes

Declarations within the **private** section of a structure are only visible to the structure itself.

```
struct Point {
 public:
   void print(void) {
      cout << “(“ << x << “,” << y << “)”;
   }
 private:
   int x,y;
};
```

We can no longer access the data items x and y directly from calling code!

But we are allowed to print them using `print()`!

```
struct Point {
 public:
    void print() {
       cout << “(“ << x << “,” << y << “)”;
    }
    void init(int u, int v) {
       x = u;
       y = v;
    }
 private:
    int x,y;
};

int main() {
   Point w;  // declares w to be of type Point
   w.init(2,5);  // allowed, since init is public
   w.print();    // also allowed
   //w.x=90; compile ERROR sincex is private
}
```

Classes

Now the structure is very secure! - no one can alter the data of the structure without using the functions that are supplied by the structure itself:

```
int main() {
   Point w;

   w.init(2,5);
   w.print();
   //w.x=90; ERROR x is private in Point
}
```

Data Hiding or Encapsulation

- Why would you want to hide data from the rest of your program?
- Perhaps to protect it from accidental misuse elsewhere in the program
- Example a **Date** class might group *day*, *month*, and *year*. These need to be kept consistent - we do not want part of the user program accidentally setting *day* to something incorrect such as **-1** or even something inconsistent such as **30** when the month is **February**.
- Encapsulation lets us restrict the ways our data variables are manipulated elsewhere in the program.

Classes

Stopping un-authorised access to data is ‘good practice’ and is one of the benefits of using C++.

The keywords `public` and `private` can be used many times within a structure.

It is usual to put all `public` members first and `private` members last.

Always use `private` and `public` - do not leave them as defaults.

Classes

C++ introduces a new keyword: `class`

A `class` is exactly the same as a `struct` except that all members are private unless specified otherwise.

Most people use `class` rather than `struct`.

Classes

```
class Point {
  int x,y;    //private
  void print();//private
public:
  void init(int, int);
private:
  int distance;
};
```

```
struct Point {
  int x,y;     //public
  void print();//public
public:
  void init(int, int);
private:
  int distance;
};
```

Summary

A class is a way of implementing a data type and associated functions and operators that operate on that data.

Classes have **public** and **private** members that provide data hiding.