Lecture 17

M M Imran

C++ Structures (Struct)

- Structures (also called structs) are a way to group several related variables into one place.

- Each variable in the structure is known as a member of the structure.

- Unlike an array, a structure can contain many different data types (int, string, bool, etc.).

Why Use Structure?

- You want to store some information about a person: his/her name, citizenship number and salary. You can easily create different variables **name, citNo, salary** to store these information separately.
- However, in the future, you would want to store information about multiple persons. Now, you'd need to create different variables for each information per person: **name1, citNo1, salary1, name2, citNo2, salary2**
- You can easily visualize how big and messy the code would look. Also, since no relation between the variables (information) would exist, it's going to be a daunting task.
- A better approach will be to have a collection of all related information under a single name Person, and use it for every person. Now, the code looks much cleaner, readable and efficient as well.

Declare a Structure

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

```
struct type_name {
member_type1 member_name1;
member_type2 member_name2;
member_type3 member_name3;
.
.
} object_names;
// defining object_names is optional
```

Example

```
struct product {
  int weight;
  double price;
} ;

product apple;
product banana, melon;
```

This declares a structure type, called product, and defines it having two members: weight and price, each of a different fundamental type.

This declaration creates a new type (product), which is then used to declare three objects (variables) of this type: apple, banana, and melon.

Another way

```
struct product {
  int weight;
  double price;
} apple, banana, melon;
```

Right at the end of the struct definition, and before the ending semicolon (;), the optional field **object_names** can be used to directly declare objects of the structure type.

Accessing Members of a Structure

- The members of structure variable is accessed using a dot (.) operator.
- Once the three objects of a determined structure type are declared (apple, banana, and melon) its members can be accessed directly.

apple.weight
apple.price
banana.weight
banana.price
melon.weight
melon.price

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

struct Person
{
    char name[50];
    int age;
    float salary;
};

int main()
{
    Person p1;

    cout << "Enter Full name: ";
    cin.get(p1.name, 50);
    cout << "Enter age: ";
    cin >> p1.age;
    cout << "Enter salary: ";
    cin >> p1.salary;

    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p1.name << endl;
    cout <<"Age: " << p1.age << endl;
    cout << "Salary: " << p1.salary;

    return 0;
}
```

```
Enter Full name: Magdalena Dankova
Enter age: 27
Enter salary: 1024.4

Displaying Information.
Name: Magdalena Dankova
Age: 27
Salary: 1024.4
```

Structure and Function

- Structure variables can be passed to a function and returned in a similar way as normal arguments.

Output

```
Enter Full name: Bill Jobs
Enter age: 55
Enter salary: 34233.4

Displaying Information.
Name: Bill Jobs
Age: 55
Salary: 34233.4
```

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

struct Person {
    char name[50];
    int age;
    float salary;
};

void displayData(Person p) {
    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p.name << endl;
    cout <<"Age: " << p.age << endl;
    cout << "Salary: " << p.salary;
}

int main() {
    Person p;

    cout << "Enter Full name: ";
    cin.get(p.name, 50);
    cout << "Enter age: ";
    cin >> p.age;
    cout << "Enter salary: ";
    cin >> p.salary;

    displayData(p);

    return 0;
}
```

Structure Function

```
struct Person {
    char name[50];
    int age;
    float salary;
};

Person getData(Person p) {
    cout << "Enter Full name: ";
    cin.get(p.name, 50);
    cout << "Enter age: ";
    cin >> p.age;
    cout << "Enter salary: ";
    cin >> p.salary;

    return p;
}
```

```
void displayData(Person p) {
    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p.name << endl;
    cout <<"Age: " << p.age << endl;
    cout << "Salary: " << p.salary;
}

int main() {
    Person p, temp;

    temp = getData(p);
    displayData(temp);

    return 0;
}
```

Structure Member Function

```
struct Person {
    char name[50];
    int age;
    float salary;

    void displayData() {
        cout << "\nDisplaying Information." << endl;
        cout << "Name: " << name << endl;
        cout <<"Age: " << age << endl;
        cout << "Salary: " << salary;
    }
};
```

```
Person getData(Person p) {
    cout << "Enter Full name: ";
    cin.get(p.name, 50);
    cout << "Enter age: ";
    cin >> p.age;
    cout << "Enter salary: ";
    cin >> p.salary;

    return p;
}

int main() {
    Person p;
    p = getData(p);
    p.displayData();

    return 0;
}
```

Structure Constructor

```
struct Person {
    int age;
    float salary;

    Person(int a, float s) {
        age = a;
        salary = s;
    }

    void displayData() {
        cout << "Displaying Information." << endl;
        cout <<"Age: " << age << endl;
        cout << "Salary: " << salary;
    }
};

int main() {
    Person p = Person(25, 4000);
    p.displayData();

    return 0;
}
```

```
Displaying Information.
Age: 25
Salary: 4000
```

Structure Pointer

Like primitive types, we can have pointer to a structure. If we have a pointer to structure, members are accessed using arrow (->) operator instead of the dot (.) operator.

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

int main()
{
    struct Point p1 = { 1, 2 };
    // p2 is a pointer to structure p1
    struct Point* p2 = &p1;
    // Accessing structure members using
    // structure pointer
    cout << p2->x << " " << p2->y;
    return 0;
}
```

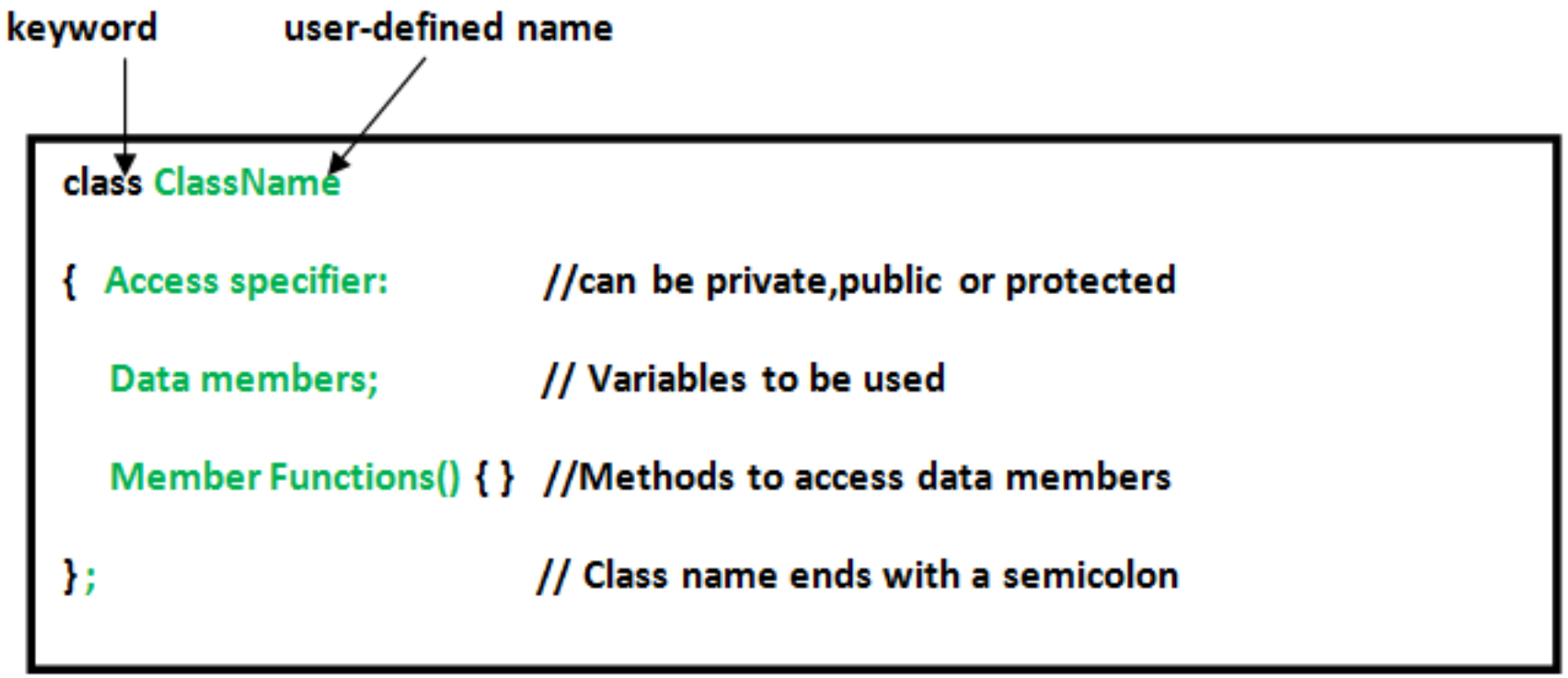
C++ Classes and Objects

- A class in C++ is the building block that leads to Object-Oriented programming.
- It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.
- A C++ class is like a blueprint for an object.

Example

Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

Defining Class and Declaring Objects



Here, we defined a class named Room.

The variables length, breadth, and height declared inside the class are known as data members.

And, the functions calculateArea() and calculateVolume() are known as member functions of a class.

```
class Room {
    public:
        double length;
        double breadth;
        double height;

        double calculateArea(){
            return length * breadth;
        }

        double calculateVolume(){
            return length * breadth * height;
        }

};
```

C++ Objects

- When a class is defined, only the specification for the object is defined; no memory or storage is allocated.

- To use the data and access functions defined in the class, we need to create objects.

- Syntax

    className objectVariableName;

```
class Room {
    public:
        double length;
        double breadth;
        double height;

        double calculateArea(){
            return length * breadth;
        }

        double calculateVolume(){
            return length * breadth * height;
        }

};
```

```
// sample function
void sampleFunction() {
    // create objects
    Room room1, room2;
}

int main(){
    // create objects
    Room room3, room4;
}
```

Access Data Members and Member Functions

We can access the data members and member functions of a class by using a . (dot) operator. For example,

room2.calculateArea();

This will call the calculateArea() function inside the Room class for object room2.

Similarly, the data members can be accessed as:

room1.length = 5.5;

In this case, it initializes the length variable of room1 to 5.5.

```
class Room {

   public:
    double length;
    double breadth;
    double height;

    double calculateArea() {
        return length * breadth;
    }

    double calculateVolume() {
        return length * breadth * height;
    }
};
```

```
int main() {

    // create object of Room class
    Room room1;

    // assign values to data members
    room1.length = 42.5;
    room1.breadth = 30.8;
    room1.height = 19.2;

    // calculate and display the area and volume of the room
    cout << "Area of Room =  " << room1.calculateArea() << endl;
    cout << "Volume of Room =  " << room1.calculateVolume() << endl;

    return 0;
}
```

Output

```
Area of Room =  1309
Volume of Room =  25132.8
```

C++ Constructors

- A constructor is a special type of member function that is called automatically when an object is created.
- In C++, a constructor has the same name as that of the class and it does not have a return type. For example,

```
class  Wall {
  public:
    // create a constructor
    Wall() {
      // code
    }
};
```

Passing Parameters to Constructor

```
class Wall {
  private:
    double length;
    double height;

  public:
    Wall(double len, double hgt) {
      length = len;
      height = hgt;
    }

    double calculateArea() {
      return length * height;
    }
};
```

```
int main() {
  // create object and initialize data members
  Wall wall1(10.5, 8.6);
  Wall wall2(8.5, 6.3);

  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;
  cout << "Area of Wall 2: " << wall2.calculateArea();

  return 0;
}
```

C++ Access Modifiers

- One of the main features of object-oriented programming languages such as C++ is data hiding.
- Data hiding refers to restricting access to data members of a class. This is to prevent other functions and classes from tampering with the class data.
- However, it is also important to make some member functions and member data accessible so that the hidden data can be manipulated indirectly.
- The access modifiers of C++ allows us to determine which class members are accessible to other classes and functions, and which are not.

Here, the variables patientNumber and diagnosis of the Patient class are hidden using the private keyword, while the member functions are made accessible using the public keyword.

```
class Patient {

    private:
        int patientNumber;
        string diagnosis;

    public:

      void billing() {
          // code
      }

      void makeAppointment() {
          // code
      }
};
```

Types of Access Modifiers

In C++, there are 3 access modifiers:

1. public
2. private
3. protected

public Access Modifier

- The public keyword is used to create public members (data and functions).
- The public members are accessible from any part of the program.

```
// define a class
class Sample {

    // public elements
   public:
    int age;

    void displayAge() {
        cout << "Age = " << age << endl;
    }
};
```

```
int main() {

    // declare a class object
    Sample obj1;

    cout << "Enter your age: ";

    // store input in age of the obj1 object
    cin >> obj1.age;

    // call class function
    obj1.displayAge();

    return 0;
}
```

private Access Modifier

- The private keyword is used to create private members (data and functions).
- The private members can only be accessed from within the class.
- However, friend classes and friend functions can access private members.

```cpp
// define a class
class Sample {

    // private elements
   private:
    int age;

    // public elements
   public:
    void displayAge(int a) {
        age = a;
        cout << "Age = " << age << endl;
    }
};
```

```cpp
int main() {

    int ageInput;

    // declare an object
    Sample obj1;

    cout << "Enter your age: ";
    cin >> ageInput;

    // call function and pass ageInput as argument
    obj1.displayAge(ageInput);

    return 0;
}
```

Restriction for private Access Modifier

- In main(), the object obj1 cannot directly access the class variable age.

  // error

  cin >> obj1.age;

- We can only indirectly manipulate age through the public function displayAge(), since this function initializes age with the value of the argument passed to it i.e. the function parameter int a.