BASUDEV GODABARI DEGREE COLLEGE , KESAIBAHAL

Department of Computer Science

"SELF STUDY MODULE"

Module Details :

- Class – 2nd Semester (2020-21) Admission Batch
- Subject Name : COMPUTER SCIENCE
- Paper Name : Programming using C++

UNIT – 2 : STRUCTURE

2.1 Introduction to Classes and object
2.2 Member Function
2.3 Out Side Function Line
2.4 Nested Member Function
2.5 Array within Class
2.6 Memory Allocation
2.7 Static Member Static Member Function
2.8 Function Argument
2.9 Friend Function

Learning Objective

After Learning this unit you should be able to

- Know the Concept of Object and Class
- How memory allocate and reallocate
- Member function
- Able to Understand the how array use within the class

You Can use the Following Learning Video link related to above topic :

https://www.youtube.com/watch?v=7pPtlFn8H48

https://www.youtube.com/watch?v=dkk0eIx3nSU

https://www.youtube.com/watch?v=iECJ62oXQms

You Can also use the following Books :

1. E. Balgurusawmy, Object Oriented Programming with C++, 4/e (TMH).
2. Paul Deitel, Harvey Deitel, "C++: How to Program", 9/e. Prentice Hall.

Reference Books:

1. Bjarne Stroustroup, Programming - Principles and Practice using C++,

And also you can download any book in free by using the following website.

- https://www.pdfdrive.com/

Classes And Objects:

The main purpose of C++ programming is to add object orientation to the C programming language and classes are the central feature of C++ that supports object-oriented programming and are often called user-defined types.

A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package. The data and functions within a class are called members of the class.

C++ Class Definitions

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

A class definition starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword **class** as follows –

```
class Box {
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};
```

The keyword **public** determines the access attributes of the members of the class that follows it. A public member can be accessed from outside the class anywhere within the scope of the class object. You can also specify the members of a class as **private** or **protected** which we will discuss in a sub-section.

Member Function :-

A member function of a class is a function that has its definition or its prototype within the class definition like any other variable. It operates on any object of the class of which it is a member, and has access to all the members of a class for that object.

Let us take previously defined class to access the members of the class using a member function instead of directly accessing them –

```
class Box {
   public:
      double length;          // Length of a box
      double breadth;         // Breadth of a box
      double height;          // Height of a box
      double getVolume(void);// Returns box volume
};
```

Member functions can be defined within the class definition or separately using **scope resolution**

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box –

```
Box Box1;          // Declare Box1 of type Box
Box Box2;          // Declare Box2 of type Box
```

Both of the objects Box1 and Box2 will have their own copy of data members.

Accessing the Data Members

The public data members of objects of a class can be accessed using the direct member access operator (.). Let us try the following example to make the things clear –

```
#include <iostream>

using namespace std;

class Box {
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};

int main() {
   Box Box1;          // Declare Box1 of type Box
   Box Box2;          // Declare Box2 of type Box
   double volume = 0.0;     // Store the volume of a box here

   // box 1 specification
   Box1.height = 5.0;
   Box1.length = 6.0;
   Box1.breadth = 7.0;

   // box 2 specification
   Box2.height = 10.0;
   Box2.length = 12.0;
   Box2.breadth = 13.0;

   // volume of box 1
   volume = Box1.height * Box1.length * Box1.breadth;
   cout << "Volume of Box1 : " << volume <<endl;

   // volume of box 2
   volume = Box2.height * Box2.length * Box2.breadth;
   cout << "Volume of Box2 : " << volume <<endl;
   return 0;
}
```

hen the above code is compiled and executed, it produces the following result –

```
olume of Box1 : 210
olume of Box2 : 1560
```

t is important to note that private and protected members can not be accessed directly using direct member access operator (.). We will learn how private and protected members can be accessed.

Classes and Objects in Detail

So far, you have got very basic idea about C++ Classes and Objects. There are further interesting concepts related to C++ Classes and Objects which we will discuss in various sub-sections listed below –

r.No	Concept & Description
	Class Member Functions A member function of a class is a function that has its definition or its prototype within the class definition like any other variable.
	Class Access Modifiers A class member can be defined as public, private or protected. By default members would be assumed as private.
	Constructor & Destructor A class constructor is a special function in a class that is called when a new object of the class is created. A destructor is also a special function which is called when created object is deleted.
	Copy Constructor The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously.
	Friend Functions A **friend** function is permitted full access to private and protected members of a class.
	Inline Functions With an inline function, the compiler tries to expand the code in the body of the function in place of a call to the function.
7	this Pointer Every object has a special pointer **this** which points to the object itself.
8	Pointer to C++ Classes A pointer to a class is done exactly the same way a pointer to a structure is. In fact a class is really just a structure with functions in it.

Objects

A class provides the blueprints for objects, so basically an object is created fro class. We declare objects of a class with exactly the same sort of declaration that declare variables of basic types. Following statements declare two objects of class B –

```
Box Box1;
Box Box2;          // Declare Box1 of type Box
                   // Declare Box2 of type Box
```

Both of the objects Box1 and Box2 will have their own copy of data members.

Accessing the Data Members

The public data members of objects of a class can be accessed using the direct member access operator (.). Let us try the following example to make the things clear –

```
#include <iostream>

using namespace std;

class Box {
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};

int main() {
   Box Box1;          // Declare Box1 of type Box
   Box Box2;          // Declare Box2 of type Box
   double volume = 0.0;     // Store the volume of a box here

   // box 1 specification
   Box1.height = 5.0;
   Box1.length = 6.0;
   Box1.breadth = 7.0;

   // box 2 specification
   Box2.height = 10.0;
   Box2.length = 12.0;
   Box2.breadth = 13.0;

   // volume of box 1
   volume = Box1.height * Box1.length * Box1.breadth;
   cout << "Volume of Box1 : " << volume <<endl;

   // volume of box 2
   volume = Box2.height * Box2.length * Box2.breadth;
   cout << "Volume of Box2 : " << volume <<endl;
   return 0;
}
```

Member Function

A member function is defined outside the class using the ::(double colon symbol) scope resolution operator. This is useful when we did not want to define the function within the main program, which makes the program more understandable and easy to maintain.

The general syntax of the member function of a class outside its scope :

< return_type > < class_name > :: < member_function > (arg1, arg2.... argN)

The type of member arguments in the member function must exactly match with the types declared in the class definition of the < class_name >. The scope resolution operator (::) is used along with the class name in the header of the function definition. It identifies the function as a member of a particular class. Without the scope resolution operator, the function definition would create an ordinary function, subject to the usual function rules of access and scope.

The following program segment shows how a member function is declared outside the class declaration and how the object of a class is created to use the class's methods and variables :-

```
#include <iostream>
using namespace std;
class TestMemberFunction
{
        public :
                int x, y;
                int sum();  // Memebr Function Declaration
};
int TestMemberFunction :: sum()
{
        return (x+y);
}
int main ()
{
        TestMemberFunction MemberFunction1;
        MemberFunction1.x = 1;
        MemberFunction1.y = 2;
        cout << "Sum is : " << MemberFunction1.sum();
        return 0;
```

→ inside the class

→ out side the class

Nesting of Member Functions :-

A member function may call another member function within itself. This is called nesting of member functions. A member function can access not only the public functions but also the private functions of the class it belongs to.

Let us see an simple example of showing nesting of the member functions :-

```
#include <iostream>
using namespace std;
class NestingMemberFunction
{
        private :
                int myint;
        public :
                void set();
                int get();
};
void NestingMemberFunction :: set()
{
        myint = -1;
}
int NestingMemberFunction :: get()
{
        set();
        return myint;
}
int main ()
{
        int printvariable;
        NestingMemberFunction NestingMemberFunction1;
        printvariable = NestingMemberFunction1.get();
        cout << "The variable is : " << printvariable;
        return 0;
}
```

Private Member Function

A function declared inside the class's private section is known as "**private member function**". A **private member function** is accessible through the only public member function. (Read more: data members and member functions in C++).

Example:

In this example, there is a class named "**Student**", which has following data members and member functions:

- **Private**
  - **Data members**
    - rNo - to store roll number
    - perc - to store percentage

- **Member functions**
  - `inputOn()` - to print a message "**Input start...**" before reading the roll number and percentage using public member function.
  - `inputOff()` - to print a message "**Input end...**" after reading the roll number and percentage using public member function.

- **Public**
  - **Member functions**
    - `read()` - to read roll number and percentage of the student
    - `print()` - to print roll number and percentage of the student

Here, `inputOn()` and `inputOff()` are the private member functions which are calling inside public member function `read()`.

Program:

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

class Student
{
	private:
		int rNo;
		float perc;
		//private member functions
		void inputOn(void)
		{
			cout<<"Input start..."<<endl;
		}
		void inputOff(void)
		{
			cout<<"Input end..."<<endl;
		}

	public:
		//public member functions
		void read(void)
		{
			//calling first member function
			inputOn();
			//read rNo and perc
			cout<<"Enter roll number: ";
			cin>>rNo;
			cout<<"Enter percentage: ";
			cin>>perc;
```

```
                    //calling second member function
                    inputOff();
                }
                void print(void)
                {
                    cout<<endl;
                    cout<<"Roll Number: "<<rNo<<endl;
                    cout<<"Percentage: "<<perc<<"%"<<endl;
                }
};

//Main code
int main()
{
    //declaring object of class student
    Student std;

    //reading and printing details of a student
    std.read();
    std.print();

    return 0;
}
```

Arrays within a Class

- Arrays can be declared as the members of a class.
- The arrays can be declared as private, public or protected members of the class.
- To understand the concept of arrays as members of a class, consider this example.

A program to demonstrate the concept of arrays as class member

Example

```
#include<iostream>
const int size=5;
class student
{
int roll_no;
 int marks[size];
 public:
void getdata ();
void tot_marks ();
};
```

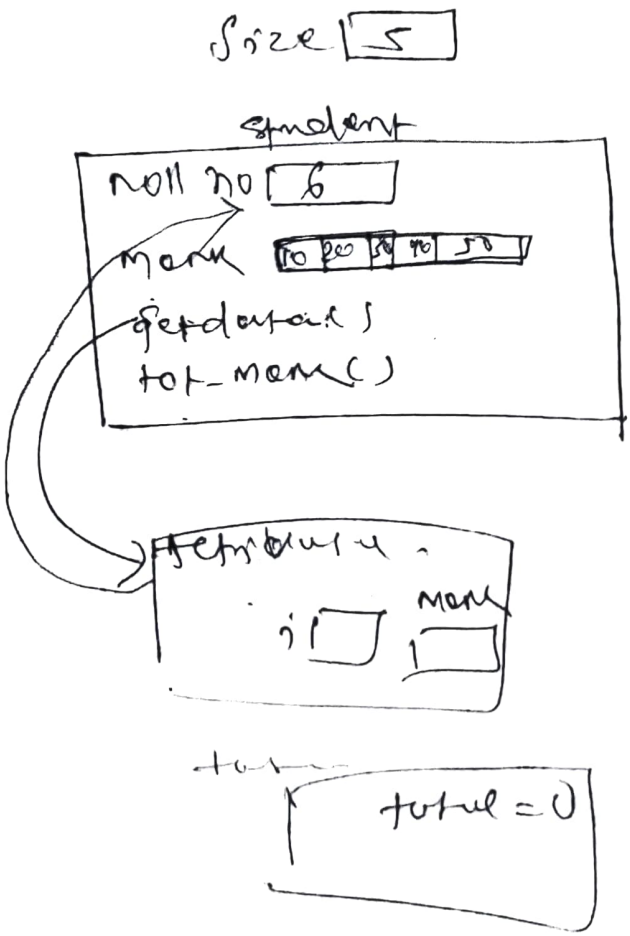
```
void student :: getdata ()
{
cout<<"\nEnter roll no: ";
 Cin>>roll_no;
for(int i=0; i<size; i++)

  {
 cout<<"Enter marks in subject"<<(i+1)<<": ";
cin>>marks[i] ;
}

 void student :: tot_marks() //calculating total marks
 {
int total=0;
 for(int i=0; i<size; i++)
 total+ = marks[i];
cout<<"\n\nTotal marks "<<total;
    }

 void main() student stu;
 stu.getdata() ;
 stu.tot_marks() ;
 getch();
  }
```

Output:



Enter roll no: 101
Enter marks in subject 1: 67
Enter marks in subject 2 : 54
Enter marks in subject 3 : 68
Enter marks in subject 4 : 72
Enter marks in subject 5 : 82
Total marks = 343

Memory Allocation for Object of Class

Once you define class it will not allocate memory space for the data member of the class. The memory allocation for the data member of the class is performed separately each time when an object of the class is created. Since member functions defined inside class remains same for all objects, only memory allocation of member function is performed at the time of defining the class. Thus memory allocation is performed separately for different object of the same class. All the data members of each object will have separate memory space.

-->We can also dynamically allocate objects.
As we know that **Constructor** is a member function of a class which is called whenever a new object is created of that class. It is used to initialize that object. **Destructor** is also a class member function which is called whenever the object goes out of scope.

Destructor is used to release the memory assigned to the object. It is called in these conditions.

- When a local object goes out of scope
- For a global object, operator is applied to a pointer to the object of the class

We again use pointers while dynamically allocating memory to objects.

Let's see an example of array of objects.

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

class A
{
        public:
        A() {
          cout << "Constructor" << endl;
        }
        ~A() {
          cout << "Destructor" << endl;
      }
};

int main()
{
        A* a = new A[4];
        delete [] a; // Delete array
        return 0;
}
```

Static data members

Static data members are class members that are declared using the static keyword. There is only one copy of the static data member in the class, even if there are many class objects. This is because all the objects share the static data member. The static data member is always initialized to zero when the first class object is created.

The syntax of the static data members is given as follows –

static data_type data_member_name;

In the above syntax, static keyword is used. The data_type is the C++ data type such as int, float etc. The data_member_name is the name provided to the data member.

→A program that demonstrates the static data members in C++ is given as follows –

Example

```
#include <iostream>
#include<string.h>
using namespace std;
class Student
{
  private:
  int rollNo;
  char name[10];
  int marks;
  public:
  static int objectCount;
  Student() {
    objectCount++;
  }

  void getdata() {
    cout << "Enter roll number: "<<endl;
    cin >> rollNo;
    cout << "Enter name: "<<endl;
    cin >> name;
    cout << "Enter marks: "<<endl;
    cin >> marks;
}

void putdata() {
   cout<<"Roll Number = "<< rollNo <<endl;
   cout<<"Name = "<< name <<endl;
```

```
    cout<<"Marks = "<< marks <<endl;
     cout<<endl;
  }
};
int Student::objectCount = 0;
int main(void) {
   Student s1;
   s1.getdata();
   s1.putdata();
   Student s2;

   s2.getdata();
   s2.putdata();
   Student s3;

   s3.getdata();
   s3.putdata();
   cout << "Total objects created = " << Student::objectCount << endl;
   return 0;
  }
```

The output of the above program is as follows −

```
Enter roll number: 1
Enter name: Mark
Enter marks: 78
Roll Number = 1
Name = Mark
Marks = 78

Enter roll number: 2
Enter name: Nancy
Enter marks: 55
Roll Number = 2
Name = Nancy
Marks = 55

Enter roll number: 3
Enter name: Susan
Enter marks: 90
Roll Number = 3
Name = Susan
Marks = 90
Total objects created = 3
```

In the above program, the class student has three data members denoting the student roll number, name and marks. The objectCount data member is a static data member that contains the number of objects created of class Student. Student() is a constructor that increments objectCount each time a new class object is created.

There are 2 member functions in class. The function getdata() obtains the data from the user and putdata() displays the data. The code snippet for this is as follows –

```
class Student {
  private:
  int rollNo;
  char name[10];
  int marks;
  public:
  static int objectCount;
  Student() {
    objectCount++;
  }

  void getdata() {
    cout << "Enter roll number: "<<endl;
    cin >> rollNo;
    cout << "Enter name: "<<endl;
    cin >> name;
    cout << "Enter marks: "<<endl;
    cin >> marks;
  }

  void putdata() {
    cout<<"Roll Number = "<< rollNo <<endl;
    cout<<"Name = "<< name <<endl;
    cout<<"Marks = "<< marks <<endl;
    cout<<endl;
  }
};
```

In the function main(), there are three objects of class Student i.e. s1, s2 and s3. For each of these objects getdata() and putdata() are called. At the end, the value of objectCount is displayed. This is given below –

```
int main(void) {
  Student s1;

   s1.getdata();
  s1.putdata();

   Student s2;
   s2.getdata();
   s2.putdata();

   Student s3;
   s3.getdata();
   s3.putdata();

   cout << "Total objects created = " << Student::objectCount << endl;

   return 0;
}
```

Constructors in C++

What is constructor?

A constructor is a member function of a class which initializes objects of a class. In C++, Constructo automatically called when object(instance of class) create. It is special member function of the class

1. program to illustrate the // concept of Constructors

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

class construct {
public:
    int a, b;

    // Default Constructor
    construct()
    {
        a = 10;
        b = 20;
    }
};

int main()
{
    // Default constructor called automatically
    // when the object is created
    construct c;
    cout << "a: " << c.a << endl
         << "b: " << c.b;
    return 1;
}
```

OUTPUT:

A: 10
B: 20

Parameterized Constructors:

It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.

2. // CPP program to illustrate // parameterized constructors

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

class Point {
private:
    int x, y;

public:
    // Parameterized Constructor
    Point(int x1, int y1)
    {
        x = x1;
        y = y1;
    }
```

```
    {
        return x;
    }
    int getY()
    {
        return y;
    }
};

int main()
{
    // Constructor called
    Point p1(10, 15);

    // Access values assigned by constructor
    cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();

    return 0;
}
```

Output:

```
p1.x = 10, p1.y = 15
```

Destructors in C++

What is destructor?
Destructor is a member function which destructs or deletes an object.When is destructor called?
A destructor function is called automatically when the object goes out of scope:
(1) the function ends
(2) the program ends
(3) a block containing local variables ends
(4) a delete operator is called
How destructors are different from a normal member function?
Destructors have same name as the class preceded by a tilde (~)
Destructors don't take any argument and don't return anything

```
class String
{
private:
    char *s;
    int size;
public:
    String(char *); // constructor
    ~String();      // destructor
};

String::String(char *c)
{
    size = strlen(c);
    s = new char[size+1];
    strcpy(s,c);
}

String::~String()
{
    delete []s;
}
```