MARUDHAR KESARI JAIN COLLEGE FOR WOMEN,VANIYAMBADI

PG AND RESEARCH DEPARTMENT OF COMPUTER SCIENCE

CLASS: I BSC COMPUTER SCIENCE

SUBJECT CODE: 23UCS11

SUBJECT NAME: Object Oriented Programming Concepts using C++

SYLLABUS

UNIT II

Classes and Objects: Declaring Objects – Defining Member Functions –Static Member variables and functions – array of objects –friend functions – Overloading member functions – Bit fields and classes – Constructor and destructor with static members.

CLASS:

- A Class is a user-defined data type that has data members and member functions.
- Data members are the data variables and member functions are the functions used to manipulate these variables together, these data members and member functions define the properties and behavior of the objects in a Class.
- In the above example of class *Car*, the data member will be *speed limit*, *mileage*, etc, and member functions can be *applying brakes*, *increasing speed*, etc.

OBJECT

An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Declaring 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, you need to create objects.

Syntax

Class Name Object Name;

Accessing data members and member functions: The data members and member functions of the class can be accessed using the dot('.') operator with the object. For example, if the name of the object is *obj* and you want to access the member function with the name *print Name()* then you will have to write *obj. print Name()*.

Accessing Data Members

The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member. This access control is given by Access modifiers in C++. There are three access modifiers: **public, private, and protected**.

```
C++ program to demonstrate accessing of data members
#include <bits/ std c++.h>
using namespace std;
class Geeks {
    // Access specifier
public:
    // Data  Members
    string geek name;
    // Member Functions()
    void printname() { cout << "Geekname is:" << geekname; }
};
int main()
{
    // Declare an object of class geeks
    Geeks obj1;
    // accessing data member
    obj1.geekname = "Abhi";
    // accessing member function
    obj1.printname();
    return 0;
}
```

Member Functions in Classes

There are 2 ways to define a member function:

- Inside class definition
- Outside class definition

To define a member function outside the class definition we have to use the **scope resolution:: operator** along with the class name and function name.

```
// C++ program to demonstrate function
// declaration outside class
#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:
    string geekname;
    int id;
 // printname is not defined inside class definition
    void printname();
    // printid is defined inside class definition
    void printid()
    {
        cout <<"Geek id is: "<<id;
```

```
    }
};
// Definition of printname using scope resolution operator ::
void Geeks::printname()
{
    cout <<"Geekname is: "<<geekname;
}
int main() {
    Geeks obj1;
    obj1.geekname = "xyz";
    obj1.id=15;
      // call printname()
    obj1.printname();
    cout << endl;
    // call printid()
```

```
    obj1.printid();

    return 0;

}
```

Output

Geekname is: xyz

Defining Member Functions

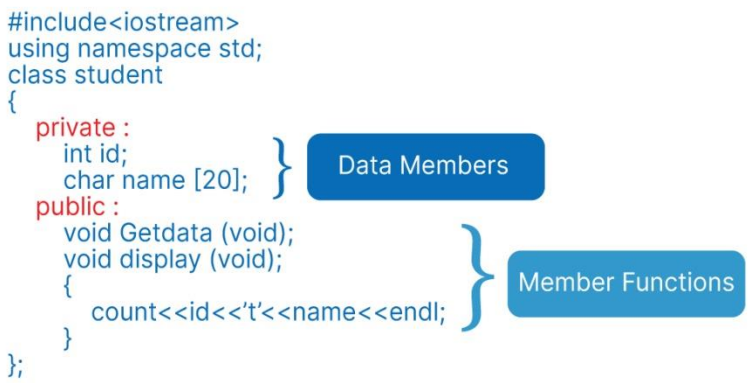
What is a member function?

In C++, a member function is a function that is associated with a specific object or class. Member functions are defined within the class definition, and they have access to the data members and other member functions of the class. They can be used to perform operations on the object's data and interact with objects of the same class.

A member function can be invoked by using dot notation, which is the object's name followed by the name of the member function and its parameters. For example, if we have a class called "Person" with a member function called "speak()", we can call this function for an object of that class called "obj1" using the notation "obj1.speak()".

When a member function is defined as "const", it can be called on a constant object and will not modify the object.

One important thing to remember when working with member functions is that, by default, they do not have access to the non-static member variables of the class unless this pointer is used to reference them explicitly.



In the above figure, we have **data members** and **member functions** in class named **student.**

Data members or class members are nothing, just simple variables. And Member functions are nothing, just simple functions we use in a program.

These member functions are defined with the "static" keyword. They do not have access to the data members of the class and can only access static data members. They can be called without creating an object of the class.

Friend member functions

These member functions are defined outside the class definition. But they have access to the private and protected members of the class. They are declared inside the class definition using the "friend" keyword.

FRIEND FUNCTION

Constant member functions

These member functions are defined with the "const" keyword. They can be called on a constant object and will not modify the object. They are useful for accessing the data members of an object without modifying them.

Virtual member functions

These member functions are defined with the “virtual” keyword. They are used in polymorphism when you want a base class pointer or reference to point to a derived class object.

Virtual Functions in C++

Operator overloading member functions

These member functions are used to overload operators for a class, allowing you to use operators such as +, -, *, /, etc., with objects of that class.

Operator Overloading in C++.

Default, copy and move constructors and assignment operators

They are also types of member functions; they are special member functions that are automatically generated by the compiler when needed.

Static Member Function in C++

Static Member Function in a class is the function that is declared as static because of which function attains certain properties as defined below:

- A static member function is independent of any object of the class.
- A static member function can be called even if no objects of the class exist.
- A static member function can also be accessed using the class name through the scope resolution operator.
- A static member function can access static data members and static member functions inside or outside of the class.
- Static member functions have a scope inside the class and cannot access the current object pointer.
- You can also use a static member function to determine how many objects of the class have been created.

The reason we need Static member function:

- Static members are frequently used to store information that is shared by all objects in a class.

For instance, you may keep track of the quantity of newly generated objects of a specific class type using a static data member as a counter. This static data member can be increased each time an object is generated to keep track of the overall number of objects.

Declaring Member Variables

At minimum, a member variable declaration has two components: the data type of the variable and its name.

type variableName; // minimal member variable declaration

A minimal variable declaration is like the declarations that you write for variables used in other areas of your Java programs such as local variables or method parameters. The following code snippet declares an integer member variable named anInteger within the class IntegerClass.

class IntegerClass {

- int anInteger;
- . . .
- // define methods here
- . . .
- }
- Notice that the member variable declaration appears within the body of the class implementation but not within a method. This positioning within the class body determines that the variable is a member variable.
- Like other variables in Java, member variables must have a type. A variable's type determines the values that can be assigned to the variable and the operations that can be performed it. You should already be familiar with data types in Java through your reading of
- Variables and Data Types
- in the previous lesson.
- A member variable's name can be any legal Java identifier and by convention begins with a lower case letter (class names typically begin with upper case letters). You cannot declare more than one member variable with the same name

in the same class. However, a member variable and a method can have the same name. For example, the following code is legal:

```
class IntegerClass
{
int anInteger;
 int anInteger()
 {       // a method with the same name as a member variable

}
}
```

ARRAY OF OBJECT

An array in C/C++ or be it in any programming language is a collection of similar data items stored at contiguous memory locations and elements can be accessed randomly using indices of an array. They can be used to store the collection of primitive data types such as int, float, double, char, etc of any particular type. To add to it, an array in C/C++ can store derived data types such as structures, pointers, etc. Given below is the picture representation of an array.

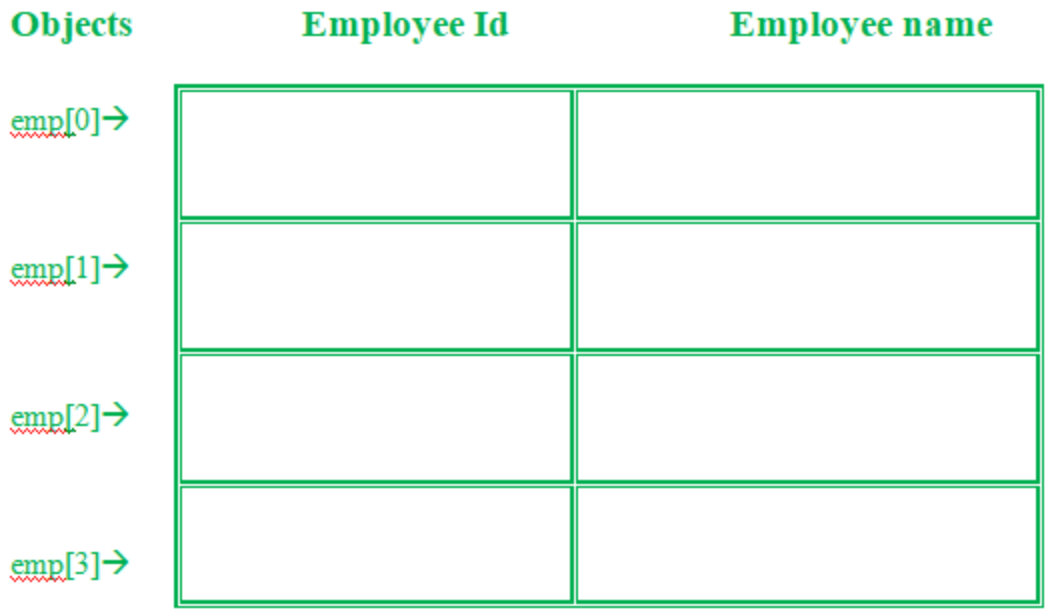
Syntax:

ClassName ObjectName[number of objects];

The Array of Objects stores *objects*. An array of a class type is also known as an array of objects.

Example#1:

Storing more than one Employee data. Let's assume there is an array of objects for storing employee data emp[50].



Below is the C++ program for storing data of one Employee:

```
// C++ program to implement

// the above approach

#include<iostream>

using namespace std;



class Employee

{
```

```
  int id;
  char name[30];
  public:
  void getdata();//Declaration of function
  void putdata();//Declaration of function
};
void Employee::getdata(){//Defining of function
  cout<<"Enter Id : ";
  cin>>id;
  cout<<"Enter Name : ";
  cin>>name;
}
void Employee::putdata(){//Defining of function
  cout<<id<<" ";
  cout<<name<<" ";
```

```
 cout<<endl;

}

int main(){

 Employee emp; //One member

 emp.getdata();//Accessing the function

 emp.putdata();//Accessing the function

 return 0:

}
```

Let's understand the above example –

- In the above example, a class named Employee with id and name is being considered.
- The two functions are declared-
    - **getdata():** Taking user input for id and name.
    - **putdata():** Showing the data on the console screen.

This program can take the data of only one Employee. What if there is a requirement to add data of more than one Employee. Here comes the answer Array of Objects. An array of objects can be used if there is a need to store data of more than one employee. Below is the C++ program to implement the above approach

Advantages of Array of Objects:

1. The array of objects represent storing multiple objects in a single name.
2. In an array of objects, the data can be accessed randomly by using the index number.
3. Reduce the time and memory by storing the data in a single variable.

Whether you're preparing for your first job interview or aiming to upskill in this ever-evolving tech landscape, are your key to success. We provide top-quality content at affordable prices, all geared towards accelerating your growth in a time-bound manner.

BIT FIELDS:

bit field by specifying the data type and then the field name with the desired width in bits.

Applications of Bit Fields in C++

1. It is majorly used for memory optimization as it allows us to use only the necessary number of bits to represent a member variable which reduces memory overhead.
2. It can be used to compress the data so that it can take less time to transfer data from one point to another over the network.
3. It can be used to design hardware registers that have specific bit structures.
4. In graphics applications, color components, pixel formats, and image data often have specific bit requirements which can be customized using bit fields.

What is Constructor in C++?

A constructor is a special member function of a class and shares the same name as of class, which means the constructor and class have the same name. Constructor is called by the compiler whenever the object of the class is created, it allocates the memory to the object and initializes class data members by default values or values passed by the user while creating an object. Constructors don’t have any return type because their work is to just create and initialize an object.

The basic syntax of the constructor is given below:

```
class class_name{ private:

  // private members

public:   // declaring constructor
```

```
 class_name({parameters}){

   // constructor body

 }

};
```

How many types of Constructors are present in C++?

There are four types of constructors in c++

- Default constructor
- Parameterized constructor
- Copy Constructor
- Dynamic Constructor

Default Constructor

Default constructor is also known as a zero-argument constructor, as it doesn't take any parameter. It can be defined by the user if not then the compiler creates it on his own. Default constructor always initializes data members of the class with the same value they were defined.

Syntax

```
 class class_name{

  private:  // private members

 public:
```

```
  // declaring default constructor

  class_name()

  {

    // constructor body

  }

};
```

Parameterized Constructor

Parameterized constructor is used to initialize data members with the values provided by the user. This constructor is basically the upgraded version of the default constructor. We can define more than one parameterized constructor according to the need of the user, but we have to follow the rules of the function overloading.

Dynamic Constructor

When memory is allocated dynamically to the data members at the runtime using a new operator, the constructor is known as the dynamic constructor. This constructor is similar to the default or parameterized constructor; the only difference is it uses a new operator to allocate the memory.

What is Destructor in C++?

Destructor is just the opposite function of the constructor. A destructor is called by the compiler when the object is destroyed and its main function is to deallocate the memory of the object. The object may be destroyed when the program ends, or local objects of the function get out of scope when the function ends or in any other case. Destructor has the same as of the class with prefix tilde(~) operator and it cannot be

overloaded as the constructor. Destructors take no argument and have no return type and return value.

How Constructor and Destructor are called when the object is Created and Destroyed

As constructor is the first function called by the compiler when an object is created and the destructor is the last class member called by the compiler for an object. If the constructor and destructor are not declared by the user, the compiler defines the default constructor and destructor of a class object.

Let’s see a code to get the proper idea of how constructor and destructor are called:

First, we will create a class with single parametrized constructors and a destructor. Both of them contain print statements to give an idea of when they are called.