C++ Syllabus covered
Note
(2nd Semester Course
of
BICTE)

Prepared by the Lecturer
Rajesh Parajuli
9847548279
www.parajulirajesh.com.np

Object oriented Programming
Language (C++)

2.1 Concept of object and class

Class: A class is a blueprint or template for creating objects. It defines a user-defined datatype by encapsulating data and methods that work on the data into one single unit.

Syntax:

```
class class_name
{
  public:
  // data members.
    int x;
    string y;
  // member functions
    void show();
    void display();
};
```

object: An object is an instance of a class. It is a specific realization of the class with actual values.

Syntax:

```
class_name obj;
obj.x = value;
obj.y = "value";
obj.show();
obj.display();
```

when a class is defined, no memory is allocated until an object of that class is created.

Q. Program example to demonstrate object and class.

```
#include <iostream>
using namespace std;
class car {
public:
    string brand;
    String model;
    int year;
void display ()
    cout << brand;
    cout << model;
    cout << year;
};
int main () // main function
    {
    car car1; // object creation
    car1. brand = "Bmw";
    car1. model = "Latest";
    car1. year = 2081;
    car1. display ();
    return 0;
}
```

Data members / instance variables

member function

output:

```
Bmw
Latest
2081
```

2.2 Define Data member and member function

Data member: variables that hold data associated with a class and its objects. Represents fields / identity.

member function: functions that operate on the data members of the class. Represents Action / Behaviour.

Data members and member functions of the class can be accessed using the dot ('.') operator with the object.

There are two ways to define member functions:

1. Inside class
2. outside class

1. Inside class :- Define member function behavior inside class.

Example:

```
class A{
public:
string name;
int id;
void details()
{ cout<<name;
cout<<id;
}
int main()
{
A a;
a.details();
}
```

2. outside class → declare the function inside the class and use 'scope resolution operator outside of class to define this member function.

Example:

```
class A
{
public:
    string name;
    int id;
    void details();
};
void A :: details()
{
cout<<name;
cout<<id;
}
int main()
{A a;
a.details();
return 0;
}
```

2.3 Create object and Access member function.

In C++, we use the class name to create object so we can say object is an instance of class. And class is blueprint for creating an object.

Syntax

```
Class A
{
  // data members;
  // member functions;
};
int main()
{
  A obj; // creating an object
}
```

Access member function. : After creating object, we use . operator to access the member function for example.

```
Class A
{
  int x;
  int y;
  public:
  void get_data()
  { cout<<x<<y;
  }
};
int main(){
  A obj;
  obj.get_data();
```

C++ program to find area of Rectangle.

```
Class Rectangle {
public:
  int length;
  int width;
  int areaofRec()
  {return length*width;
  }
};
int main(){
  Rectangle rect;
  rect.length=10;
  rect.width=5;
  cout<<"area:"<<rect.areaof-
  Rec()<<endl;
  return 0;
}
```

2.4 making outer function Inline

Inline functions are defined outside the class definition but declared inside the class. The compiler replaces the function call with the function code. It is faster than the execution of normal function and cannot be perform with:

(i) Loop (for, while, do-while)
(ii) Recursion
(iii) If a function has static variables.
(iv) whether a function uses a goto or switch statement

Syntax:

```
inline return_type function_name(parameters)
{
// function code
}
```

Example:

```
#include <iostream>
using namespace std;
class ~~Arithmetic~~ operation
{
public:
int a, b, add, sum, mul;
float div;
public:
  void get();
  void sum();
  void difference();
  void product();
  void division();
};
```

```
inline void operation :: get ()
{
  cout << "enter a value:";
  cin >> a;
  cout << "enter b value :";
  cin >> b;
}
 inline void operation :: sum()
 {
  add = a+b;
  cout << "add of two numbers: " << a+b << "\n";
 }
 inline void operation :: product()
  { mul = a*b;
    cout << " product of two numbers: " << a*b << "\n";
  }
    inline void operation :: division()
   { div = a/b;
    cout << "Division of two numbers : " << a/b << "\n";
   }
    int main()
   {
     operation S;
     S.get();
     S.sum();
     S.product();
     S.division();
   return 0;
   }
```

2.5 Array with in class

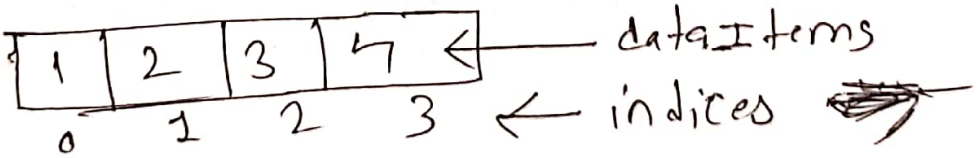
Class is the combinations of Data members and member function. Class is also the combination of both primitive & Derived datatypes (Structures, Pointers).

Array is a collection of homogeneous variables.
Array is a collection of similar types of data items.
Here, Arrays can be declared as the members of a Class. The arrays can be declared as private, public or protected members of the class.

Syntax: You can have arrays as data members within a class.

```
class classname
{
  int array[size];
};
```

→ Similar data Items stored at contiguous memory locations. and elements can be accessed using indices of an array.



Example 1

```
#include<iostream>
using namespace std;
class classroom{
public:
int studentAges[5];
void displayAges()
{for(int i=0; i<5; i++)
{cout<<"student"<<i+1<<"age:
 "<<studentAges[i]<<endl;
}
};
int main(){
 classroom class1;
 class1.studentAges[0]=15;
 class2.studentAges[1]=16;
        "       "  [4]=20;
 class1.displayAges();
 return 0;
}
```

C++ program to store the salaries of 5 employees in an array and display those salaries.

```
#include <iostream>
using namespace std;
class emp_salaries
{
    float salary[5];
    public:
    void putSalaries()
    {
    cout << "enter salaries of 5 employee :";
    for(int i=0; i<5; i++)
    {cout << "employee" << (i+1) << ": ";
     cin >> salaries[i];
    }}
    void displaySalaries()
    {
     cout << "| Salaries of the employees are:"
     << endl;
     for(int i=0; i<5; i++)
     {
      cout << "employee" << (i+1) << ":"
        << Salaries[i] << endl;
     }
};}
```

```
int main ( )
{
  Employee salaries employees;
  employees.inputSalaries();
  employees.displaySalaries();
}  return 0;
```

2.6 Array of objects in C++

The array of objects Stores multiple objects of the same class.

Syntax:

```
Class_Name object_Name[number of objects];
```

Example:

```
#include <iostream>
 using namespace std;
Class student{
public:
 int id;
 string name;
 void display()
 {cout<<"ID: "<<id;
  cout<<"Name:"<<name;
};
 int main(){
 student students[3]; //Array of objects
 students[0].id = 101;
 students[0].name = "rajesh";
 students[1].id = 102;
 students[1].name = "Hari";
 student[2].id=103;
 student[2].name="suman";
 //Accessing array of objects
 for(int i=0; i<3; i++)
 {
  students[i].display();
 }return 0;
}
```

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 index numbers.
3. Reduce the time and memory by storing the data in a single variable.

2.7 Static Data member and Static function

Static data members are class members that are declared using static keywords. A static data member is a data member of a class that is shared by all objects of the class. only one copy of the static data member exists, and it is common to all objects. This means that if one object changes the value of the static data member, the change is reflected across all objects.

Static functions can only access static data members.

Static ~~data~~ members are initialized before any object of the class is created.

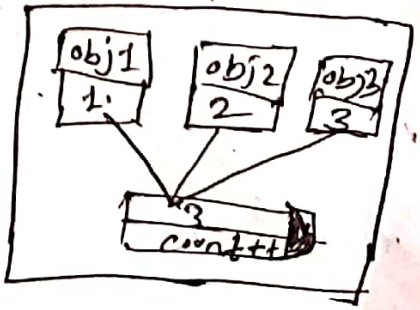
Example:

```
Class counter{
public:
    static int count;
    counter()
    {
     count++;
    }
    static void displaycount()
     {
      cout << "count " << count << endl;
     }
};
int counter::count = 0;
int main(){
 counter c1, c2, c3;
 counter::displaycount();
 return 0;
}
```

Syntax:

```
data_type classname::
 variable_name = value;
```

"A static data member is a class variable that is shared by all objects of the class, rather than being unique to each object."



Static member

Declaration and initialization

→ Static data members are declared inside the class, but they must be defined and initialized outside the class.

→ Initialization is done only once, and this happens outside the class. ~~typically~~

Example

```
#include <iostream>
using namespace std;
class myclass {
public:
  static int x; // Declaration of static
                   data member
  void show()
  {
    cout << "Static variable: " ;
  }
};
int myclass :: x = 0; // Definition & initialization
                         of static data member
int main(){
myclass obj1, obj2;
obj1.show(); // output : static variable: 0
obj2.show(); // output : static variable = 0
  // Changing the static variable using obj1
  obj1.x = 10;
  obj1.show(); : output : static variable : 10
  obj2.show(); : output : static variable : 10
return 0;
}
```

Important key Points

Static data member is defined using the 'static' keyword, and it is typically declared within the class definition but outside of any member function. The static data member can be accessed using the class name, rather than an object of the class.

Static member function

Static function can be defined using "static" keyword, and it is typically declared within the class definition but outside of any member function.
We can access static member functions by using the class name and scope resolution operator.

Syntax: Accessing function

```
class_name :: function_name()
```

Static member function syntax:

```
Class ClassName
{
  //other properties & methods
  static return_type function_name(parameter-list);
};
```

extra information

A static member function does not have access to the 'this' pointer because it is not associated with any particular instance of the class.
It can be called using the class name instead of an object.

Example:-

```
#include<iostream>
using namespace std;
Class A
{public:
  static int x;
  static void abc(){
```

```
    cout << "Static function called ";
    cout << x;
}
};

int A::x=5;
int main()
{
    A::abc();

// Static member functions can also be called
   using an object, but it's not typical
    A obj;
    obj.abc();
return 0;
}
```

2.8 friend function

we know that the private section of class is allessible only and only through the public section of the same class.

what if we want to give allces to private member, a function abt side the class, In such circumstances we use the concept of friend function.

Keypoints

(i) A friend function cannot be called using the object of the class. It is called like a normal function.

(ii) friend function can use the resources of a class only using and object of the same class.

(iii) usually a friend function has object as an argument.

Example:

```
#include <iostream>
using namespace std;
class Aa{
private:
  int a;
  int b;
public:
   void get_data();
   friend int sum(Aa);
};
  void Aa::get_data()
  {
  cout<<"Enter two numbers:";
  cin>>a>>b;
}
int sum(Aa ref){
 return(ref.a+ref.b);
}
int main()
{
 Aa aaa;
 aaa.get_data();
cout<<"addition = "<<
  sum(aaa);
 return 0;
}
```

→ A friend function is not a member of a class but would have the rights to access the private members of the class.

→ friend function can't be called using the object of a class. The friend function is called directly as normal function without the use of an object.

→ Generally, the friend function has objects as its arguments.

example

* write a program to find the area of circle using friend function

```
#include<iostream>
using namespace std;
class circle{
private:
  double radius;
public:
  void setRadius(double r)
  {
    radius = r;
  }
  friend double area(circle ref);
};
double area(circle ref)
{
return 3.14159 * ref.radius * ref.radius;
}
int main(){
double r;
cout<<"Enter radius of the circle: ";
cin>>r;
circle obj;
obj.setRadius(r);
cout<<"area of circle is: "
    <<area(obj);
return 0;
}
```

2.9 Concept of Constructor and Destructor

Constructor :-

A constructor is a special member function of a class that is automatically called when an object of that class is created. The constructor has the same name as the class and does not have a return type, not even 'void'.

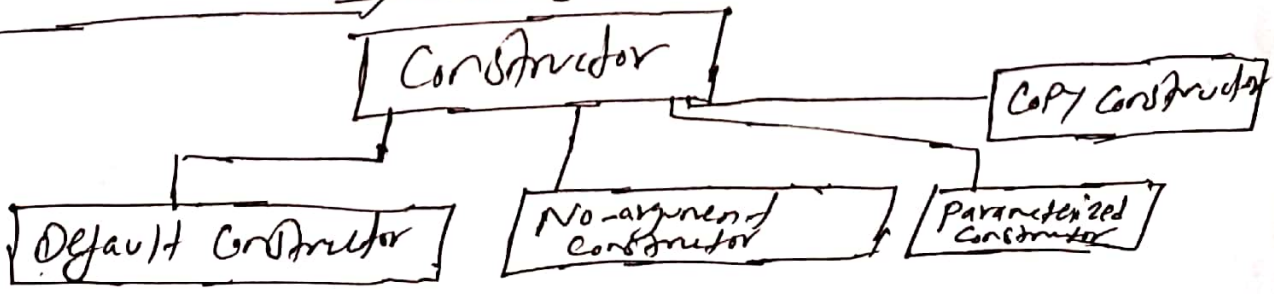
Why constructor? → It is used to ~~assigned~~ set the ~~values~~ initial state of data members of objects ~~it means by to initialize the object~~. assigning values to its attributes.

→ Constructors do not return values; hence they do not have a return types.
→ Constructor is always public.

Syntax :-

```
class class_name {
public:
Class_name (parameters)
{
// constructor body
}
};
```

2.10 Types of Constructor



Default Constructor :- A default Constructor is a Constructor that is automatically generated by the compiler if no other constructors are explicitly defined in the class. It initializes the class members to default values (eg: 'zero' for numeric types, 'false' for boleans, 'null' for strings).

Example:

```
#include <iostream>
using namespace std;
class myclass
{
  int x;
  public:
  void display()
  {
   cout << "x is " << x;
  }
};
int main()
{
  myclass obj;
  obj.display();
  return 0;
}
```

output: x is 0

② No-Argument Constructor :- A Constructor having no arguments / parameters is known as No-Argument Constructor. It is also called a zero-argument Constructor. This type of Constructor initializes data members with some fixed values.

Example :-

```
#include<iostream>
using namespace std;
class NoAr_cons{
int x;
public:
NO Ar_cons ( )
{
  x=50;
  cout << x;
}
};
int main ( )
{
 No Ar_ Cons obj;
 return 0;
}
```

output : 50

Example2 : Write a C++ program to Calculate Area of Rectangle using No-Argument Constructor.

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

```
class Rectangle
{
private:
   double length; // length of a rectangle
   double breadth; // width of rectangle
public:
   Rectangle()
   {
      length = 2.0;
      breadth = 2.0;
   }
   double Area of rectangle()
   {
      return length * breadth;
   }
};
   int main()
   {
      Rectangle rect;
      rect.Area of rectangle();
      Cout << " rect.Area of rectangle();
      return 0;
   }
```

output: 4.0

(2.10) Parameterized Constructor :- A constructor having one or more arguments is known as parameterized constructor. The values of arguments are assigned to data members of the class. parameterized Constructor pass arguments actual values while defining the object of the class.

Example:

```
#include <iostream>
using namespace std;
Class Student
{
  String S_name;
 String S_address;
 int S_rollno;
 public:
  Student (String n, String a, int r)
  {
    S_name = n;
    S_address = a;
    S_rollno = r;
  }
  void display ()
  {
    Cout << "student name is : " << S_name;
    Cout << "student address is : " << S_address;
    Cout << "Student roll number is : " << S_rollno;
  }
};
   int main ()
```

```
// creating an object using the parameterized constructor.
    Student obj("rajesh", "sukhad", 2);
    obj.display();
    return 0;
}
```

output: student name is : rajesh
student address is : sukhad
student rollnumber is : 2

Example 2:

```
#include <iostream>
using namespace std;
class sumcalc {
  int number1;
  int number2;
 public:
  void displaysum()
  { int sum = number1 + number2;
   cout << sum;
   cout << "the sum of two number is:" "<< sum;

  }
  sumcalc (int n1, int n2)
  {
    number1 = n1;
    number2 = n2;
  };
  int main()
  { creating an object
sumcalc sumcalcc(10, 20);
sumcalcc.displaysum();
return 0;
}
```

output: 30
the sum of two number is:3

Copy Constructor

A copy constructor is a special constructor used to create a new object as a copy of an existing object.

A copy constructor is a special constructor that initializes an object using another object of the same class. It is typically used when a new object is created as a copy of an existing object.

Syntax:

```
class_name (class_name &ref)
{
// body of copy constructor
}
```

Example :-

```
#include<iostream>
using namespace std;
class copycons
{
  int x;
  int y;
  public: // parameterized constructor
  int
  copycons (string a, int b)
  {
    x = a;
    y = b;
  }
  // copy constructor
  copycons (copycons &ref)
  {
  x=ref.x;
  y=ref.y;
  }
```

```
void display()
{
  cout<<"\nx: "<<x;
  cout<<"\ny: "<<y;
}
};

int main()
{
  copycons obj1(10, 20);
  obj1.display();
  copycons obj2(obj1);
  return 0;
}
```

output:-

```
x: 10
y: 20
x: 10
y: 20
```

2.11 Define Destructor

A destructor has the same name as the class but it is preceded by a tilde (~). it cannot have parameters or a return type.

Example:

```
Class Sample
{public:
 Sample ()
 {Cout << "constructor called" << endl;
 }
 ~sample ()
  {cout << "Destructor called" << end1;
 }
};
 int main ()
  {sample obj;
   return 0;
}
```

A destructor works opposite of constructor. Destructor release resource allocated space which is allocated by constructor.