OOP

OBJECTS AND CLASSES

CHAPTER 4

KABIN DEVKOTA

UDITA BISTA

TABLE OF CONTENTS

Our content is
divided into the given topics.
Each part will be described in the slides
that follow

C++ CLASSES

A Class is a group of similar objects and describes both characteristics(data members) and behavior(member functions) of the object. Classes are user defined data types that bind together data types and functions.

```
class class_name{
 //members of the class are defined here
};
```

Declaration of a class involves four attributes:
Tag name/Class name: the name by which the objects of the class are created
Data Members: the data types which makes up the class.
Member Functions/Methods: the function which operate on the data of the class.
Program Access Levels(private/public/protected): defines where the members of class can be used

C++ CLASSES

A general class construct is show below:

class class_name {
private:
//private data members and member functions

public:
//public data members and member functions

protected:
// protected data members and member functions
};

ACCESS SPECIFIERS

Access modifiers are used to implement an important aspect of Object-Oriented Programming known as Data Hiding. Access Modifiers or Access Specifiers in a class are used to assign the accessibility to the class members, i.e., they set some restrictions on the class members so that they can’t be directly accessed by the outside functions.

There are 3 types of access modifiers available in C++:

1. **Public:** All the class members declared under the public specifier will be available to everyone. The data members and member functions declared as public can be accessed by other classes and functions too. The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.

ACCESS SPECIFIERS

2. **Private:** The class members declared as private can be accessed only by the member functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of the class.

3. **Protected:** The protected access modifier is similar to the private access modifier in the sense that it can't be accessed outside of its class unless with the help of a friend class. The difference is that the class members declared as Protected can be accessed by any subclass (derived class) of that class as well.

OBJECTS AND THE MEMBER ACCESS

After the declaration of a class, memory is not allocated for the data member of the class. The declaration of class develops a template but data members cannot be manipulated unless an object of its types is created. Objects are the instances of the class just as variables are instances of basic data types. An object is declared like a normal variable, but using the class name as data type.

class_name o1, o2; *// o1 and o2 are objects of class object*

The data members and member functions(in public section) can be accessed just like in structures using the dot(.) operator.

Eg: ***o1.data;*** *//accessing data member*

o1.function(); *//accessing member function*

Note: if a pointer of the class is created '->' operator is used instead of '.'

DEFINING MEMBER FUNCTIONS

Member functions are declared inside the class but may be defined inside a class or outside the class.

Defining Member Function inside a class

A member function can be defined inside a class just like a normal function. A member function defined inside a class is automatically inline.

Eg:

```
class X{
            int a;
      public:
             void input(){
                        cout<< “Enter value of a: ”;
                         cin >> a;
             }
};
```

DEFINING MEMBER FUNCTIONS

Defining Member Function outside a class

A member function can be defined outside a class using class name and scope resolution operator (::) before the function name as shown in example below:

```
class X {
            int a;
            void input();   // function declaration inside class
};
void X::input(){
             cout <<"Enter value of 'a':";
             cin >> a;
}
```

The scope resolution operator is used to specify the class to which the member function belongs.

CONSTRUCTORS

A constructor is a special member function that can be used for the necessary initialization of the data members of an object. Some major points about constructors:

- The constructor has the same name as its class.
- The constructor is automatically called at the time of object creation.
- The constructor doesn’t have any return type not even “void” but it can take arguments.
- The constructors are always declared/defined inside the public section.

```
class class_name
{     ....
      public:
            class_name()
            {
                        //constructor body definition
            }
};
```

Default Constructor

Default constructor is the constructor with no parameter.

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

class Example
{
 int a;
 public:
  Example(){
   a = 10;
  }
  void display(){
   cout << “The value of a is ” << a <<endl;
  }
};
```

```
int main()
{
 Example e1;
 e1.display();

 return 0;
}
```

Output:

```
The value of a is 10
```

Parameterized constructor

The constructor function that can take arguments is called a parameterized constructor.

```
#include<iostream>
using namespace std;
class Example
{
    int a;
    Example();
    public:
        Example(int c){
                a = c;
        }
        void display() {
            Example e1;
            cout << a <<endl;
        }
};
```

```
Example:: Example()
{
 a = 0;
}
int main()
{
        //Example e1; //since the default constructor is not defined in public, it cannot be called
        Example e2(10);      //implicit call
        Example e3 = Example(5);    //explicit call
        e2.display();
        e3.display();
        return 0;
}
```

Output:

```
10
5
```

Copy constructor

This constructor has an argument of an object of the same type or same class as a reference. It is used for initializing an object of a class through another object of the same class.

```
#include<iostream>
using namespace std;
class Example
{
 int r, m;
 public:
  Example(){
   cout << “Default constructor” << endl;
   r = m = 0 ;
  }
Example(int x, int y){
 cout << “Parameterized constructor” << endl;
 r = x;
 m = y;
}
Example(Example &T1){
 cout << “Copy constructor” << endl;
 r = T1.r + 1;
 m = T1.r + 1;
}
void display(){
 cout << r << “ + ” << m << “i” << endl;
}
};
```

Copy constructor

```
int main()
{
 Example e1;
 Example e2(1, 2);
 Example e4(e1);     //implicit
 Example e5 = e2;   //explicit
 e4.display();
 e5.display();
 e4 = e2;       // this does not invoke the copy constructor

 e4.display();

 return 0;
}
```

Output:

```
Default constructor
Parameterized constructor
copy constructor
copy constructor
1 + 1i
2 + 3i
1 + 2i
```

Initialization List

C++ supports another method of initializing the class objects. This method is known as the initialization list in the constructor function.

constructor(argument_list) : initialization section{
//body of constructor
}

```
#include<iostream>
using namespace std;
class complex
{
 int imag;
 int real;
 public:
  complex(int x, int y): real(x), imag(y){}
  void display(){
   cout << real << " + i" << imag << endl;
 }};
```

```
int main()
{
 complex c(10,20);
 c.display();
 return 0;
}
```

Output:

```
10 + i20
```

Initialization List

When using an initialization list to initialize objects, the members are initialized in the order in which they are declared rather than in the order in which they are placed in the Initialization list. Thus, using the data member declared later cannot be used to initialize the members coming earlier. For example in previous example,

```
complex(int x):real(imag),imag(x) {}  // valid
complex(int x):real(x),imag(real) {}  //invalid
```

Note: In the initialization list, members are given value before the constructor even starts to execute.

Initialization List

Initialization list is generally used to initialize constant and reference data members.

Let us consider the following example

```
class ABC{
        const int x = 10;
        …
};
```

AND

```
class ABC{
        const int x;
        int &y;
        public:
            ABC(int a, int &b){
                x = a;
                y = b;
}};
```

Both the above classes produce errors. Constants and Reference data members must be initialized when they are declared (their memory is allocated). The solution to this problem is an initialization list. They can be initialized in the initialization list as:

```
            ABC(int a,int &b): x(a),y(b){}
```

DESTRUCTORS

It is a member function. It has same name as a class named preceded by tilda(~). It does not have any arguments and it does not have any return type(not even void). It is invoked automatically when the object of that class goes out of the scope or flushed from the memory.

```
#include<iostream>
using namespace std;
class demo
{
 int id;
 static int count;
 public:
  demo()
  {
   count++;
   id = count;
   cout << “\nID” << id << “ object created.”;
  }
  ~demo()
  {
  cout << “\nID” << id << “ object destroyed.”;
  }
};
int demo:: count = 0;

int main()
{
 demo d1, d2;
 return 0;
}
```

Output:

```
ID1 object created.
ID2 object created.
ID2 object destroyed.
ID1 object destroyed.
```

OBJECT AS FUNCTION ARGUMENT AND RETURN TYPE

Objects can also be passed as function argument or be returned by a function like normal data type.

```
#include <iostream>
using namespace std;
class complex
 {
 public:
 int r, i;
 void input()
 {
cout << "Enter real part: ";
cin >> r;
cout << "Enter imaginary part: "; cin >> i;
}
void display() {
cout << r << "+i" << i << endl;
}
complex add(complex x, complex y) {
complex t;
t.r = x.r + y.r;
t.i = x.i + y.i;
return t;
}
};

int main() {
complex c1,c2,c3; c1.input();
c2.input();
c3 = add(c1,c2);
cout << "The sum is: "; c3.display();
return 0;
}
```

ARRAY OF OBJECTS

We know that an array can be of any data type including struct. Similarly, we can also have an array of variables that are of type class. Such variables are called array of objects.

classname arrayname[arraysize];

The members can be accessed using the following syntax:

arrayname[index].datamember; //for data member
arrayname[index].function(); //for member functions

Consider the following class definition:

```
class employee{
   char name[20];
   float age;
   public:
         void getdata();
         void putdata();
};
```

```
employee e[4];   // create array of objects of size 4
```

The member functions can be accessed as:

```
e[i].name;
e[i].putdata ();
```

POINTERS TO OBJECTS AND MEMBER ACCESS

Similar to pointer of other data type, we can also create pointer type of object of class. This pointer holds the address of an object of the class. The general form of declaring the pointer type of object is:

classname *Pointername;

Similar to the pointer type variable of structure, the pointer object to class uses the arrow operator(->) to access the members of the class. The general form is

Pointerobject->member

Or,

(*Pointerobject).member

POINTERS TO OBJECTS AND MEMBER ACCESS

For instance, let us consider a class named kantipur having input() and display() as its public member function, then.

```
kantipur k;   //here k is a object of class kantipur
kantipur *p = &k; // p is a pointer that points to the object k;
```

Now, we can access input() and display() through pointer p as:

```
 p->input();
 p->display();
        Or
(*p). input();
(*p).display();
```

ARRAY OF DYNAMIC MEMORY ALLOCATION FOR OBJECTS AND OBJECT ARRAY

Similar to DMA of other datatypes, we can dynamically allocate memory for an object or an array of objects.

classname *pointerobject;

pointerobject = new classname;

pointerobject = new classname[size];

For deallocation of memory:

delete [ ] pointerobject;

Let us consider we have class named “college”

college * ptr;

ptr = new college; *//allocates memory dynamically for an object of class college*

ptr = new college[5]; *//allocates memory dynamically for 5 objects of class college*

delete [ ] ptr;

THIS POINTER

“this” is a C++ keyword. “this” always refers to an object that has called the member function currently. We can say that “this'' is a pointer that points to the object for which this function was called. For example, the function call A.max() will set the pointer “this” to the address of the object A.

```
#include<iostream>
using namespace std;
class test
{
        int x;
        public:
                test(int value)
                {
                        x= value;
                }
                void print();
};
```

```
void test::print()
{
        cout<<”X =”<<x<<endl;
        cout<<this<<endl;
        cout<<”(*this).x = ”<<(*this).x<<endl;
        cout<<”this -> x = ”<< this -> x<<endl;
}
int main()
{
        test t(12);
        t.print();
}
```

Output:

```
X = 12;
0x16b31728c
*this.x = 12
this->x = 12
```

STATIC DATA MEMBER AND STATIC FUNCTION

- Since it is known that every object has its own copy of data members defined by the class but that is not always true in case of static data members.
- When a static data member is defined then only such an item is created for the entire class regardless of number of objects and static data are shared by all the objects.
- These kinds of variables are declared inside class but initialized outside class.
- Static data members are useful when sharing information that is common to all objects of a class such as number of objects created etc.

STATIC DATA MEMBER AND STATIC FUNCTION

```
#include <iostream>
using namespace std;
class Counter
{
        static int c;
        public:
        void display()
        {
                c++;
                cout << "The call to display function " << c << endl;
        }
};
int Counter::c=0;
```

```
int main()
{
        counter C1, C2, C3;
        C1.display();
        C2.display();
        C3.display();
        return 0;
}
```

Output:

```
The call to display function1
The call to display function2
The call to display function3
```

Static Member Function

- A member function which is defined as a static can access only static data member and it can be invoked or called using name or object of that class.

```
#include<iostream>
using namespace std;
class Counter
{
   int a;
   static int c;
   public:
   void input()
  {
      cout<<"\n Enter the value of a:";
      cin>>a;
      cout<<"\n Enter the value of c:";
      cin>>c;
   }
 static void display()
{
      cout<<"\nThe value of c is:"<<c<<endl;
} };
int Counter::c=0;
int main() {
   Counter C1,C2,C3;
   C1.input();
   C2.input();
   C3.input();
   cout<<"\nThe value of variable c is:";
   C1.display();
   Counter::display();
   C3.display();
}
```

Output

```
Enter the value of a:2
Enter the value of c:3
Enter the value of a:4
Enter the value of c:5
Enter the value of a:5
Enter the value of c:54

The value of variable c is:

The value of c is:54
The value of c is:54
The value of c is:54
```

CONSTANT MEMBER FUNCTIONS AND CONSTANT OBJECTS

Constant Member Functions

Constant member function is a function that can’t modify the data member of a class but can use that data member. A member function is declared as a constant member function using keyword ***const***.

return_type function_name(parameters) const;

For example,
void large(int, int) const;

The qualifier ***const*** appears both in member function declaration and definitions. Once a member function is declared as const, it cannot alter the data values of the class. The compiler will generate an error message if such functions try to alter the data values.

Constant Member Functions

```
#include<iostream>
#include<cmath>
using namespace std;
class Coordinate
{
        int x;
        int y;
        public:
        void input()
        {
                cout << "Enter X and Y: ";
                cin >> x >> y;
         }
         void display() const;
};
```

```
void Coordinate::display() const
 {
      float sum;
      cout << "\nX Coordinate : " << x;
      cout << "\nY Coordinate : " << y;
      sum = sqrt(pow(x,2) + pow(y,2));
      cout << "\nMagnitude : " << sum;
      // x++ ; Error : Cannot change the value of
variable inside the constant member function
}
int main()
{
    Coordinate C1;
    C1.input();
    C1.display();
    return 0;
}
```

Note: A constant member function of a class can only invoke other constant member functions of the same class i.e. In above program, we cannot call input() from display().

CONSTANT MEMBER FUNCTIONS AND CONSTANT OBJECTS

Constant Objects

Just like constant variables, a constant object is an object of a class that cannot be modified. A constant object can call only const member functions because they are the only ones that guarantee not to modify the object.

const class_name object_name(parameter);

or,

class_name const object_name(parameter);

The member of a constant object can be initialized only by a constructor, as a part of the object creation procedure.

Constant Objects

```
#include <iostream>
using namespace std;
class Coordinate
 {
      int x;
      int y;
      public:
            Coordinate(int a, int b) {
                     x = a;
                     y = b; }
            void get_coordinate() {
                     cout << "Enter the xcoordinate:";
                     cin >> x;
                     cout << "Enter the ycoordinate:";
                     cin >> y;
             }
          void show_coordinate() const {
                     cout << "\nxcoordinate:" << x;
                     cout << "\nycoordinate:" << y;
             }
void display() {
cout << "\nxcoordinate:" << x;
cout << "\nycoordinate:" << y;
}
};
int main() {
      Coordinate c1(0,0);
      c1.get_coordinate();
      c1.show_coordinate();
      const Coordinate c2(10,20);
      //c2.get_coordinate()
          //Invalid:get_coordinate() modifies the member of Coordinate class
      c2.show_coordinate();
             //Valid: show_coordinate() is constant member function
   //c2.display();
      //Invalid: constant object can only invoke constant member function
     return 0;
}
```

CONSTANT MEMBER FUNCTIONS AND CONSTANT OBJECTS

mutable keyword

As discussed above, const objects can only invoke const member functions and this const member functions cannot change the data member defined in a class. However, a situation may arise when we want to create const object but we would like to modify a particular data item only. In such situation, we can make it possible by defining the data item as ***mutable***.

Note: the const function changing the value of mutable data cannot have statements that try to change the value of other ordinary data.

mutable keyword

```
#include <iostream>
using namespace std;
class Student
{
    char *name;
    mutable char *address;
    public:
        Student(char *n, char *ad)  {
                name = n;
                address = ad;
        }
       void change_name(char *new_name)
       {
               name = new_name;
       }
void change_address(char *new_address) const {
address = new_address;
}
void display() const {
cout << "Name:" << name << endl;
cout << "Address:" << address << endl; }
};

int main()
{
const Student s1("ABC","PlanetEarth");
//s1.change_name("XYZ");
 s1.change_address("Nepal");
s1.display();
return 0;
}
```

FRIEND FUNCTION AND FRIEND CLASSES

Friend Function

- As we know, the private members of a class can be accessed only through the public section of the same class. But, if we want to give access to the private member to the function outside the class, we can use the concept of friend function in such circumstances.
- A friend function is a function that is not a member of a class but has access to the class's private and protected members.
- Some important points about the friend functions are:
  - A friend function cannot be called using the object of the class. They are called like a normal function.
  - A friend function can access the resources of a class using the object of the same class.
  - Usually, a friend function has an object as its argument.
  - Friend declaration can be placed anywhere in the class and the access specifier does not matter.

FRIEND FUNCTION AND FRIEND CLASSES

Friend Function

Syntax:

```
class class_name
{...
 friend return_type function_name(arguments);
 };
return_type function_name(arguments)
{
 // body of the function
 }
```

Example:

```
class demo
{
   int a;
   float b;
public:
void input() {
   cout<<"Enter the value of a and b";
   cin>> a >> b ;
}
friend void output(demo);
};
void output(demo d)
{
      cout<<"The values of a and b are : "<< d.a <<
endl<< d.b;
}
int main()
{
    demo d1;
    d1.input();
    output(d1);
    return 0;
}
```

FRIEND FUNCTION AND FRIEND CLASSES

Friend Function

WAP to make global function which returns the average of 10 number list stored at a member class

```
#include <iostream>
using namespace std;
const unsigned int SIZE = 10;
class Num
{
        int n[SIZE];
        public:
              void input()
              {
                 cout << "Enter the elements:";
                 for(int i=0; i<SIZE; i++)
                 cin >> n[i];
              }
              friend float average(Num);
};
```

```
float average(Num n1)
{
    float sum=0.0;
    for(int i=0; i<SIZE; i++)
    {
             sum += n1.n[i];
     }
    return sum/SIZE;
}
```

```
int main()
{
     Num n1;
     float avg;
     n1.input();
     avg = average(n1);
     cout << avg ;
     return 0;
 }
```

FRIEND FUNCTION AND FRIEND CLASSES

Friend as a bridge function

- Friend functions can be used as a bridge between two classes.
- It can be used to access the private and protected members of two or more classes.
- This can be accomplished by declaring the same function as the friend of the classes it is required to link

Friend as a bridge function

WAP to swap the private data of two different class

```
#include <iostream>
using namespace std;
class ObjB;
class ObjA {
    int x;
    public:
        void input(){
            cout << "Enter value of x: ";
            cin >> x;
        }
        void display(){
            cout << "X : " << x << endl;
        }
    friend void swap(ObjA&, ObjB&);
};
```

```
class ObjB {
    int a;
    public:
        void input(){
            cout << "Enter value of a: ";
            cin >> a;
        }
        void display()
        { cout << "a : " << a << endl; }
friend void swap(ObjA&, ObjB&); };

void swap(ObjA& a, ObjB& b){
      int tmp = a.x;
      a.x = b.a;
      b.a = tmp;
}
```

```
main() {
ObjA a1;
ObjB b1;
a1.input();
b1.input();
swap(a1, b1);
a1.display();
b1.display();
return 0;
}
```

Output:

```
Enter value of x: 1
 Enter value of a: 2
 X : 2
 a : 1
```

FRIEND FUNCTION AND FRIEND CLASSES

Member function of a class as a friend of another

The member functions of a class can be friend functions of another class. In such case, their declaration as a friend in another class uses their qualified name (full name).

Friend function of a class as a friend of another

WAP to make a mult() function of class A as a friend of B and display the proper output

```
#include <iostream>
 using namespace std;
class Beta;
class Alpha {
int x;
public:
void input()
{
cout << "Enter the value of x :";
cin >> x ;
}
void mult(Beta);
 };
```

```
class Beta {
int y;
public:
void input(){
    cout << "Enter the value of y:";
    cin >> y;
}
friend void Alpha::mult(Beta);
};
void Alpha::mult(Beta t){
cout << x << "*" << t.y << "=" << x*t.y << endl;
}
```

```
main() {
Alpha a;
Beta b;
a.input();
 b.input();
a.mult(b);
return 0;
 }
```

Output:

```
Enter the value of x :1
Enter the value of y: 2
1*2=2
```

FRIEND FUNCTION AND FRIEND CLASSES

Friend Class

There may be a situation when all the member functions of a class have to be declared as friend of another class. In such situations, instead of making the functions friend separately, we can make the whole class a friend of another class. A friend class is a class whose member functions can access another class's private and protected members. This can be specified as follows:

```
class x;
class z
{
     ........
    friend class x;        //all member functions of class x are friends to z.
};
```

Friend Class

```
#include <iostream>
 using namespace std;
class ABC;
class XYZ {
int x;
 public:
friend class ABC;
};
```

```
class ABC {
int a;
 public:
void getdata(XYZ &o1)
{
cout << "Enter the value of a:"; cin >> a;
cout << "Enter the value of x:"; cin >> o1.x;
}
void sum(XYZ &o1)
{
cout << "The sum is:" << a+o1.x << endl;
}
void product(XYZ &o1)
{
cout << "The product is:" << a * o1.x << endl;
}
};
```

```
int main() {
ABC obj1;
XYZ obj2;
obj1.getdata(obj2);
 obj1.sum(obj2);
obj1.product(obj2);
return 0;
 }
```

Output:

```
Enter the value of a:2
Enter the value of x:1
The sum is:3
The product is:2
```

THANK YOU