**CONSTRUCTORS**

A constructor is a member function of a class that is automatically called, when an object is created of that class. It has the same name as that of the class's name and its primary job is to initialize the object to a legal initial value for the class.

Consider the following class having a constructor :

class student

{

private :

int roll no;

float marks:

public :

student ( ) // constructor

{

roll no = 0;

marks = 0.0;

}

};

Whenever an object of the student type will be created the compiler will automatically call the constructor student : student ( ) for the newly created object.

The constructors have some special characterstics. These are

1. They should be declared in the public section.
2. They are invoked automatically when the objects are created.
3. They do not have return types, not even void and therefore, and they cannot return values.
4. They cannot be inherited, though a derived class can call the base class constructor.
5. They can have default arguments.
6. Constructors cannot be virtual.
7. A constructor may not be static.
8. Member functions may be called from within a constructor.
9. A class have multiple constructors for various situations.
10. Constructor are member to declared in header file of their class, or are inherited from their base class

**DEFAULT CONSTRUCTORS**

A constructor that accepts no parameters is called the default constructor.

eg. X : : X ( ) is the default constructor for class X since it takes no parameter.

when the user-defined class does not contain any constructor, the compiler automatically supplies a default constructor, having no arguments. Consider the following code -

Class A{

public :

void get val (void);

void put val (void) :

: // member functions def.

};

A o1 ; // user default constructors for

o1.getval ( ); // creating o1

o1.putval ( ) ;

Having a default constructor simply means that an application can declare instances of the class, since C++ requires that whenever an instance is created, its constructor is called. The default constructor provided by the compiler does not do anything specific. It initializes the datamembers by any dummy value.

Once you declare a constructor with arguments, the default constructor becomes hidden. after this you cannot invoke the default constructor.

**PARAMETERIZED CONSTRUCTORS** -

A constructor may also take arguments. such constructors are called parameterized constructor. These type of constructors allows us to initialize the various data elements of different objects with different values when they are created. This is achieved by passing different values as arguments to the constructor function when the objects are created.

The initial values can be passed at the time of object creation in two manners is by calling the constructor implicity (implicit call)

- By implicit call, it means that the consturctor is called (invoked ) even when its name has not been mentioned in the statement.

For eg.,

(i)to create an object obj1 of type ABC type with initial values 13, 11.4 and 'p', the constructor will be called implicitly as follows :

ABC obj1 (13, 11.4, 'p'); \\ implict call

This method is sometimes called shorthand method.

(ii) by calling the constructor explicitly (Explicit-call) - By explicit call, it means that the name of the constructor is explicity provided to invoke it, so that the object can be initialized for eg.

ABC ob1 = ABC (13, 11.4, 'p'); \\ explict call

The explicit call to a constructor also allows you to create a temporary instance or temporary object. A temporary instance is the one that lives in the memory as long as it is being used or referred in the expression and after this it dies.

for eg.

class sample { int i};

public :

simple ( int a , int b)

{ i = a ; j = b ;}

void print (void)

{ cout <<i<<j<<”/n”;

}

void test (void)

{ sample S1 (2,5)

S1. print ( );

Sample (4,9).print ( ); // Data values of a temprary sample instance printed. It lives to memory as long as sample (4,9).

**COPY CONSTRUCTOR** :-

A copy constructor is a constructor of the form classname (classname & ).

The compiler will use the coy constructor whenever you initialize an instance using values of another instance of same type. for eg.

Sample S1 ; // default constructor used

sample S2 = S1; // copy constructor used.

In this, the compiles will copy the instance S1 to S2 member by member.

A copy constructor takes a reference to an object of the same class an argument.

eg. class code { int id ;

public :

Code c1 { } // constructor

Code (int a)

{ id = a :} constructor agaes

Code (code & X) // copy constructor }

{ id = x.id;}

void display (void)

{ cout < < d : } } ;

void main ( )

Code A(100) ; // object A is created and initialized

Code B = A ; // copy constructor called

Code C = A : // copy constructor called again

Code D ; // D is created, not initialized

D = A; // copy does not called

Cout << A.display ( );

Cout << B.display ( );

Cout << C display ( );

Cout << D display ( );

Why the argument is passed by reference to a copy constructor ?

If we try to pass argument by value to a copy constructor, the compiles complains out of memory. The reason being that when an argument is passed by value, a copy of it is constructed. To create a copy of object, who walks? The copy constructor. But the copy constructor is creating a copy of the objects for itself, thus, it calls itself. Again the called copy constructor requires another copy so again it is called. In fact, it calls itself again and again until the compiler runs out of memory. So, in the copy constructor, the arguments must be passed by reference, which creates no copies.

**Dynamic initialization through constructor**

Class objects can be initialized dynamically too. That is to say, the initial value of an object may be provided during run time. One advantage of dynamic initialization is that we can provide various initialization formats, using overloaded constructors. This provides the flexibility of using different format of data at run time depending upon the situation.

**Dynamic constructor**

The constructors can also be used to allocate memory while creating objects. This will enable the system to allocate the right amount of memory for each object when the objects are not of the same size, thus resulting in saving of memory. Allocation of memory to the objects at the time of construction is known as dynamic construction of objects.

#include <iostream>

#include <cstring>

class person

{  
public:  
 char \*name;

int age,size;

person (char \*n , int a ) //dynamic constructor

{

size = strlen(n)+1;

name = new char[size];

strcpy (name, n);

age = a;

}  
……..

};  
   
int main ()

{  
 person p;

cout << p.name << ", age " << p.age << endl << endl; // output: no name, age 0

person k ("John", 56);

cout << k.name << ", age " << k.age << endl << endl; // output: John, age 56

return 0;

}

**Static Member Data and Static Member Functions**

By declaring a **function member** as **static**, we make it independent of any particular object of the class. A **static member function** can be called even if no objects of the class exist and the **static functions** are accessed using only the class name and the scope resolution operator ::.

**POLYMORPHISM**

* Polymorphism is the ability to use an operator or method in different ways. Polymorphism gives different meanings or functions to the operators or methods. Poly, referring to many, signifies the many uses of these operators and methods. A single method usage or an operator functioning in many ways can be called polymorphism. Polymorphism refers to codes, operations or objects that behave differently in different contexts.
* Polymorphism is implemented using overloaded function & operators.
* The overloaded member function are selected for invoking by matching arguments, both type and number. This information to the compiler at the compile time. Therefore compiler selects the appropriate function for a particular call the compile time itself. This is called early binding static binding or static linking.

Polymorphism is a powerful feature of the object oriented programming language C++. A single operator + behaves differently in different contexts such as integer, float or strings referring the concept of polymorphism. The above concept leads to operator overloading. The concept of overloading is also a branch of polymorphism.

Polymorphism refers to the ability to call different functions by using only one type of function call. This approach leads to different functions executed by the same function call. Polymorphism is used to give different meanings to the same concept.

### Features and Advantages of the concept of Polymorphism:

### Applications are easily extendable:

* Helps in reusability of code.
* Provides easier maintenance of applications.
* Helps in achieving robustness in applications.

