}

★ <u>Constructor</u> :- It is the member function of the class called automatically when one object is created of that class. Constructor has the same name as that of class name and does not return any type. Also the constructor is always public. If we do not specify a constructor C++ compiler generate a default constructor for as accept no parameters and has an empty body.

(10) Program to call member functions using object of that class-

1.
```
class comment
{
public:
void show()
{
cout <<" I am JAVA learner";
}
};
~~int~~ main()
{
comment yes;          Output :-
yes. show;                I am Java learner.
getch();
return 0;
}
```

(ii) 2. Example of constructor -

```
class comment
{
public:
comment() → function member
                constructor
{
cout<<" I love programming";
}
void show ()
{
cout <<" I am Java learner";
}
};
int main()          Output:-
{                        I love programming.
comment yes;             I am Java learner.
yes. show ();
getch();
return 0;
}
```

There are three types of constructor -

1- Default constructor

2- Parameterised constructor
3- Copy constructor

1- Default constructor - A constructor without ~~a~~ any argument with default value for every argument is said to be default constructor.
What is significance of default constructor? Will the code be generated for every default constructor? Will there be any code inserted by computer to the user implemented default constructor behind the scenes.
The compiler will implicitly declare default conitructor. If not provided by programmer will define it when is need. Compiler define default constructor is required to do certain initialization of class intervals. It will not touch the data members or plain old data type. Consider a class derived from another class will default constructor or a class containing another class object with default constructor. The compiler need to insert code to call the default constructor of base class/ object.

Example of default constructor -

Program.9.

```
class student → class name
{
private:
int sid;      } data members declaration
float marks;
public:
student(); → function declaration
void showdetails() → simple function
                      definition
{
cout <<"Student ID:" << sid <<"Marks"<<
                                   marks;
}};        ↗ scope operator
student :: student() → constructor function
                        definition
{
sid = 2;        > Data member
marks = 80.0;     value initialization
}
int main()
{
clrscr();
student s1; → class Object
s1.showdetails(); → function calling
getch();
return 0;
}
```

Output :- Student ID : 2 Marks : 80

2- **Parameterized constructor** - A default constructor does not have any parameter. But if you need a constructor can have parameter, this helps you to assign initial value to an object at the time of its creation. These are the constructors with parameter. Using this constructor you can provide different value to data members of different objects by passing the appropriate values as an argument.

(12) **Example of parameterized constructor** -

```
#include <iostream.h>
#include <conio.h>
class student
{
private:
int sid;
float marks;
public:
student (int i, float j)
{
sid = i;
marks = j;
}
void showdetails()
{
cout<<"student ID:"<<sid<<endl<<"marks:"
                                    <<marks;
```

Pradeep
Page No.
Date: / /20

```
}};
void main()
{
student s1(3,700);
s1. showdetails( );
getch();
}
```

```
student ID: 3
marks : 700
```

3- Copy constructor - It is a constructor of the form class name (class name &). Compiler will use the copy constructor whenever you initialize an instance using values of another instance of same type. A copy constructor takes a reference to an object of the same class as an argument.

Example-

③
```
# include < iostream.h>
# include < conio.h>
class student
{
int roll;
int marks;
public :
student (int m, int n) → function
                              definition
{
roll = m;
marks = n;
}
student (student &t);
void showdetails ()
{
cout<<"\n Roll no:"<< roll;
cout<<"\n marks:"<< marks;
}};
student :: student (student &t)
{
roll = t. roll;
marks = t' marks;
}
int main()
{
cout<<"\n Parameterized constructor output r";
student r. (60,130);
r. showdetails ();
cout<<"\n Copy constructor output
student stu(r);
stu . showdetails ();

}
```

* **Destructor** :- It is used to destroy the objects that have been created by constructor. A destructor destroy value of the object being destroyed. Destructor is called automatically by compiler. Destructor clean up storage that is no longer accessible.

Characteristics of destructor -

1- We can have only one constructor for a class. Destructor cannot be overloaded.
2- No argument can be provided to a destructor, neither does it return any value.
3- Destructor cannot be inherit.
4- A destructor may not be static.

Limitations of destructor -

1- These are case sensitive.
2- They are not good for program having thousands lines of code.

Example :-

(14)
```
#include <iostream.h>
#include <conio.h>
class student
{
private:
```

```
int sid;
float marks;
public:
student (int i, float j)
{
sid = i;
marks = j;
}
void showdetails()
{
cout<<"Student ID:"<<sid<<"Marks:"<<marks;
}
~student()
{
cout<<"Destructor is invoked";
}};
int main()
{
student s1(5,50.5);
s1.showdetails();
getch();
return 0;
}
```

Output:-

```
        Student ID: 5
        Marks: 50.5
        Destructor is invoked
```

* <u>Garbage collection</u> :- In C++ programming, garbage collection is a form of automatic memory management. The garbage collector or junk collector attempt to reclaim garbage or memory occupied by objects that are no longer in use by the program.
Garbage collection is essentially the opposite of manually memory management which requires the programmer to specify which object to deallocate and return to the memory system.
Ex: Destructor

* What is memory leak?
In C++, memory allocation that takes place at run time must be managed by the executing program itself. Typically run time memory is allocated on stack section for function and on heap section. There is no need to manage stack section, memory is allocated for the function call and is deallocated when the function returns.
Secondly the heap section memory is allocated using new operator and is release using delete operator.
Memory leak is a situation when executing progress is allocated memory from heap section but not release in stack.

Example of garbage collection -

```
#include <iostream.h>
#include <conio.h>
class mysample
{
public:
mysample()
{
cout<<" In constructor ___ "<< endl;
}
~mysample()
{
cout << " In destructor ___ "<< endl;
}
};
void main()
{
mysample obj;
mysample *p = new mysample();
getch();
}
```

Output :-
```
In constructor ___
In destructor ___
```

* Dynamic memory allocation :- Dynamic memory allocation allow to set array size dynamically during run time rather than at compile time.

This helps when the program does now know in advance about the number of items (variables-value to be stored). Dynamic memory allocation in C/C++ refer to performing memory allocation manually by programmer. Dynamically allocated memory is allocated on heap and non-static and local variables get memory allocated on stack details.

How it is different from memory allocated to normal variables?
For normal variable like int a, char, string, str[10] etc, memory is automatically and deallocated. For dynamically allocated memory like -

```
int *p = new int();
```

it is programmer's responsibility to deallocate memory, when no longer needed. It causes memory leak (memory is not deallocated until program terminate).

(16) Example of dynamic memory allocation-

```
#include <iostream.h>
#include <conio.h>
class cube
{
public:
cube()
{
cout<<"Constructor is called!"<<endl;
}
~cube()
{
cout<<"Destructor is called!"<<endl;
}
};
int main()
{
clrscr();
cube * mycubearray = new cube[3];
delete[] mycubearray;
getch();
return 0;
}
```

Output :-
```
Constructor is called!
Destructor is called!
- - - - ——— 3
```

Note:- The new operator return a pointer to the object is allocated, the program must define a pointer with suitable scope to access those objects.

* **Abstract class :-** An abstract class is a class that is design to be specifically use as a base class. An abstract class contains at least one pure virtual function. You can declare a pure virtual function by using a pure specifier in the declaration of a virtual member function in the class declaration. Sometime implementation of all functions cannot be provided in a base class because we do not know the implementation such as a class is called abstract class.
An abstract class can have constructor. If we do not override the pure virtual function in derived class then derived class also become abstract class.
We can have pointers and refrence of abstract class type. A class is abstract if it has at least one pure virtual function.

(17.)
```
#include <iostream.h>
#include <conio.h>
class base
{
public:
virtual void disp()=0;
};
class d : public base
```

```
{
public:
void disp()
{
cout<<"Derived class";
}
};
int main()
{
d obj;
obj. disp()
getch();
return 0;
}
```

Output :- Derived class