}

UNIT -IV Object Oriented Programming with C++

CHAPTER 15

Polymorphism

1.What is Function overloading?

- The ability of the function to process the **message or data** in **more than one form** is called as function overloading.

Ex. **float area (float r);**
float area (float l,float b);

2.What is function's signature?

- The **number and types** of a function's parameters are called the **function's signature**.

3.Define Overloaded resolution.

- The process of **selecting the most appropriate** overloaded **function** or operator is called overload **resolution**.

4.What are the advantages of function overloading? or What is the use of overloading a function?

- Function overloading is used to **reduces** the number of comparisons in a program
- It makes the program to **execute faster**.
- It also helps the programmer by reducing the number of function names to be **remembered.**

5.Explain Function over loading with an example.

- The ability of the function to **process** the **message** or **data in more than one form** is called as function overloading.

Ex. **float area (float r);**
float area (float l,float b);

Rules for function overloading

- The overloaded function must **differ** in the **number** of its arguments or **data types**.
- The **return type** of overloaded functions are **not considered** for overloading same data type
- The **default arguments** of overloaded functions are **not considered** as part of the in function overloading parameter list.

Example:

```
#include <iostream>
using namespace std;
        float area ( float r )
        { return ( 22/7 * r * r );}
        float area ( float l, float b )
        { return ( l *b ) ;}
        void main()
        { cout<<"circle"<<area(5.2);
        cout<<"Rectangle"<<area(5.3,8.2);
```

6.Does the return type of a function help in overloading a function?

No, The **return type** of overloaded functions are **not considered** for overloading same data type

7.Define Constructor overloading.

- Function overloading can be applied for constructors, called as Constructor overloading .
- A class can have more than one constructor with different signature.
- Constructor overloading provides flexibility of creating multiple type of objects for a class.

8.*class add{int x; public: add(int)};* Write an outline definition for the constructor.

```
        add ::add(int y)
        {
        y=x;
        }
```

9.How does a compiler decide as to which function should be invoked when there are many functions? Give an example.

When you call an overloaded function,

- The compiler determines the most appropriate definition to use,
- by **comparing** the **number** of argument and their **types** to **call the function** definitions.
- The process of selecting the **most appropriate overloaded function** or operator is called overload **resolution.**

```
        area(float r)
                {
                cout<<3.14*r*r;
                }
        area( float l,float b)
                {
                cout<<l*b;
                }
        void main()
                {
                area(4.6);
                area(5.7,4.3);
                }
```

10.class sale (int cost, discount ;public: sale(sale &); Write a non inline definition for constructor specified;

```
sale :: sale(sale &a)
{
cost=s.cost;
discount=s.discount;
}
```

11.Define Operator overloading

- The **mechanism** of giving **special meaning** to an **operator** is known as operator overloading.
- Operator overloading provides **new definitions** for most of the C++ **operators**

12.List out the operators that cannot be overload in C++

- scope operator (::)
- sizeof
- member selector (.)
- member pointer selector (*)
- ternary operator (?:)

13.How to define operator overload in C++

- The definition of the overloaded operator is given using the keyword '**operator**' followed by an **operator** symbol.

Syntax:

Inline:

```
ReturnType   operator  operatorSymbol(argument)
{
}
Ex. complex operator +( complex c2)
        {
        }
```

Outline:

```
ReturnType   classname : : operator  operatorSymbol(argument)
{
}
```

14. What are the Rules or Restrictions on Operator Overloading?

- **Precedence and Associativity** of an operator **cannot** be changed.
- **No new operators** can be created,
- Only **existing** operators can be overloaded.
- **Cannot redefine** the meaning of an operator's procedure.
- Overloaded operators **cannot have default** arguments.
- When binary operators are overloaded, the **left hand** object must be an **object** of the relevant **class**

15.Define and explain the operator overloading?

Define :

- The **mechanism** of giving **special meaning** to an **operator** is known as operator overloading.
- Operator overloading provides **new definitions** for most of the C++ **operators**

The definition of the overloaded operator is given using the keyword '**operator**' followed by an **operator** symbol.

Syntax:

Inline:

```
ReturnType   operator  operatorSymbol(argument)
{
}
Ex. complex operator +( complex c2)
        {
        }
```

Outline:

```
ReturnType   classname : : operator  operatorSymbol(argument)
{
}
```

Rules or Restrictions on Operator Overloading

- **Precedence and Associativity** of an operator **cannot** be changed.
- **No new operators** can be created,
- Only **existing** operators can be overloaded.
- **Cannot redefine** the meaning of an operator's procedure.
- Overloaded operators **cannot have default** arguments.
- When binary operators are overloaded, the **left hand** object must be an **object** of the relevant **class**