Expt No

Assignment - 2

Page No.
Date.

(1) local variables are stored in an area called
(a) Heap (c) Free memory
(b) Permanent storage area (d) Stack

(2)
```
#include Using namespace std;
class Base {};
class Derived : public Base{};
int main()
{
    Base * bP = new Derived;
    Derived *dP = new Base;
}
```
(a) No Compiler Error
(b) Compiler Error in line "Base *bP = new Derived;"
(c) Compiler Error in line "Derived *dP = new Base;"
(d) Runtime Error

(3) When the inheritance is private, the private methods in base class are _____ in the derived class (in C++)
(a) Inaccessible (c) Protected
(b) Accessible (d) Public

(4) which of the following is true ?
(a) The number of times destructor is called depends on number of Objects created. True.
(b) Destructor is called only once.

Teacher's Signature .....

(c) There can be more than one destructors in the class.

(d) Programmer have to always call destructor at the end of program.

(5) State true or false -

Type conversion is automatic whereas type casting is explicit

A. True

B. False.

Short answer type question

1. Explain about new and delete keywords with code.

Ans> The new operator denotes a request for memory allocation on the free store. If sufficient memory is available, new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

The syntax to allocate memory of any data type is

```
pointer-variable = new datatype;
```

Since memory allocate has to be deallocated dynamically, we use delete operate by C++ to do this.

Syntax :-

```
delete pointer-variable;
```

Delete is an operator that is used to destroy array and non-array (pointer) objects which are created by new expression.

An example for this is following -

```
#include <iostream>
using namespace std;
int main()
{
    int *p = NULL;
    p = new (nothrow) int;
    if (!p)
    {
        *p = 29;
        cout << " value of p" << *p << end;
    }
    float *r = new float (75.25);
    cout << " value of r" << *r << endl;
    int n = 5;
    int *q = new (nothrow) int(n);
    if (!q)
    cout << " memory allocation failed !!" << endl;
    else
    {
        for (int i=0 ; i<n ; i++)
            q[i] = i+1;
        cout << " value store in block of memory ";
        for ( int i= 0; i<n; i++ )
            cout << q[i] << " ";
    }
    delete p;
    delete r;
    delete [ ]q;
    return 0;
}
```

Expt. No.

Page No.
Date

2. What are constructors? why they are required? Explain different types of constructors with suitable example.

Ans → A constructor is a member function of a class which initializes objects of a class. It is a special member function of a class.

> Constructor has same name as the class itself
> It doesn't have any return type.
> It is called automatically when the object of the class is created.
> If not specified c++ compiler generates a default constructor.

The main purpose of the constructor in c++ programming is to construct an object of class ie it is used to initialize all class data members.

Types of constructors are – 1. Default constructor
2. parameterized constructor
3. copy constructor.

1. Default constructor :- [classname()] Default constructor is a constructor which doesn't take any argument. It has no parameter.

Teacher's Signature .....

①

②

```
#include<iostream>
using namespace std;
class Default constructor
{
 public:
   int a, b;
 Construct ( )   // default constructor
  {
    a = 10;
    b = 20;
  }
};
int main ( )
{
  Default Constructor c;  // constructor called automatically on creating
  cout << "a: " << c.a << endl;        object.
  cout << "b: " << c.b << endl;
}
```

o/p

a = 10

b = 20

2. Parameterized Constructors:- [class_name (parameters)] It is possible to pass arguments to constructors. these arguments help to initialize an object when it is created.

// CPP program to explain parameterized constructor.

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

```
class Point
{
  private:
     int x, y;
  public:
     Point(int x1, int y1)  // parameterized constructor.
     {
        x = x1;
        y = y1;
     }
   int getx( )
     {
       return x;
     }
    int gety( )
     {
       return y;
     }
};
int main( )
{
   Point P1(10, 15);  // constructor called.
   cout << "P1.x = " << P1.getx( ) << "P1.y = " << P1.gety( );

 return 0;
}
```

Page No.
Date.

Q3 Copy constructor :- [class-name (const class-name old-object)] A copy constructor is a member function which initializes an object using another object of same class.

```
#include <iostream>
using namespace std;
class point
{
 private:
    double x, y;
 public:
  point (double px, double py)
   {
        x = px, y = py;
    }
};
 int main (void) {
  point a[10];          // line causes error
  point b = point (5, 6);
}
```

Q3 Explain the difference b/w object oriented and procedural programming language in details.

Teacher's Signature .....

Expt No.

Page No.
Date

Object Oriented programming	Procedural programming
In OOP, program is divided into small parts called objects	In procedural programming, program is divided into small parts called function
Object oriented programming follows bottom up approach.	Procedural programming follows top down approach.
They have access specifiers like private, public, protected etc.	There is no access specifiers in procedural programming.
Adding new data and function is easy	Adding new data on function is not easy.
It provides data hiding so it is more secure.	It doesn't have any proper way for hiding data, so it is less secure.
Overloading is possible.	Overloading is not possible.
Data is more important than function.	Function is more important than data.
Object oriented programming is based on real world.	Procedural programming is based on unreal world.
Examples:- c++, Java, python, C# etc.	Examples:- C, FORTRAN, Pascal, Basic etc.

Long Question Answers-

Q 1 Explain type of polymorphism with code-

Polymorphism in c++ means, the same entity function or object behaves differently in different scenarios.

Teacher's Signature

The different types of polymorphisms are -

(1) · Compile time polymorphism – function overloading
– operator overloading

(2) Runtime polymorphism – virtual function.

(1) Compile time polymorphism - A function is called at the time of program compilation.

(a) Function overloading - Function overloading means one function can perform many tasks. In C++, a single function is used to perform many tasks with same name and different types of arguments.

Example -

```
#include <iostream>
Using namespace std;
class Addition{
  public:
   int ADD (int x, int y) // function with parameter.
      { return x+y;  // fun. performing addition of INT
      }
   int ADD ( ){      // fun. with same name but without parameter.
     String a = "Hello";
     string b = "SAM";
     String c = a+b;      // concatenation of string is done.
     Cout << c << endl;
   }
};
```

```
int main ( void ){
    Addetion obj;   // Object is created
    Cout << obj. ADD (123, 10) << endl;  // 1st function called
     Obj. ADD ( );  // 2nd function called.
     return 0;
}
```

(b) Operator overloading - Operator Overloading means defining additional tasks to operators without changing its actual meaning.

> The purpose of operator overloading is to provide a special meaning to the user-defined data type.

> The advantage of it is to perform different operations on the same operand.

Example -

```
#include <iostream>
using namespace std;
class A
{ string x;
  public:
   A ( ){

   A(string i)
    {
      x= i;
    }
   void operator + (A);
    void display ( );
};
void A:: operator + ( A a)
 {
```

```
    String m = x + a.x;
    Cout<< "The result of the addition of two objects is" << m;
}

int main( )
{
    A a1 ("Programming");
    A a2 (" in cpp");
     a1 + a2;
    return 0;
}
```

Output:

The result of addition of two object is programming in cpp

(2) Runtime Polymorphism. - In a Runtime polymorphism, functions are called at time ~~of time~~ the program execution.

> It is also known as late binding or dynamic binding.

(a) Function overriding - In these, we give new definition to base class function in the derived class. ie the base class functions are overridden.

> In function overriding we have 2 definition of the same function one in the superclass and one in derived class.

Example -

```
#include <iostream>
using namespace std;
class Animal {
  public :
     void function( )
       {
```

```
    cout << "Eating" << endl;
  }
};
class Man : Public Animal
 {
   public
     void function ( )
      {
        cout << " walking" << endl;
      }
   };
int main ( )
{
  Animal A = Animal ( );
  A. function ( );        // parent class calling.

  Man m = Man ( );
   m. function ( );  // Derived class calling.

  return 0;
}
```

Output -
Eating
walking.

(b) Virtual Function - A virtual function is declared by keyword virtual. The return type of virtual function may be int, void, float.

> It is a member function in the base class and can be redefined in a derived class.

> It helps to tell compiler to perform dynamic / late binding on the function.

```
# include <iostream>
using namespace std;
class Add
{
  public:
     virtual void print( )
     {
       int a = 10, b = 20;
       cout << "base class in action" << a+b << endl;
     }
    void show( )
     {
      cout << " show base class" << endl;
     }
};
class sub: public Add( )
 {
  public:
    void print( )
    { int x = 40, y = 10;
    cout << "derived class in action" << x+y << endl;
    }
     void show( )
      { cout << " show derived class" << endl;
      }
};
int main( )
  Add * aptr;
   Sub s;
   aptr = &s;
Aptr -> print( ); // Virtual function binded at runtime.
aptr -> show( ); // non-virtual function binded at compile time.
  return 0;
}
```

O/P

derived class in action 30

Show base class

```
#include <bits/stdc++.h>
using namespace std;
void sort(int a[], int arr_size)
{
    int lo = 0;
    int hi = arr_size - 1;
    int mid = 0;
  while (mid <= hi)
  {
    switch (a[mid])
    {
      case 0:
      swap(a[lo++], a[mid++]);
      break;
    case 1:
      mid ++;
    break;
   case 2:
   swap(a[mid], a[hi--]);
   break;
  }
 }
}
void printArray(int arr[], int arr_size)
{
  for(int i=0, i< arr_size; i++)
     cout << arr[i] << "  ";
}
 int main()
{
 int arr[] = {0,1,1,0,1,2,1,2,0,0,0,1};
  int n = sizeof(arr)/sizeof(arr[0]);
    sort(arr, n);
  cout << "array after sorting";
  printArray(arr, n);
  return 0;
}
```

③

```
#include <iostream>
#include <string>
using namespace std;
class member{
 char name[20], address[40];
 double number;
 int age;
public:
int salary;
 void input(){
cout<<"name :"<<endl;
 cin.getline(name,20);
cout<<" Age :"<<endl;
 cin>>age;
 cout<<" phone number: "<<endl;
 cin>>number;
cout<<" Address : "<<endl;
 cin.getline(address,40);
 cout<<" salary : "<<endl;
 cin >> salary;
}
void display(){
 cout<<" Name : "<<name<<endl;
 cout<<" Age : "<<age<<endl;
 cout<<" phone number : "<<number;
cout<<" Address:"<<address<<endl;
cout<<" salary :"<<salary<<endl;
}
};
class employee : public member {
char specialization[20], department[20];
public:
    void input()
{
cout<<" enter employee details";
 member::input();
cout<<"specialization:"<<endl;
 cin.getline(specialization,20);
cout<<" Department :"<<endl;
 cin.getline(department,20);
}
 void display()
{
cout<<"Displaying employee details";
 member::display();
cout<<"specialization"<<specialization
        <<endl;
cout<<"Department"<<department<<endl;
}
 void printsalary()
{
  cout<<salary of the member is<<
   salary<<endl;
}
};
class manager : public member {
char specialization[20], department[20];
public:
   void input()
 {
cout<<Enter manager details'<<endl;
 public:
```

```
void input();
cout << "specialization" << endl;
cin.getline(specialization, 20);
cout << "Department" << endl;
cin.getline(department, 20);
}
void display()
{
cout << "Displaying manager details";
member :: display();
cout << "specialization" << specialization
     << endl;
cout << "Department" << department
     << endl;
}
void printsalary()
{
cout << "salary of members" <<
     salary << endl;
}
};
int main()
{
employee e;
manager m;
e.input();
m.input();
e.display();
e.printsalary();
m.display();
m.printsalary();
}
```