Assignment-2

1. D
2. C
3. A
4. A
5. A

Short Answer type Questions:

1. The new Operator: The new operator requests for the memory allocation in heap. If the sufficient memory is available, it initializes the memory to the pointer variable and returns its address.

   Here is the syntax of new operator in C++ language,

   → pointer_Variable = new datatype;

   Here is the syntax to initialize the memory,

   → pointer_Variable = new datatype(value);

   Here is the syntax to allocate a block of memory,

   → pointer_Variable = new datatype [size];

Example: #include <iostream>

```
#include<iostream>
Using namespace std;
int main() {
    int *ptr1 = NULL;
    ptr1 = new int;
    float * ptr2 = new float (223.324);
    int *ptr3 = new int [28];
    * ptr1 = 28;
    cout << " Value of pointer Variable1 :" <<
* ptr1 << end1;
    cout <<" Value of pointer Variable2 :" <<
* ptr2 << end1;
    if (!ptr3)
    cout << "Allocation of memory fail
    else {
      for (int i=10; i <15; i++)
      ptr3[i] =i+1;
      cout << "Value of store in block of
memory:";
      for (int i=10; i<15, i++)
      cout << ptr3[i] <<" ";
    }
    return 0;
}
```

Output:

Value of pointer Variable 1 : 28

Value of pointer Variable 2 : 223.324

Value Of Store in block of memory : 11 12 13 14 15

The delete Operator:

The delete operator is used to deallocate the memory. User has privilege to deallocate the created pointer Variable by this delete Operator.

Here is the Syntax of ~~dete~~ delete Operator in c++ language :

→ delete pointer_Variable;

Here is the syntax to delete the block of allocated memory,

→ delete [] pointer_Variable;

Example:

```
#include <iostream>
using namespace std;
int main() {
    int * ptr 1 = NULL;
    ptr 1 = new int;
    float * ptr 2 = new float (299.121);
    int * ptr 3 = new int [28];
    * ptr 1 = 28;
```

```
cout<<" Value of pointer Variable 1 :" <<*ptr1<<endl;
cout <<"Value of pointer Variable2 :" <<*ptr2<<endl;
if (!ptr3)
 cout << "Allocation of memory failed\n";
else {
    for (int i=10; i<15; i++)
    ptr3[i] = i+1;
     cout <<" Value of store in block of
memory:";
     for(int i=10; i<15; i++)
      cout <<ptr3[i] <<" ";

 }
delete ptr1;
delete ptr2;
delete [] ptr3;
return 0;
```

Output:

Value of pointer variable 1 : 28

Value of pointer variable 2 : 299.121

Value of store in block of memory : 11 12 13 14 15

2. Constructors are special class function which performs initialization of every object. The compiler calls the constructor whenever an object is created. Constructors initialize values to object members after storage is allocated to the object.

while defining a constructor you must remember that the name of constructor will be same as the name of the class, and constructors will never have a return type.

Constructors can be defined either inside the class defination or outside class defination using class name and scope resolution :: operator.

Types of constructors in C++

1. Default constructor
2. parametrized constructor
3. Copy constructor

1. default constructor: default constructor is the constructor which doesn't take any argument. It has no parameter

example :

```
class cube
{
    public :
    int side ;
    cube ()
    {
        side = 10 ;
    }
};
int main ()
{
    cube c;
    cout << c. side;
}
```

Output :

10

2. parameterized constructors: These are the constructors with parameter. Using this constructor you can provide different values to data members of different objects, by passing the appropriate values as argument

:cout

Example:

```
class cube
{
    public:
    int side;
    cube(int x)
    {
        side=x;
    }
};
int main()
{
    cube c1(10);
    cube c2(20);
    cube c3(30);
    cout << c1.side;
    cout << c2.side;
    cout << c3.side;
}
```

Output:

10
20
30

3. Copy constructors: These are special type of constructors which takes an object as argument, and is used to copy values of data members of one object into other object.

example:

```
#include <iostream>
using namespace std;
class Demo {
    private:
    int num1, num2;
    public:
    Demo (int n1, int n2) {
        num1 = n1;
        num2 = n2;
    }
    Demo (const Demo &n) {
        num1 = n.num1;
        num2 = n.num2;
    }
    void display () {
        cout << "num1 = " << num1 << endl;
        cout << "num2 = " << num2 << endl;
    }
};
int main () {
    Demo Obj1 (10, 20);
    Demo Obj2 = Obj1;
    Obj1.display();
    Obj2.display();
    return 0;
}
```

Output:

num 1 = 10
num 2 = 20

num 1 = 10
num 2 = 20

3) procedural Oriented programming:

1. In procedural programming, program is divided into small parts called functions
2. procedural programming follows top down approach.
3. There is no access specifier in procedural programming.
4. Adding new data and function is not easy
5. Procedural programing does not have any proper way for hiding data so it is less secure
6. In procedural programming, function is more important than data
7. Procedural programming is based on unreal world.

examples: C, fortan, pascal, Basic etc

Object oriented programming:

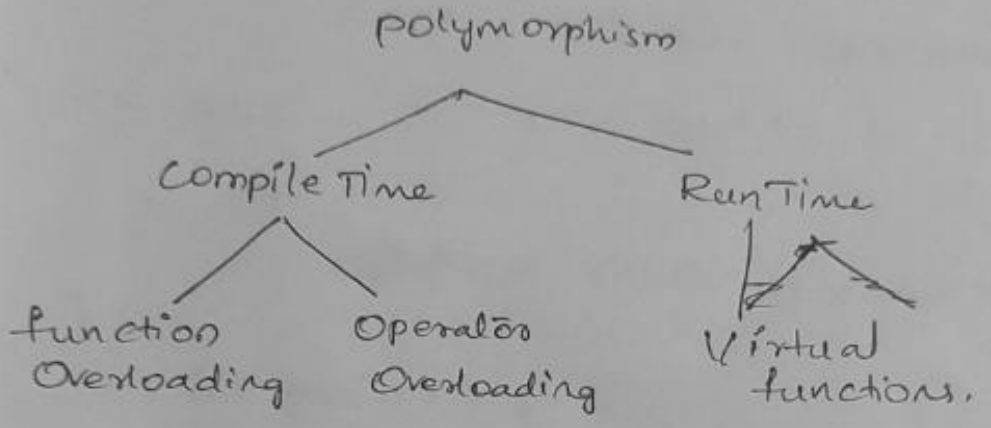
1. In object oriented programming, program is divided into small parts called Objects.
2. Object oriented programming follows bottom up approach.
3. Object oriented programming have access specifiers like private, public, protected etc.
4. Adding new data and function is easy.
5. Object Oriented programming provides data hiding so it is more secure.
6. Overloading is possible in object oriented programming.
7. In Object Oriented programing, data is more imp than function.
8. Object oriented programming is based on real world.

Examples: C++, Java, python, C# etc.

long answer type Question

A) In c++ polymorphism is mainly divided into two types:

i) Compile time polymorphism
ii) Runtime polymorphism.



1. Compile time polymorphism: This type of polymorphism is achieved by function Overloading or Operator Overloading.

→ Function Overloading: When there are multiple functions with same name but different parameters then these functions are said to be Overloaded. Functions can be Overloaded by change in no. of arguments or/and change in type of arguments

Eg:
```
#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:
    void func(int x)
    {
        cout << " value of x is " << x << endl;
    }
```

```
Void func (double x)
{
    cout << "Value of x is" << x << endl;
}
 Void func (int x, int y)
 {
    Cout << "Value of x and y is
" << x << ", " << y << endl;
    }
    Cout << "Value of x and y is"
};
int main () {
    Obj 1;
    Obj 1.func (7);
    Obj 1.func (9.132);
    Obj 1.func (85, 64);
     return 0;
```

Output:

```
Value of x is 7
Value of x is 9.132
Value of x and y is 85, 64
```

Operator Overloading: C++ also provide Option to Overload Operators. For example, we can change the Operator ('+') for string class to concatenate two strings.

Example:

```
# include <iostream>
Using namespace std;
Class Complex {
private:
    int real, imag;
public:
    Complex (int r=0, int i=0)
{real = r; imag = i;}
    Complex operator + (Complex Const &obj){
      Complex res;
      res.real = real + obj.real;
      res.imag = imag + obj.imag;
      return res;
    }
    Void print () {cout << real << "+i"
<< imag << end l; }
};
int main ()
{
    Complex C1 (10,5), C2 (2,4);
     Complex C3 = C1+C2;
       C3 . print ();
}
```

Output

12 + i9.

2. Runtime polymorphism: This type of polymorphism is achieved by function Overriding.

→ Function overriding: On the other hand occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be Overriden.

```
#include <bits/stdc++.h>
using namespace std;
class base
{
 public;
     virtual void print ()
      { cout << "print base class" <<endl;}
      void show ()
      { cout << "show base class" <<endl;}
};
class derived : public base
{
 public:
      void print ()
      { cout << "print derived class" << endl;}
      void show ()
      { cout << "show derived class" <<endl;}
};
```

```
int main()
{
    base * bptr;
    derived d;
    bptr = &d;
    bptr -> print();
    bptr -> show();
    return 0;
}
```

Output:

print derived class
show base class.