Choose the Correct Options

1) delete
3) stack
4) ~~There~~ Compiler Error in Line "Base* bp = new Derived"
5) Inaccessible
6) Programer have to always call destructor at the end of program."
7) True
8) Size of

1) New Operator: 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 general syntax of the new operator:-

```
pointer_variable_of_data_type = new data type;
```

* This data type can be a built in data type or any user-defined data type including classes and structures.

Example:-

```
int *ptr = NULL;
ptr = new int(1);
```

* In the above example, we have declared a pointer variable 'ptr' to integer and initialized it to null. Then using the 'new' operator we allocate memory to the 'ptr' variable. If memory is available on the heap, the second statement will be successful. If no memory is available then the new operator throws "std:: bad alloc" exception.

We can initialize variables using ~~new~~ new operator:-

```
ptr = new int(10);
```

* The pointer variable 'ptr' is the allocated memory using the new operator and at the same time, the assigned value is 10.

New Operator with arrays:-

The new operator is also used in allocating memory for arrays. Here we specify the number of elements to be allocated for the array.

Example:-

```
int *myarray = NULL;
myarray = new int[10];
```

* Here, new operator allocates 10 continuous elements of type integer to the pointer variable myarray and returns the pointer to the first element of myarray.

Delete Operator:- The memory allocated dynamically using the new operator has to be freed explicitly by the programmer. For this purpose we need delete operator.

General syntax of the delete operator:-

```
delete pointer_variable;
```

* We can free the memory allocated to ptr variable

```
delete ptr;
```

* This frees the memory allocated to the variable 'ptr' back to the memory pool.
* We can also use the delete operator to free the memory allocated to arrays.

```
delete [] myarray;
```

* Note the subscript operator used with the delete operator. This is because, as we have allocated the array of elements, we need to free all the locations.
* But Instead, if we used the statement `delete myarray;`
* As we know that myarray points to the first element in the array, so the above statement will only delete the first element of the array. Using subscript "[]", indicates that the variable whose memory is being freed is an array and all the memory allocated is to be freed.

2) Constructor:- A constructor is a special type of a function with no return type. Name of constructor should be same as the name of the class. We define a method inside the class and conductor is also defined inside a class. A constructor is called automatically when we can't call a constructor explicity.

* Constructor does not return any value
* Constructor should ~~not~~ have a public access modifier.
* Name of constructor should be same as the ~~me~~ name of class.
* Constructor is called automatically when we create an object of the class.

Types of Constructors:-

1) Default
2) Parameterized
3) Copy
4) Static
5) Private

Default Constructor:- Default constructor does not take any parameter

* Default constructor does not take any parameter
* Default constructor is called when we create an object of class.
* Default constructor is created by the compiler, if we donot create any constructor inside the class.

Example:-

```
Public class Adminclass
    {  string userId = string.Empty;
       string password = string.Empty;
     Public Adminclass()
       {  UserId = "Shr";
          password = "1234";
       }
    }
```

Parameterized Constructor:-

* Parameterized constructor is created by the developer, a compile does not create any parameterized constructor.
* Parameterized constructor takes at least one parameter
* Parameterized constructor is called when we create an object of class.

Example:-

```
Public class Admin class
{ string userId = String.Empty;
  String password = string.Empty;
public Admin class Cstring username, string userpassword)
{
  userId = username;
  password = userPassword;
}
}
```

Copy Constructor:-

* A copy constructor is a member function which initializes an object using another object of the same class.
* Whenever we define one or more non default constructors for a class, a default constructor should also be explicitly defined as the compiler will not provide a default constructor in this case

Example:-

```
# 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];
  Point b = point (5, 6);
}
```

Private Constructor:- A private constructor is a special instance constructor.
* It is generally used in classes that contain static members only.
* If a class has one or more private constructors and no public constructors, other classes cannot create instances of this class.

Example:-

```
Public class Counter
{
    Private Counter () { }
    Public static int currentCount;
    public static int IncrementCount ()
    {
        return ++currentCount;
    }
}
class TestCounter
{
    static void Main ()
    {
        Counter.currentCount = 100;
        Counter.IncrementCount ();
        Console.WriteLine ("New Count: {0}", Counter.current
    Console.WriteLine ("Press any key to exit."); Count);
        Console.ReadKey ();
    }
}
```

3) Procedural Oriented Programming :-

* In procedural programming, program is divided into small parts called functions
* Procedural programming follows top down approach
* There is no access specifier in procedural programming.
* Procedural programming does not have any proper way for hiding data so it is less secure.
* In procedural programming overloading is not possible
* Function is more important than data.
* It is based on unreal world
* Examples: C, FORTRAN, Pascal, Basic etc.,

Object Oriented Programming:-

* Program is divided into small parts called objects
* Follows bottom up approach
* have access specifiers like private, public, protected etc.,
* Adding new data and fuction is easy
* Data hiding so it is more secure.
* Overloading is possible
* data is more important than function
* Based on real world
* Examples:- C++, Java, Python, C#, etc.,

Long Answer Questions:-

A) **Polymorphism**:- Polymorphism means having many forms. We can define polymorphism as the ability of a message to be displayed in more than one form. Polymorphism is considered as one of the important features of object oriented programming

Types of Polymorphism:

* Compile time Polymorphism
* Runtime Polymorphism.

Compile time Polymorphism: This type is obtained by function overloading or operator overloading.

* There are two types of Compile time Polymorphism.

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 number of arguments and/and change in type of arguments.

Example:-

```
#include <bits/stdc++.h>
using namespace std;
class Long
{ 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)
  { Count <<"value of x and y is"<<x<<endl;
  }
};
int main(){
    Long obj1;
    obj1.func(7);
    obj1.func(9.132);
    obj1.func(85.64);
  } return 0;
```

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 types. The advantage of Operator overloading 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("Welcome");
      A a2("back");
      a1+a2;
      return 0;
   }
```

Runtime Polymorphism:- In a runtime polymorphism, functions are called at the time the program execution. It is also known as late binding or dynamic binding.

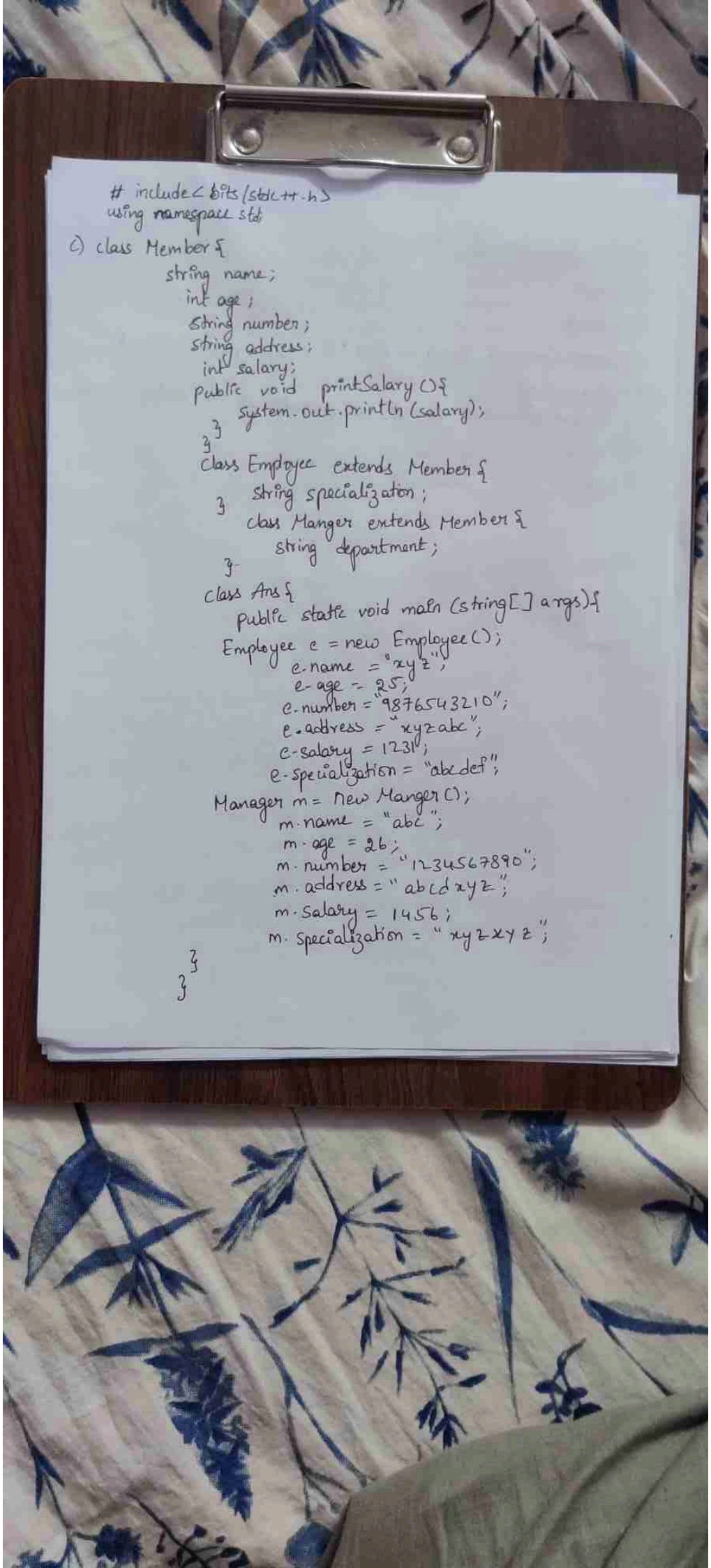
* Function overloading is a part of runtime polymorphism. In function ~~overloading~~ overriding, more than one method has the same name with different types of the parameter list.
* It is achieved by using virtual functions and pointers. It provide slow execution as it is known as runtime. It is more flexible as all the things executed at the run time.

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 (void)
    {  Animal A = Animal()
       A.function();
       Man m = Man();
       m.function();
       return 0;
    }
```

B)

```
# include <bits/stdc++.h>
  using namespace std;
void sort012 (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) / size of (arr[0]);
         sort012 (arr, n);
       cout << "array after segregation";
       printArray (arr, n);
         return 0;
       }
```



```
# include<bits/stdc++.h>
using namespace std
c) class Member{
        string name;
         int age;
         string number;
         string address;
          int salary;
          Public void printSalary(){
              system.out.println(salary);
          }
          class Employee extends Member{
              string specialization;
          }
              class Manger extends Member{
                  string department;
              }
          class Ans{
              Public static void main(string[] args){
              Employee e = new Employee();
                      e.name = "xyz";
                       e.age = 25;
                      e.number = "9876543210";
                      e.address = "xyzabc";
                      e.salary = 1231;
                     e.specialization = "abcdef";
                Manager m = new Manger();
                      m.name = "abc";
                       m.age = 26;
                      m.number = "1234567890";
                      m.address = "abcdxyz";
                      m.salary = 1456;
                     m.specialization = "xyzxyz";
    }
}
```