Faizan Ishan Siddiqui

Assignment - 2

Choose the correct option

(1) (d) Stack

(2) (c) Compiler error in Line "Derived *dp = new Base;"

(3) (a) Inaccessible

(4) (a) The number of times destructor is called depends on Number of objects created.

(5) (A) True

Short Answer type Questions

1- The new operator denotes a request for memory allocation on the free store.

To allocate memory of any data type, the syntax is:

pointer_variable = new data_type;

```
int *p = new (nothrow) int;
if (!p)
{
  cout << "Memory allocation failed \n";
}
```

The delete operator: Since it's programmer responsibility to deallocate dynamically allocated memory, programmers are provided delete operator.

Syntax - delete pointer-variable;

Ex # free the allocated memory

~~delete~~

delete p;

delete r;

free the block of allocated memory

delete [] q;

2 A constructor is a member function of a class which initialises objects of a class. In C++ constructor is automatically called when object (instance of class) created. It is a special member function of the class. The main purpose of class ~~of the~~ construction in C++ is to construct an object of the class.

Default constructor: It is a constructor which does not take any argument. It has no parameter.

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

Parameterized Then are the constructors with parameter using the construction you can provide different value to data member of different object by passing the appropriate value as argument.

Ex
```
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;
}
```

Copy constructor: Then are special type of constructor which take an object as argument and is used to copy value of data member of one object into other object.

Ex

```
class A
{
  public:
  int x;
  A(int a)
  {
    x = a;
  }
  A(A &i)
  {
    x = i.x;
  }
};
  int main()
  {
    A a1(20);
    A a2(a1);
    cout << a2.x;
    return 0;
  }
```

3 Object oriented programming:

- In OOP program is divided into small parts called objects.
- Object oriental programming follows bottom up approach.
- OOP have access specifiers like private, public, protected etc
- adding new data and function is easy.
- OOP provides data hiding so it is more secure
- OOP based on real world.
- Ex C++, Java, Python, etc.

procedural Oriented programming.

- In POP program is divided into small parts called function
- It follows top down approach.
- There is no access specifiers in procedural programming.
- Adding new data and function is not easy.
- It does not have any proper way for data hiding so it is less secure.
- It is based on ~~unreal~~ unreal world.
- Ex C FORTRAN etc.

(c)

```
class Member
{
String name;
int age;
string number;
String address;
int salary;
public void printSalary();
{
system.out.println(salary);
}
}
class Employer extends Member
{
  String specialization;
}
class Manager extends member
{
  String department;
}
  class Ans
{ public static void main (String [])
{ Employee e = new employer();
 e.name = "xyz";
 e.age = 23;
 e.number = "12345678";
 e.address = "xyzxyz";
 e.specialization = "xyzxyz";
Manager m = new manager();
```

```
m.name = "xyz";
m.age = 23;
m.number = "12345678";
m.address = "xyz xyz";
m.salary = "1111";
m.specialization = "xyz xyz";
}
};
```

(B) Write a C++ program to sort an array with 0, 1, and 2 in single pass.

```
#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;}}}
```

```
int main()
{
    int arr[] = {0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    sort012(arr, n);
    cout << "array after segregation";
    printArray(arr, n);
    return 0;
}
```

(A) Type of polymorphism.

C++ support two type of polymorphism:

Compile time polymorphism:

You invoke the overloaded function by matching the number and type of argument The information is present during compile time This means the C++ compiler will select the right function at compile time.

Compile time polymorphism is achieved through function overloading and operator overloading.

Function Overloading: Function Overloading occurs when we have many function with similar name but different argument. This arguments may differ in terms of number or type.

Ex

```
void test (int i)
{
cout << "The int is :" << i << endl;
}
void test (double f)
{
cout << "The float is :" << f << endl;
}
void test (char const *ch)
{
cout << "The char is :" << ch << endl;
}
int main()
{
  test (5);
  test (5.5);
  test ("Five");
  return 0;
}
```

Operator Overloading :- In operator overloading we define a new meaning for a C++ operator. It also change how the operator works. Ex- we can define the + operator to string we know that it is addition operator for adding numeric values after our definition when placed b/w integers it will add them when place between strings it will concatenate them.

```
class complex num
{
  Private :
    int real, over;
  public:
    complex Num (int rl=0; int ov=0)
  { real = rl;
    over = ov;
  }
 complex Num operator + complex num; Cons & Obj
 {
   ⊙ complex Num result;
 result real = real + obj : real;
 result over = over + obj: over;
 return result;
 }
  void print ()
  {
    cout << real << "+i" << over << endl;
    }
  }
  int main ()
  {
    complex Num c1(10,2), c2(3,7);
    complex Num c3 = c1+c2;
     c3. print ();
  }.
```

Runtime polymorphism

This happens when one object method is invoked called during runtime rather than during compile time. Runtime polymorphism is achieved through function overriding this function to be called invoked is established during runtime.

function Overriding :- It occurs when a function of the base class is given a new definition in a derived class. At that time we can say. The base function has been overriding.

ex

```
class mammal
{ public:
    void eat()
  { cout << " mammals eat... ";
  }
};
class cow : public mammal :
public:
    void eat()
{ cout << " Cow eat gras";
  }
}; int main(void) {
    cow c = cow();
    c.eat();
    return 0;
    }
```

Virtual function

A virtual function is another way of implementing run time polymorphism in C++. It is special function defined in a base class and redefined in the derived class. To declare a virtual keyword. The virtual keyword should precedes the declaration of the function in the Base class

If the virtual virtual function class is inherited the the virtual class redefines the virtual function to suit it's needs

Ex

```
Class class A
{
  public:
    virtual void show()
{ cout << " The show function in base class invoked
        "<< endl;
  }
};
class class B : public class A
{ public:
    void show()
    { cout << " This show function is derived in
      class invoked "; } };
  int main() {
  { class A *a;
    class B . b;
    a = &b;
    a -> show();
    }
```