CIPHER SCHOOL CPP-LPU

ASSIGNMENT - 2

Choose the correct option

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

⇒ Ans: d) Stack.

2) Choose the correct option?

.....

⇒ Ans: c) Compiler error in line "Derived *dp = new Base;"

3) When the inheritance is private, the private methods in base class are ______ in the derived class (in C++)

Ans: a) Inaccesible

4) Which of the following is true?

→ A) The no. of times a constructor is called depends on no. of objects created.

5) State true or false

~~→ B) False~~

→ A) True

Short answer type question

1. Explain about new and delete keywords with code

→ The new and delete keywords are used in situations where Dynamic Memory Allocations are required

The 'new' keyword is used to allocate space dynamically. It is a unary operator and is suffixed by the type being allocated.

Code:
```
int *p;  // declares a pointer 'p' which points
         an int type data.

p = new int;  // dynamically allocate memory
              to contain one single element
              of type int and store the address
              in 'p'.
```

The 'delete' keyword is used to free memory dynamically. There are some memory that are allocated for specifics period of time and once no longer needed, are freed using the 'delete'.

Code:
```
delete p;  // releases memory allocated using
           int *p.
delete []p;  // releases memory allocated
             using int *p = new int[5];
```

Here, the first statement releases the memory of a single element allocated using new, and the second one releases the memory allocated for arrays of ~~et~~ elements using new and a size in brackets [].

2. What are constructors? Why are they required? Explain different types of constructors with suitable examples.

→ A constructor is a member function of a class which initializes objects of a class. In C++ they are automatically called when objects are created.

There are 3 types of constructors in C++, they are:

(i) Default
(ii) Parameterised
(iii) Copy

(i) Default constructor: It takes no argument and has no parameters.

Eg:
```
class ....
{
  public:
    ....
   // default constructor
     construct()
   {
      ....
   }
};
```

(ii) Parameterized constructor : They are used when we want to pass arguments to constructors. A constructor can be parameterized by simply adding parameters like that in functions.

~~(iii)~~ Eg: class para

```
class para
{
  ....
  public:
    //parametrized constructor
    para (int x , int y)
      { ...
      }
};
int main()
{  //calling parametrized constructor
   para p1 (1, 2);
   ...
};
```

(iii) Copy constructor : A member function which initializes an object using another object of same class.

3. Explain the difference b/w OOP and Procedural programming language in detail

→ Procedural Programming is a conventional approach which ~~is~~ follows a top-down approach whereas Object oriented Programming is a newer approach related to real life objects and their properties and follows a ~~[illegible]~~ bottom-up approach.

OOP	POP
Object Oriented	Structure oriented
Program is divided into objects	Program is divided into functions
Inheritance concept	No inheritance concept
Has access specifier	No access specifier
Encapsulation concept	No encapsulation concept
Virtual functions	No virtual functions
C++, Java	C, Pascal

Long answer type questions

A) Explain the types of polymorphism with code

→ In C++ we have two types of polymorphism

1) compile time polymorphism : known as static.
2) Runtime polymorphism : known as dynamic.

1) Compile time polymorphism : whenever an object is bound with their functionality at the compile time, it is known as compile time polymorphism. Function overloading and operator overloading are perfect examples of compile time polymorphism.

Example program :

```
#include <bits/stdc++.h>
using namespace std;
class Add {
        public:
            int sum (int n1, int n2){
                    return n1 + n2;
            }
            int sum ( int n1 , int n2, int n3){
                    return n1 + n2 + n3;
            }
};
```

```
int  main(){
    Add obj;
    //calling first function
    cout <<" Sum:  "<< obj.sum (1, 2) << endl;
    //calling second function
    cout << " Sum :  " << obj.sum (1, 2, 3) << endl;
 return 0;
 }
```

Output :

```
Sum: 3
Sum : 6
```

---

2) Runtime polymorphism : It is a process in which a call to an overriden method is resolved at runtime rather than complile time. In this process, an overriden method is called through the reference variable of the superclass.

Eg. program :

```
#include <bits/stdc++.h>
using namespace std;
class A{
public:
    void display (){
        cout << "Super class function" << endl;
    }

};
```

```
class B: public A{
 public:
     void disp(){
        cout << " Sub class function ";
     }
 };
 int main(){
     A. obj;
     obj. display();
     B. obj 2;
     obj 2 .disp();
     return 0;
 };
```

Output:

```
Super class function
Sub class function
```

B) Write a program to sort an array of 0, 1, 2 in the best possible time and space complexity.

→

```
#include <bits/stdc++.h>
 using namespace std;
 // function to sort input array
 void sort (int a[], int a-size)
 {
        int l = 0;
        int m = 0;
        int h = a-size - 1;

  // iterating all elements till they are sorted
        while (m <= h){
           switch (a[m]){
               case 0 :
                   swap (a[l++], a[m++]);
                   break;
               case 1 :
                    m++;
                    break;
                case 2 :
                    swap (a[m], a[h--]);
                    break;
           }
        }
 }
  // function to print array
 void printA (int arr[], int arr-size)
 {
      for (int i = 0; i<arr-size; i++)
        cout << arr [i] << " ";
 }
```

```
int main()
{
    int arr[] = {0, 1, 0, 2, 1, 1, 0, 2, 0};
    int n = sizeof(arr) / sizeof(arr[0]);

     sort(arr, n);

     cout<< " Sorted array: ";

      printA(arr, n);
      return 0;
  }
```

Complexity analysis :

Time complexity : O(n)
Only one traversal of arrays is needed

Space complexity : O(1)
No extra space is required.

c) Create a class named 'Member' having the following members:

→
```
#include <bits/stdc++.h>
 using namespace std;
 class Member{
    public:
        string Name;
        int Age;
        string Phone_Number;
        string Address;
        double Salary;

        void  printSalary(){
              cout << "Salary: " << Salary;
        }
 };
 class Employee : Public Member{
     public:
         string specialization;
         void disp(){
              cout << "\n Name: " << Name;
              cout << "\n Age : " << Age;
              cout << "\n Phone Number : " << Phone_Numbe
              cout << "\n Address: " << Address;
              cout << "\n Salary : " << Salary;
              cout << "\n Specialization: " << specialization;
         }
 };
```

```
class Manager : public Member{
    public:
        string department;
        void disp(){
                cout<<"\n Name:" << Name ;
                cout<< "\n Age: "<< Age ;
                cout<< "\n Phone Number:"<<Phone_Number;
                cout << "\n Address: "<< Address;
                cout << "\n Salary : " << Salary;
                cout <<"\n department: "<< department ;
        }
    };

int main(){
        Employee e;
        e.Name = " Bob ";
        e. Age = 30;
        e. Phone_Number = "9898998899";
        e. Address = "new town ";
        e. Salary = 10000;
        e. specialization = "data ";
        e. disp();
        cout << endl;
        Manager m;
        m. Name = " ted";
        m. Age = 25;
        m. Phone_Number = "1800180099";
        m. Address = "old town ";
        m. Salary = 15000;
        m. department = "management ";
        m. disp();

        return 0;
    }
```