SOLUTIONS TO OOPS (I.T) MID SEMESTER-2018

1. a.(i)

b.(ii)

2.(a)Since every function has complete access to the global variables, the new programmer can corrupt the data accidentally by creating function. Similarly, if new data is to be added, all the function needed to be modified to access the data.Meaning in now data already is created is secured.In POP, Data can move freely from function to function in the system.

In following ways oops solve the problems:

(i).In OOP, objects can move and communicate with each other through member functions.

(ii).In OOP, objects can move and communicate with each other through member functions.

(iii).OOP provides Data Hiding so provides more security.

(b).

```
#include<iostream>
using namespace std;
    class Test {
    int num;
public:
        Test() {
          cout << "Enter a Number :";
          cin>>num;
          if (num % 2 == 0) {
                cout << "\nNumber is Even :" << num;
          } else {
                cout << "\nNumber is Odd :" << num;
          }
    };
    int main() {
```

```
    Test tt;

    return 0;

}
```

3.(a)A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.It has a keyword "Friend".

A friend function has the following advantages :

(i). Provides additional functionality which is kept outside the class.

(ii). Provides functions that need data which is not normally used by the class.

(iii). Allows sharing private class information by a non member function.

(b).Functions can be instructed to compiler to make them inline so that compiler can replace those function definition wherever those are being called.These types of functions are called Inline functions.

Advantages of inline functions:

1) Function call overhead doesn’t occur.

2) It also saves the overhead of push/pop variables on the stack when function is called.

3) It also saves overhead of a return call from a function.

```
#include <iostream.h>

#include <conio.h>

inline int cube(int r) {

return r*r*r;

}

void main() {

clrscr();

int r;

cout<<"PROGRAM TO COMPUTE CUBE\n";

cout<<"Enter value to compute cube: ";

cin>>r;
```

```
cout<<"cube of the number: "<<cube(r);

getch();

}
```

4.(a)Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.A template is a blueprint or formula for creating a generic class or a function. The library containers like iterators and algorithms are examples of generic programming and have been developed using template concept.

```
#include <iostream>

using namespace std;

template <typename T>

T myMax(T x, T y)

{

    return (x > y)? x: y;

}

  int main()

{

  cout << myMax<int>(3, 7) << endl;

  cout << myMax<double>(3.0, 7.0) << endl;

  cout << myMax<char>('g', 'e') << endl;

  return 0;

}
```

Function Templates :-

```
ex---- int max(const int& a, const int& b) { ... }

            string max(const string& a, const string& b) { ... }
```

- We often find a set of functions that look very much alike,

e.g. for a certain type T, the max function has the form

T max(const T& a, const T& b) { ... }

- In C++, we can define one single function definition

using templates:

template<typename T>

T max(const T& a, const T& b) .

Class Templates:-

- The template mechanism works for classes as well. This is particularly useful for defining container classes.

In the next few slides we will define template<typename T> class List_Nodeand a container classtemplate<typename T> class List that uses List_Node.

- Note the line friend class list<T>in the definition of List_Node.

(b).Two or more functions having same name but different argument(s) are known as overloaded functions.

```
#include <iostream>

using namespace std;

class Addition {

public:

    int sum(int num1,int num2) {

        return num1+num2;

    }

    int sum(float num1,float num2,) {

        return num1+num2+num3;

    }

};

int main() {

    Addition obj;

    cout<<obj.sum(20, 15)<<endl;
```

```
        cout<<obj.sum(1.1, 8.6);
        return 0;
}
```

5.Operator Overloading is a type of polymorphism in which an operator is overloaded to give user defined meaning to it.When an operator is defined as a member, the number of explicit parameters is reduced by one, as the calling object is implicitly supplied as an operand. Thus, binary operators take one explicit parameter and unary operators none. In the case of binary operators, the left hand operand is the calling object, and no type coercion will be done upon it. This is in contrast to non-member operators, where the left hand operand may be coerced.

```
include <iostream.h>
#include <conio.h>
class Minus
{
     private:
                              int a, b, c;
       public:
                               Minus( ) {}
                               Minus(int A, int S, int C)
                    {
                              a = A;
                              b = S;
                              c = C;
                     }
                              void display(void);
                              void operator - ( );
                inline void Minus:: display(void)
          {
```

```
        cout<<"\t a="<< a <<endl;
        cout<<"\t b=" << b <<endl;
        cout<<"\t c = "<< c<< endl;
    }
        inline void Minus :: operator - ()
    {
            a = -a;
            b = -b;
            c = -c;          }
        void main(void)
{
        clrscr();
        Minus M(5, 10, -15);
        cout <<"\nBefore Activating the Operator - ( )\n";
        M.display( );
        -M;
        cout<< "\nAfter Activating the Operator - ( )\n";
        M.display( );
        getch();
    }
```

6.In object-oriented programming, inheritance is the concept that when a class of objects is defined, any subclass that is defined can inherit the definitions of one or more general classes.

There are Various types of inheritance in Java:

i. Single Inheritance

ii. Multiple Inheritance

iii. Multilevel Inheritanc:

iv. Hierarchical Inheritance

v. Hybrid Inheritance

Examples:

i. >Single Inheritance

```
#include <iostream>
using namespace std;
class Vehicle {
    public:
        Vehicle()
        {
          cout << "This is a Vehicle" << endl;
        }
};
class Car: public Vehicle
};
int main()
{
        Car obj;
        return 0;
}
```

ii. Multiple

```
class subclass_name : access_mode base_class1, access_mode base_class2, ....
{
    //body of subclass
};
```

iii.Multilevel

```
class Vehicle
{
    public:
        Vehicle()
        {
            cout << "This is a Vehicle" << endl;
        }
};
class fourWheeler: public Vehicle
{    public:
        fourWheeler()
        {
            cout<<"Objects with 4 wheels are vehicles"<<endl;
        }
};
// sub class derived from two base classes
class Car: public fourWheeler{
      public:
          car()
          {
              cout<<"Car has 4 Wheels"<<endl;
          }
```

iv.Hierarchial

```
class Vehicle
```

```
{
    public:
        Vehicle()
        { cout << "This is a Vehicle" << endl;      }
};
  // first sub class
class Car: public Vehicle
{
    };
// second sub class
class Bus: public Vehicle
{
v.Hybrid
class Fare
{
        public:
        Fare()
        {
              cout<<"Fare of Vehicle\n";
        } };
  // first sub class
class Car: public Vehicle
{
}; // second sub class
class Bus: public Vehicle, public Fare
```

(b).If bca is having some member functions and variables then using inheritace bca will access these functions from abc.If cab is also inheriting from abc them member functions will access it from bca.

7.

(a)A constructor is a special method of a class or structure in object-oriented programming that initializes an object of that type. A constructor is an instance method that usually has the same name as the class, and can be used to set the values of the members of an object, either to default or to user-defined values.

Constructors are of three types : Default Constructor. Parametrized Constructor. Copy Constructor.

Default Constructors: Default constructor is the constructor which doesn't take any argument. It has no parameters.

```
#include <iostream>

using namespace std;

  class construct {

public:

        int a, b;

  // Default Constructor

        construct()

        {

                a = 10;

                b = 20;

        }

};

  int main()

{

        // Default constructor called automatically

        // when the object is created

        construct c;
```

```
    cout << "a: " << c.a << endl
         << "b: " << c.b;
    return 1;
}
```

Parameterized Constructors: It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.

```
#include <iostream>
using namespace std;
  class Point {
private:
    int x, y;
  public:
    // Parameterized Constructor
    Point(int x1, int y1)
    {
        x = x1;
        y = y1;
    }
  int getX()
    {
        return x;
    }
    int getY()
    {
        return y;
```

```
        }
};
    int main()
{
        // Constructor called
        Point p1(10, 15);
       // Access values assigned by constructor
        cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
    return 0;
}
```

(b).

```
#include <stdio.h>
int main()
{
        int arr[100], freq[100];
        int size, i, j, count;
        printf("Enter size of array: ");
        scanf("%d", &size);
        printf("Enter elements in array: ");
        for(i=0; i<size; i++)
        {
                scanf("%d", &arr[i]);
                freq[i] = -1;
        }
    for(i=0; i<size; i++)
```

```
    {
        count = 1;
        for(j=i+1; j<size; j++)
        {                   if(arr[i]==arr[j])
            {
                count++;
                freq[j] = 0;
            }
        }           if(freq[i] != 0)
        {
            freq[i] = count;
        }
    }       printf("\nFrequency of all elements of array : \n");
    for(i=0; i<size; i++)
    {
        if(freq[i] != 0)
        {
            printf("%d occurs %d times\n", arr[i], freq[i]);
        }
    }
  return 0;
}
```