

## Computer Programming -I

Q.No.1. Find o/p of the following programs(5 marks each)

20

**Note : Please pay attention for bold part of the Q.No-1  
Outputs are according to those changes.**

a. 

```
void main ()
{
int a=1,b=2,c=3,d=4.75,x;
x=++a + b++ * ++c %d++;
cout<<a<< " "<<b<< " "<<c<< " "<<d<< " "<<x<< " "<<endl;
}
```

Ans:

Output  
2 3 4 5 2

b. 

```
void main()
{
int x=1;
cout<<x<<(x=x+2)<<(x<2)<<endl;
x<2;
cout<<++x<<x++<<++x<<endl;
}
```

Ans:

Output: 3 3 4  
6 4 4

c. 

```
int x;
void f1()
{
++x;
}
void main()
{
int x=10;
f1();
x=::x+10;
cout<<x<< " "<<::x<<endl;
}
```

Ans:

Output: 11 1

d. 

```
class Test
{
public:
Test()
{
cout<<" constructor"<<endl;
}
```

## Computer Programming -I

```

~Test()
{
    Cout<<"destructor"<<endl;
}
};
void main()
{
    Test t1;
    {
        Test t2,t3;
    }
    Test t4;
}

```

Ans:

Output:

```

constructor
constructor
constructor
destructor
destructor
constructor
destructor
destructor

```

Q.No.2. a. Write a program to print following pattern.

```

1
21A
321AB
4321ABC
54321ABCD

```

10

Ans:

```

#include<iostream.h>
Void main()
{
    int r,lc,rc,b;
    for(r=1;r<=5;r++)
    {
        //blank space
        for(b=1;b<=5;b++) cout<<" ";

        //left column of the row
        for(lc=r;lc>=1;lc--) cout<<lc;

        //right column of the row
        for(rc=65;rc<=63;rc++) cout<<rc;

        //move cursor to next line
        cout<<endl;
    }
}

```

## Computer Programming -I

```

}
}

```

b. Write the program for following expression

10

$$X=1!+3!+5!+\dots+(2n-1)!$$

```

#include<iostream.h>
Void main()
{
  Int n,no;
  Float x=0,fact;
  Cout<<"Input n"<<endl;
  Cin>>n;
  For(no=1;no<=2*n-1;no=no+2)
  {
    Fact=1;
    For(int i=1;i<=no;i++) fact*=i;
    X=x+fact;
  }
  Cout<<"x="<<endl;
}

```

Q.No.3. a. What is recursion? Write a function to find nth Fibonacci term and use this function to find n Fibonacci terms.

10

Ans:

Recursion:- Recursion is when a function calls itself again and again. That is, in the course of the function definition there is a call to that very same function.

Program:-

```

#include<iostream.h>
Int fib(int n)
{
  If(n= =1 || n= =2) return(1);
  Else return (fib(n-2)+fib(n-1));
}

```

Computer Programming -I

```
}  
Void main()  
{  
  Int I,n;  
  Cout<<"input n"<<endl;  
  Cin>>n>>endl;  
  For(i=1;i<=n;i++) cout<<fib(i)<<endl;  
}
```

- b. What is Function overloading? Overload function add to two integers, two float and two arrays. 10

Ans:

**Function Overloading:**

This process of using two or more functions with the same name but differing in the signature is called function overloading.

**Advantage**

The biggest advantage of overloading is that it helps us to perform same operations on different datatypes without having the need to use separate names for each version.

**Example**

```
#include<iostream>  
using namespace std;  
int abslt(int );  
long abslt(long );  
float abslt(float );  
double abslt(double );  
int main()  
{  
  int intgr=-5;  
  long lng=34225;  
  float flt=-5.56;
```

Computer Programming -I

---

```
double dbl=-45.6768;

cout<<" absolute value of \ "<<intgr<<" = \ "<<abslt(intgr)<<endl;

cout<<" absolute value of \ "<<lng<<" = \ "<<abslt(lng)<<endl;

cout<<" absolute value of \ "<<flt<<" = \ "<<abslt(flt)<<endl;

cout<<" absolute value of \ "<<dbl<<" = \ "<<abslt(dbl)<<endl;

}

int abslt(int num)
{
if(num>=0)
return num;
else
return (-num);
}

long abslt(long num)
{
if(num>=0)
return num;
else return (-num);
}

float abslt(float num)
{
if(num>=0)
return num;
else return (-num);
}

double abslt(double num)
```

## Computer Programming -I

```

{
if(num>=0)

return num;

else return (-num);

}

```

## OUTPUT

absoulte value of -5 = 5

absoulte value of 34225 = 34225

absoulte value of -5.56 = 5.56

absoulte value of -45.6768 = 45.6768

Q.No.4. a. *What is operator overloading? Overload operator +,+=,++ on complex numbers. ++ increments real and imaginary parts by 1.* 10

Ans: Operator overloading is a mechanism where meaning of existing system defined operators can be modified by user to make them work for their own classes. Operator can be overloaded by defining function for it.

```

#include<iostream.h>
#include<math.h>
class complex
{
int rp,ip;
public:
void getComplex();
void printComplex();
Complex operator+(Complex c);
Complex operator +=(Complex c);
Complex operator ++();
};
void Complex::getComplex()
{
cin>>rp>>ip;
}
void Complex::printComplex()
{
if(ip<0)
cout<<rp<<"-i"<<abs(ip)<<endl;
else

```

## Computer Programming -I

```

cout<<rp<<"i"<<ip<<endl;
}
Complex Complex::operator+(Complex c)
{
Complex t;
t.rp=rp+c.rp;
t.ip=ip+c.ip;
return(t);
}
Complex Complex::operator++()
{
++rp;
++ip;
return(*this);
}
void main()
{
Complex c1,c2,a;
c1.getComplex();
c2.getComplex();
a=c1+c2;
a.printComplex();
c1+=c2;
c1.printComplex();
c2.printComplex();
++c1;
c1.printComplex();
}

```

Q.No.4 b. Find output

10

```

Void f()
{
Extern int n3;
Static int n1;
Int n2=20;
N1=n1+10;
N2=n1+n2;
N3=n1+n2;
Cout<<n1<<" "<<n2<<" "<<n3<< endl;
}

```

Ans:

10 30 40

20 40 60

30 50 80

## Computer Programming -I

Q.No.4. c. *Write a program to count spaces, digits, vowels, constants in the string.* 10

Ans:

```
#include<istream.h>
#include<conio.h>
#include<String.h>
void main()
{
    int sc=0,dc=0,vc=0,cc=0;
    char *s;
    cout<<"Input String:";
    gets(s);
    for(int i=0;i<strlen(s);i++)
        if(isdigit(s[i]))
            ++dc;
        else if(isspace(s[i]))
            sc++;
        else if(isalpha(s[i]))
            switch(toupper(s[i]))
            {
                case 'A':
                case 'E':
                case 'T':
                case 'O':
                case 'U':
                    ++vc;
                default:
                    ++cc;
            }
    }
    cout<<"Number of spaces:"<<sc<<endl;
    cout<<"Number of digits"<<dc<<endl;
```

Q.No.5 a. *Write a program to create a singly linked list of 10 student nodes. The student node contain roll no and percentage. Read information and print information and print information in the linked list. Node can be structured or class.*

Ans:

```
#include<istream.h>

struct Student
{
    int rn;
    float p;
    Student *next;
};

void main()
{
    Student *node,*newnode,*start,*t;
    //create first node
```

## Computer Programming -I

```

node=new Student;
cin>>node->rn>>node->p;
node->next=null;
//making this node start node
start=node;
//create remaining 9 nodes
for(int i=1;i<=9;i++)
{
    newnode= new newnode;
    cin>>newnode>>rn>>newnode->p;
    newnode->next=null;
    node->next=newnode;
    node=newnode;
}
Student *start;
while(t!=null)
{
    cout<<t<<rn<<t->p<<endl;
    t=t->next;
}
}

```

Q.No.5. b. *Write a program to using pointer to allocate memory for 10 integers. Read and print these integers. Find average of these integers.*

Ans:

```

#include<iostream.h>
void main()
{
    int *p,s=0;
    float a;
    cout<<"Input 10 integers"<<endl;
    for(int i=0;i<10;i++)
    {
        cin>>*(p+i);
        s/=(p+i);
    }
    a=s/10.0;
    cout<<"Integers input"<<endl;
    for(i=0;i<10;i++)
    cout<<*(p+i)<<endl;
    cout<<"Average="<<a<<endl
}

```

Q.No.6 a. *What is function overriding. Give example.*

Computer Programming -I

Ans: Definition:- A Function of derive class and inherited function from base class has same signature (same return type , same parameters and name.) The function of derive class override inherited function of base class.

e.g

```
#include<iostream.h>
```

```
class Base
{
public:
void f1()
{
cout<<"f1() of base"<<endl;
}
};
class Derive: public Base
{
public:
void f1()
{
cout<<"f1 is derived"<<endl;
}
};
void main()
{
Derive d;
d.f1;
}
```

Q.No.6 b. Explain Inheritance visibility.

Ans: Inheritance Visibility

In C++ there are three inheritance access specifiers:

- public
- protected
- private

The three access modifiers public, protected and private are analogous to the access modifiers used for class members.

**public**

When a base class is specified as public ie: class c : public base {}; the base class can be seen by anyone who has access to the derived class. That is, any members inherited from base can be seen by code accessing c.

**protected**

Computer Programming -I

When a base class is specified as protected ie: class c : protected base {}; the base class can only be seen by subclasses of C.

**private**

When a base class is specified as private ie: class c : private base {}; the base class can only be seen by the class C itself.

**Public inheritance**

When you inherit a base class publicly, all members keep their original access specifications. Private members stay private, protected members stay protected, and public members stay public.

**Private inheritance**

With private inheritance, all members from the base class are inherited as private. This means private members stay private, and protected and public members become private.

**Protected inheritance**

Protected inheritance is the last method of inheritance. It is almost never used, except in very particular cases. With protected inheritance, the public and protected members become protected, and private members stay private.

Q.No.6. c. *Create class Circle with data members radius and member functions to read radius, print radius, and calculate radius. Derive class cylinder from class circle. Class cylinder should have data member height and inherited radius from circle and member functions to read height and radius and calculate area.*

Ans:

```
#include<iostream.h>
#include<math.h>
class Circle
{
protected:
float r,a;
public:
void getradius()
{
cin r;
}

void printradius()
{
cout<<r<<endl;
}
float area()
{
```

## Computer Programming -I

```

return (3.14*r*);
}
};

class Cylinder:public Circle
{
protected:
float h;
public:
void getheight()
{
getRadius();
cin>>h;
}

void printheight()
{
printRadius();
cout<<h<<endl;
}
float area()
{
return (3.14*r*r*h);
}
};

void main()
{
Circle cr;
Cylinder cy;
cr.getRadius();
cout<<"Area od circle="<<cr.area()<<endl;
cy.getHeight();
cout<<"Area of Cylinder:"<<cy.area()<<endl;
}

```

Q.No.7. a. *Explain virtual function with example.*

Ans: **Virtual Function:-**

In object-oriented programming, a virtual function or virtual method is a function or method whose behaviour can be overridden within an inheriting class by a function with the same signature. This concept is a very important part of the polymorphism portion of object-oriented programming (OOP).

C++ Virtual Function - Properties:

- C++ virtual function is,
- A member function of a class
- Declared with virtual keyword

## Computer Programming -I

- Usually has a different functionality in the derived class
- A function call is resolved at run-time

The difference between a non-virtual c++ member function and a virtual member function is, the non-virtual member functions are resolved at compile time. This mechanism is called static binding. Where as the c++ virtual member functions are resolved during run-time. This mechanism is known as dynamic binding.

```
class Derive:public Base
{
public:
void f1()
{
cout<<"f1() of Derive"<<endl;
}
}
void main()
{
Base *p;
Derive d;
p=&d;
p->f1();
}
```

b. *Explain pure virtual function with example.*

Ans:

A virtual function with no body structure is called pure virtual function. We can declare a pure virtual function by adding the notation =0 to the virtual function declaration. For example, area() is a virtual function, but when it is declared ***virtual int area() = 0***, it becomes a pure virtual function.

```
#include <iostream.h>
Class Polygon
{
protected:
int width, height;

public:
void setup(int first, int second)
{
width = first;
height = second;
}
virtual int area () = 0;
};
```

```
class Rectangle: public Polygon
{
```

## Computer Programming -I

```

public: int area()
{
    return(width * height);
}
};
class Triangle: public Polygon
{
public :
int area()
{
    return(width * height / 2)
}
};
int main()
{
    Rectangle rectangle;
    Triangle triangle;
    Polygon * polygon;
    Polygon * ptr_polygon1 = &rectangle;
    Polygon * ptr_polygon2 = &triangle;
    ptr_polygon1 -> setup(2,2);
    ptr_polygon2 -> setup(2,2);
    cout<<ptr_polygon1->area()<<endl;
    cout<<ptr_polygon2->area()<<endl;
    return 0;
}

```

Output:

```

4
2

```

c.

*Write pure virtual function convert to convert liters to milliliters. grams to kilograms, dollar to rupee. Assume 1 dollar is 45 rupees.*

Ans:

```

#include<iostream.h>
class Convert
{
protected:
    int n1,n2;
public:
    virtual void convert()=0;
};

class L_ML
{
public:

```

## Computer Programming -I

```
void convert()
{
    cout<<"Input liters"<<endl;
    cin>>n1;
    n2=n1*1000;
    cout<<"Liters="<<n1<<endl;
    cout<<"Mililiters="<<n2<<endl;
}
};

class G_KG
{
public:
    void convert()
    {
        cout<<"Input Grams"<<endl;
        cin>>n1;
        cout<<"Grams="<<n1<<endl;
        cout<<"Miligrams="<<n2<<endl;
    }
};

class D_R
{
public:
    void convert()
    {
        cout<<"Input Dollers"<<endl;
        cin>>n1;
        n2=n1*45;
        cout<<"Dollers="<<n1<<endl;
        cout<<"Rupees="<<n2<<endl;
    }
};

void main()
{
    Convert *p;
    p=new L_ML;
    p->convert();
    p=new G_KG;
    p->convert();
    p=new D_R;
    p->convert();
}
```