DATE : / /
PAGE NO. :

Assignment Questions

Q1. Write a C++ program to print following pattern :-

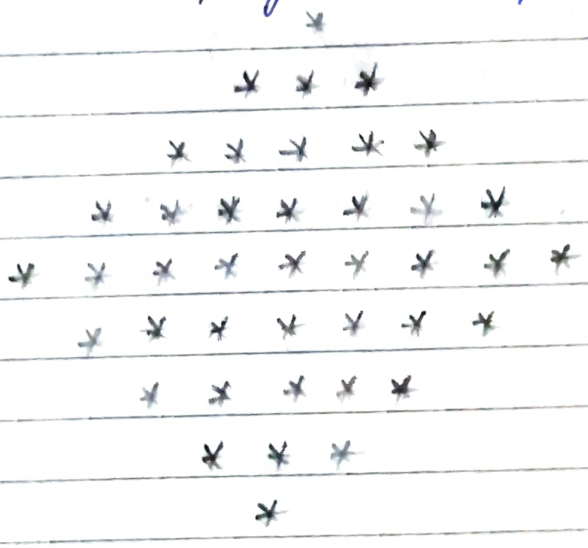
Ans.

```
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    int p, k, r;
    cout <<" Enter no of rows : ";
    cin >> r;
    p = 1;
    while ( p<= r)
    {
        k = p;
        while ( k<= r)
        {
            cout <<"*";
            k++;
        }
    k = 1;
```

```
while ( k<= (2*p-2))
{
    cout <<" ";
    k++;
}
k = p;
   while (k <= r)
   {
    cout <<"*";
    k++;
   }
   cout<<"\n";
   p++;
}
    p = 1;
   while (p<= r)
   {
    cout << "*";
    k++;
   }
  k = (2*p-2);
  while( k< (2*r))
   {
     cout <<" ";
     k++;
   }
  k = 1;
   while (k<= p)
   {
     cout <<"*";
```

```
            k++;
        }
        cout << "\n";
        p++;
    }
    getch();
    return 0;
}
```

Q2 write a c++ program to print following pattern:



Ans.

```
#include <iostream>
using namespace std;
int main()
{
    int i, j, row, space;
    cout << "Enter no. of rows: ";
    cin >> row;
    space = row - 1;
    for (i = 1; i <= row; i++)
    {
```

```
        for (j=1; j<=space; j++)
            cout << " ";
            space--;
            for (j=1; j<(2*i-1); j++)
                cout << "*";
            cout << endl;
        }
        space = 1;
        for (i=1; i<=(row-1); i++
        {
            for (j=1; j<=space; j++)
                cout << " ";
            space++;
                for (j=1; j<=(2*(row-i)-1); j++)
                cout << "*";
            cout << endl;
        }
    cout << endl; return 0;
    }
```

Q3. Explain this pointer in C.

Ans. Every object in C++ has access to its own address through an important pointer called this pointer. The this pointer is an implicit parameter to all member function. Therefore, inside a member function, this may be used to refer to the invoking object.

DATE / /
PAGE NO :

Q4. WAP that defines a class "Time" that store time values as hours, min and sec respectively. Perform operations $t_3 = t_1 + t_2$ on its object.

Ans.

```
#include<iostream>
using namespace std;
class Time
{
   private :
      int hours;
      int min;
      int sec;
   public :
      void getTime(void);
      void putTime(void);
      void addTime (Time T1, Time T2);
};
void Time :: getTime (void)
{
    cout <<" Enter Time : " << endl.
    cout << " hours ? " ; cin >> hours;
    cout << " Minutes ? " ; cin >> min;
    cout << " Seconds ? " ; cin >> sec;
}
void Time :: putTime (void)
{
    cout << endl;
    cout << " Time after adding : ";
    cout << " hours << " : " << min << " : " << sec
                << endl;
}
```

```
void Time :: addTime ( Time T1, Time T2 )
{
    this -> secs = T1.sec + T2.sec;
    this -> min = T1.min + T2.min + this -> sec /60;
    this -> hours = T1.hours + T2.hours + (this -> min/60);
    this -> min %= 60;
    this -> sec %= 60;
}

int main ()
{
    Time T1, T2, T3;
    T1.getTime ();
    T2.getTime ();
    T3.addTime (T1, T2);
    T3.putTime ();

    return 0;
}
```

Q5. WAP to overload unary pre and post increment.

Ans. Pre-increment overloading :-

```
using namespace std;
class Integer
{ private :
      int i;
  public :
    Integer ( int i=0)
```

```
    {
        this -> i = i;
    }

    Integer operator ++ ()
    {
        Integer temp;
        temp.i = ++i;
        return temp;
    }
    void display ()
    {
        cout << "i = " << i << endl;
    }
};
    int main()
    {
        Integer i1(3);
        cout << "Before increment : ";
        i1.display();

        Integer i2 = ++i1;
        cout << "After pre increment : ";
        i2.display();
    }
```

DATE / /
PAGE NO

Q6. Explain dynamic constructor. WAP to initialize array using dynamic constructor.

Ans. When allocation of memory is done dynamically using dynamic memory allocator new in a constructor, it is known as dynamic constructor.

```
#include<iostream>
using namespace std;
class Constructor
{
    const char* p;
    public:
        Constructor()
        {
            p = new char[6];
            p = "constructor";
        }
    void display()
    {
        cout<<p<<endl;
    }
};
int main()
{
    Constructor obj;
    obj.display();
}
```

DATE
PAGE NO

Q7 WAP to find area of Rectangle, circle & square using function overloading

Ans
```
#include <iostream>
using namespace std;

int area(int);
int area(int, int);
float area(float);
float area(float, float);

int main()
{
    int l, b;
    float r;
    cout << "Enter length & breadth of rect : ";
    cin >> l >> b;

    cout << "Enter radius of circle : ";
    cin >> r;

    cout << "\n Area of rectangle is " << area(l,b);
    cout << "\n Area of circle is " << area(r);
}
int area (int l, int b)
{
    return (l*b);
}
    float area (float r)
    { return (3.14 * r * r);
    }
```

Q8 What is a Constructor and destructor? Explain various types of Constructors. Is it mandatory to use constructor in a class?

Ans

Constructors	Destructors
It helps to initialize an object of a class.	It is used to destroy an instance.
It can either accept arguments or not.	It cannot have any arguments.
It is called when an instance or object of a class is created.	It is called when an object of a class is freed or deleted.
It is used to allocate the memory to an instance or object.	While it is used to deallocate the memory of an object of class.
It can be overloaded	It can't be overloaded.
They are called as successive order.	They are called as reverse of order of constructor.

DATE : / /
PAGE NO :

★ 4 types of constructors :-

1. Default Constructors
2. Parameterized Constructors
3. Copy Constructors
4. Dynamic Constructors

Q.9. Describe Operator overloading. List the operators which cannot be overloaded. Create a class to perform matrix addition using operator overloading.

Ans. Operator Overloading is a technique by which operators used in a prog. lang. are implemented in user defined types with customized logic that is based on the types of arguments passed.

Operator overloading facilitates the specification of user defined implementation for operations wherein one or both operands are of user defined class or structure.

Operator overloading is helpful in cases where the operators used for certain types provide semantics related to the domain context.

DATE
PAGE NO

Operators that cannot be overloaded :-

1. Scope Operator ::
2. Size of
3. member selector .
4. member pointer selector *
5. Ternery operator ? :

Q10. What is Friend function? Write a program to swap private data of two different class.

Ans. A friend function is a special fun" in CPP which inspite of not being member fun" of a class has privilege to access private and protected data of a class.

```
#include <iostream>
using namespace std;
class Swap {
    int temp, a, b;
public :
    Swap ( int a , int b)
    {
        this -> a = a;
        this -> b = b;
    }
    friend void swap ( Swap& );
};
```

```
void swap ( Sway & s1)
{
    cout <<"\n Before swapping : "<< s1.a <<
                                " " <<< s1.b;

    s1.temp = s1.a;
    s1.a = s1.b;
    s1.b = s1.temp;
    cout << "\n After swapping : "<< s1.a <<
                                " " <<s1.b;

}
    int main()
    {
        Swap s(4, 6);
        swap (s);
        return 0;
    }
```

Q11. WAP that defines a class 'Money' that store time values as Rupee and Paisa respectively Perform operator M3 = M1 + M2 on its object.

Ans.

```
#include <iostream>
using namespace std;
class Money
{
    long int rs;
    int paisa;
    public:
     Money ()
     {
```

```
        rs = 0;
        paisa = 0;
    }
    void get()
    {
        cout << "\n Enter money in rs : ";
        cin >> rs;
        cout << "\n Enter money in paisa : ";
        cin >> paisa;
    }
    void operator- = (Money &t)
    {
        rs -= t.rs;
        paisa -= t.paisa;
        cout << "\n\n subtracted money rs : " << rs
                                    << " " << paisa;
    }
    void put()
    {
        cout << Rs << " " << paisa << endl;
    }
};
void main()
{
    Money m1, m2;
    clrscr();
    m1.get();
    m2.get();
    cout << "\n\t first money rs : ";
    m1.put();
```

```
    cout << "\n\t second money rs: ";
    m2.put();
    m1 += m2;
    m1 -= m2;
    getch();
}
```

Q12. WAP to add two complex number by overlooding binary plus (+) operator.

Ans.

```
#include <iostream>
using namespace std;
class Complex{
    private:
        int real, imag;
    public:
    Complex(){
    real = imag = 0;
}
  Complex (int r, int i)
{
    real = r;
    imag = i;
}
    string to_string(){
    stringstream ss;
    if (imag >= 0)
      ss << "(" << real << "+" imag << "i");
    else
```

```
    ss << "(" << real << "+" << imag << "i)";
    return ss.str();
}
Complex operator + (complex c2)
{
    Complex ret;
    ret.real = real + c2.real;
    ret.imag = imag + c2.imag;
    return ret;
}};
int main()
{
    Complex c1(8,-5), c2(2,3);
    Complex res = c1 + c2;
    cout << res.to_string();
}
```

Q 13. What is static variable and static function

Ans.

static Variable	Static function
A static variable is a variable that is declared using the keyword static.	It is a member fun that is used to access only static dat members.

DATE : / /
PAGE NO.

8/6. Explain Virtual Base class with suitable C++ code

Ans. Virtual Base classes are used in virtual inheritence in a way of preventing multiple "instances" of a given class appearing in an inheritence hierarchy when using multiple inheritence.

```
#include<iostream>
using namespace std;
class A {
public:
    int num;
    A()
    {
      num = 10;
    }
};
class B : public virtual A {};
class C : public virtual A {};
class D : public B, public C {};

int main()
{
  D object;
  cout << "number =" << object.num << endl;
  return 0;
}
```

Q18. Explain all access specifiers with help of an example.

Ans: (a) **public** : members are accessible from outside the class.

(b) Private : members cannot be accessed from outside the class.

(c) Protected : members cannot be accessed from outside the class, however they can be accessed in inherited classes.

Example :

```
class Myclass {
    public :
        int x ;
    private :
        int y ;
} ;

int main() {
Myclass myObj ;
myObj . x = 25 ;
myObj . y = 50 ;
return 0 ;
}
```

DATE : / /
PAGE NO. :

Q19. Explain scope resolution operator with an example.

Ans. The scope resolution operator (::) is used for several reasons. If the global variable name is as same as local variable name, the scope resolution operator will be used to call the global variable.

Example :

```
#include <iostream>
using namespace std;
char a = 'm';
static int b = 50;

int main()
{
  char a = 's';
  cout << " The static variable : " << ::b;
  cout << "\n the local variable : " << a;
  cout << "\n The global variable : " << ::a;

  return 0;
}
```

Q20 What is a destructor? Explain with example.

Ans Destructors are usually used to deallocate memory and do other cleanup for a class object and its class members when the object is destroyed.

Example

```
#include <iostream>
using namespace std;
class Demo {
    private:
  int num1, num2;
  public:
    Demo (int n1, int n2)
  {
    cout << "Inside constructor" << endl;
        num1 = n1;
        num2 = n2;
    }
      void display ()
  {
      cout << "num1 : " << num1 << endl;
      cout << "num2 : " << num2 << endl;
  }
  ~Demo () {

    cout << "Inside destructor";
  }
};
```

```
int main ()
{
  Demo obj1 (10, 20);
  obj1.display ();
  return 0;
}
```

2. WAP to demonstrate runtime polymorphism.

```
#include<iostream>
using namespace std;
class Animal {
  public:
  void eat(){
    cout <<"Eating ..";
  }
};
class Dog : public Animal
{
  public:
    void eat ()
    {
      cout <<"Eating bread";
    }
};
  int main (void)
{   Dog d = Dog ();
    d.eat();
    return 0;
}
```

Q22 WAP to demonstrate the use of function temp

ans.

```
#include <iostream>
using namespace std;

template <typename T>
T add (T num1, T num2){
    return (num1 + num2);
}

int main(){
    int result1;
    double result2;

    result1 = add<int>(2, 3);
    cout << "2 +3 = " << result << endl;

    result2 = add<double>(2.2, 3.3);
    cout << "2.2 +3.3 = " << result2 << endl;

    return 0;
}
```

Q23. WAP to copy the contents of a file to another file.

Ans.

```
#include <iostream>
using namespace std;
int main()
{
    char ch;
    FILE *source, *target;
    char source_file[] = "x1.txt";
    char target_file[] = "x2.txt";
    source = fopen(source_file, "r");

    if (source == NULL) {
        printf("Press any key to exit..\n");
        exit(EXIT_FAILURE);
    }
    target = fopen(target_file, "w");
    if (target == NULL) {
        fclose(source);
        printf("Press any key to exit..\n");
        exit(EXIT_FAILURE);
    }
    while ((ch = fgetc(source)) != EOF)
        fputc(ch, target);
    printf("file copied successfully\n");
    fclose(source);
    fclose(target);
    return 0;
}
```

DATE / /
PAGE NO.

Q24 WAP to demonstrate the use of class templa

Ans.

```
#include <iostream> template <class T>
class Number {
    private:
     T num;
    public:
     Number(T n) : num(n) {}
    T getNum() {
        return num;
}};
    int main() {
     Number <int> numberInt(7);
     Number <double> numberDouble (7.7);
    cout << "int Number = " << numberInt.getNum()
                                << endl;
    cout << "double number = " << numberDouble.getNum
                                << endl;
    return 0;
}
```

DATE : / /
PAGE NO :

Q25 WAP to demonstrate exception handling.

Ans.

```
#include <iostream>
using namespace std;

int main()
{
    int x = -1;

    cout << "Before try \n";
    try {
        cout << "Inside try \n";
        if (x < 0)
        {
            throw x;
            cout << "After throw \n";
        }
    }
    catch (int x) {

        cout << "Exception caught \n";
    }
    cout << "After catch \n";
    return 0;
}
```