RECURSION

PROGRAMS RELATED TO NUMBERS

Dedicated to all my lovely students. May God help you always.

This guide is a collection of various programs on based on numbers done using the Recursive concept. It also consists of helpful tips and explanations on how to do different types of programs using the recursion concept.

The unique feature of this collection is that along with the recursive codes, we have provided the Iterative codes of the same program as well.

I hope that this will facilitate the students in understanding the concept of recursion well and will be helpful for students preparing for their ISC Computer Science Exams.

- Md. Zeeshan Akhtar

Important Note: In this guide, most of the time, we have just provided the main codes for doing a program by implementing recursion and have not declared the class or written the main() method.
In order to write the complete program, you are required to first declare a class and then write the recursive methods. An example of the code which you need to write before the methods is given below:

```
import java.util.*;
class Sample
  {
    static Scanner sc = new Scanner(System.in);
```

After writing the recursive methods, the main method must be written in order for the complete program to run. The **main()** method contains nothing but a few lines for calling the above created methods.

An example of the main method for the above program is given below:

```
public static void main(String args[])
{
Sample ob = new Sample();
ob.display();
}
```

Factorial of a number

Recursive Method 1

```
int factorial(int n)
 {
  if(n==0)
    return 1;
  else
    return (n*factorial(n-1));
 }
void display()
{
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   int a = factorial(n);
   System.out.print("The factorial of the number = "+a);
}
```

Recursive Method 2

```
int f=1;
int factorial(int n)
 {
  if(n==0)
    return f;
  else   {
     f = f*n;
     return (factorial(n-1));
    }
 }
```

The display() method will be the same as on the left.

Note : If the return type of the function is 'double' as in **double factorial(int n)**, then in method 1 replace 'return 1' with 'return 1.0'. In method 2 replace 'int f=1' with 'double f=1.0'.
Then in the function **void display()** change the line 'int a=factorial(n);' with 'double a=factorial(n);'

Recursive Method 3

```
int f=1;
int factorial(int n, int i)
{
  if(i<=n)   {
     f=f*i;
     return (factorial(n,i+1));
    }
  else
     return f;
}
void display()
{
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   int a = factorial(n,1);
   System.out.print("The factorial of the number = "+a);
}
```

Corresponding Iterative Function

```
int f=1;
int factorial(int n)
 {
  int i=1;
  while(i<=n)    {
       f=f*i;
       i++;
       }
  return f;
 }
void display()
{
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   int a = factorial(n);
   System.out.print("The factorial of the number = "+a);
}
```

Note: The 1st method is the most widely used and the shortest of all the above methods, hence students are advised to use the 1st method while finding factorial of a number using recursion.

Some Programs dealing with Factorial

1. Evaluating the Permutation Function: or $P(n,r) = \frac{n!}{(n-r)!}$

```
import java.util.*;
class Permutation
{
   static Scanner sc = new Scanner(System.in);
   int n,r;
   Permutation()
   {
      n = 0;
      r = 0;
   }
   void readInput()
   {
      System.out.print("Enter the value of n : ");
      n = sc.nextInt();
      System.out.print("Enter the value of r : ");
      r = sc.nextInt();
   }
   int factorial(int n)
   {
      if(n==0)
         return 1;
      else
         return (n*factorial(n-1));
   }
   void compute()
   {
      int a = factorial(n);
      int b = factorial(n-r);
      int res = a/b;
      System.out.println("Answer : "+res);
   }
   public static void main(String args[])
   {
      Permutation ob = new Permutation();
      ob.readInput();
      ob.compute();
   }
}
```

2. Evaluating the Combination Function: or $C(n,r) = \frac{n!}{r!\,(n-r)!}$

```
import java.util.*;
class Combination
{
   static Scanner sc = new Scanner(System.in);
   int n,r;
   Combination()
   {
      n = 0;
      r = 0;
   }
   void readInput()
   {
      System.out.print("Enter the value of n : ");
      n = sc.nextInt();
      System.out.print("Enter the value of r : ");
      r = sc.nextInt();
   }
   int factorial(int n)
   {
      if(n==0)
         return 1;
      else
         return (n*factorial(n-1));
   }
   void compute()
   {
      int a = factorial(n);
      int b = factorial(r);
      int c = factorial(n-r);
      int res = a/(b*c);
      System.out.println("Answer : "+res);
   }
   public static void main(String args[])
   {
      Combination ob = new Combination();
      ob.readInput();
      ob.compute();
   }
}
```

In both the above examples, we are using the recursive function '**int factorial(int n)**' for finding the factorial of a number, and we are calling this function from inside the **compute()** function to find the values of n!, r! and (n-r)! and them applying the given formula to calculate the Permutation or the Combination Function.

3. Checking whether a number is a Special Number (Krishnamurthy Number) or not

```
import java.util.*;
class Krishnamurthy
{
   static Scanner sc = new Scanner(System.in);
   int factorial(int n)
   {
      if(n==0)
      return 1;
      else
      return (n*factorial(n-1));
   }
   void isSpecial(int n)
   {
      int d,s = 0,copy = n;
      while(n>0)     {
         d = n%10;
         s = s+factorial(d);
         n = n/10;
      }
      if(s==copy)
         System.out.println("The number is Special");
      else
         System.out.println("The number is Not Special");
   }
   void display()
   {
      System.out.print("Enter any number : ");
      int n = sc.nextInt();
      isSpecial(n);
   }
   public static void main(String args[])
   {
      Krishnamurthy ob = new Krishnamurthy();
      ob.display();
   }
}
```

Note: In such programs, you may be asked to take the variable 'n' as instance variables. In that case you don't need to take it as parameter to the function **isSpecial()**.

It can also be given that you create a separate function for taking inputs from the user to the variable 'n'. In that case you create that separate function, and write the code for inputting 'n' inside it. If you do this, then you don't need to input 'n' in the **display()** method.

Finding power of a number (a^b)

Recursive Method 1	Recursive Method 2	Corresponding Iterative Code

int power(int a, int b)
{
if(b==0)
return 1;
else
return (a*power(a,b-1));
}
 	
void display()
{
System.out.print("Enter any number : ");
int m = sc.nextInt();
System.out.print("Enter the power : ");
int n = sc.nextInt();
int x = power(m,n);
System.out.print("Result = "+x);
}
	int p=1;
	int power(int a, int b)
	{
	if(b==0)
	return p;
	else
	{
	p = p*a;
	return (power(a,b-1));
	}
	}
	
	The display() method will be the same as on the left.
		int p=1;
		int power(int a, int b)
		{
		int i=1;
		while(i<=b)
		{
		p = p*a;
		i++;
		}
		return p;
		}
		
		The display() method will be the same as on the left.

Note : If the return type of the function is 'double' as in **double power(int a, int b)**, then in method 1 replace 'return 1' with 'return 1.0'. In method 2 replace 'int p=1' with 'double p=1.0'.

Then in the function **void display()** change the line 'int x=power(m,n);' with 'double x=power(m,n);'

Some Programs dealing with Power and Factorial

Note: For solutions to all the questions in this section, the following code will be written. The only change will be there in the function **void calcSeries()**. So, we will be writing only the code for the function **void calcSeries()**.

```
import java.util.*;
class SeriesSum
{
   static Scanner sc = new Scanner(System.in);
   int x,n;
   double sum = 0.0;
   SeriesSum(int xx, int nn)
   {
      x = xx;
      n = nn;
   }
   double factorial(int n)
   {
      if(n==0)
         return 1.0;
      else
         return (n*factorial(n-1));
   }
   double power(int a, int b)
   {
      if(b==0)
         return 1.0;
      else
         return (a*power(a,b-1));
   }
   void calcSeries()
   {
      ...
   }
   void display()
   {
      calcSeries()
      System.out.println("Sum of the series : "+sum);
   }
   public static void main(String args[])
   {
      System.out.print("Enter the value of n : ");
      int n = sc.nextInt();
      System.out.print("Enter the value of x : ");
      int x = sc.nextInt();
      SeriesSum ob = new SeriesSum(x,n);
      ob.display();
   }
}
```

$$S = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

```
void calcSeries()
{
   double num, den;
   for(int i=0;i<n;i++)       {
      num = power(x,i);
      den = factorial(i);
      sum = sum+(num/den);
   }
}
```

$$S = 1 + \frac{x}{2!} + \frac{x^2}{3!} + \frac{x^3}{4!} + \cdots + \frac{x^n}{(n+1)!}$$

```
void calcSeries()
{
   double num, den;
   for(int i=0;i<n;i++)       {
      num = power(x,i);
      den = factorial(i+1);
      sum = sum+(num/den);
   }
}
```

$$S = \frac{x^2}{1!} + \frac{x^4}{3!} + \frac{x^6}{5!} + \cdots + \frac{x^{2n}}{(2n-1)!}$$ **[ISC 2014]**

```
void calcSeries()
{
   double num, den;
   for(int i=1;i<=n;i++)       {
      num = power(x,(2*i));
      den = factorial((2*i) - 1);
      sum = sum+(num/den);
   }
}
```

$$S = x - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \cdots + \frac{x^n}{n!}$$

```
void calcSeries()
{
   double num, den;
   for(int i=1;i<=n;i++)       {
      num = power(x,i);
      den = factorial(i);
      if(i%2==0)
         sum = sum+(num/den);
      else
         sum = sum+(num/den);
   }
}
```

Note: For the below given series, the value of the variable sum will be declared as 1.0 instead of 0.0 as **double sum = 1.0;**

$$S = 1 + \frac{x^2}{3!} + \frac{x^3}{4!} + \cdots + \frac{x^n}{(n+1)!}$$

```
void calcSeries()
  {
    for(int i=2;i<=n;i++)
    {
      num = power(x,i);
      den = factorial(i+1);
      sum = sum+(num/den);
    }
  }
```

$$S = 1 + \frac{x}{1!} + \frac{x^3}{2!} + \frac{x^5}{3!} + \cdots + \frac{x^{2n-1}}{n!}$$ **[ISC 2005]**

```
void calcSeries()
  {
    for(int i=1;i<n;i++)
    {
      num = power(x,(2*i - 1));
      den = factorial(i);
      sum = sum+(num/den);
    }
  }
```

Fibonacci Series

Recursive Code for generating the **nth** term of the Fibonacci Series. This is the most widely used function for generating the Fibonacci series.

```
int fib(int n)
{
if(n<=1)
  return 0;
else if(n==2)
  return 1;
else
  return (fib(n-1)+fib(n-2));
}
```

If we want to print 1st 10 terms, then we will call the above method 10 times as follows:

```
int c;
for(int i=1;i<=10;i++)
{
c = fib(i);
System.out.print(c+"  ");
}
```

In the above code, the variable 'c' is storing the value of the 1st term, then the 2nd term, then the 3rd term till 10th terms. As soon as 'c' gets a value, we are printing it. Hence 10 terms of the Fibonacci series is generated.

Another Recursive Method	Corresponding Iterative Method

`int a=0,b=1,c=0;`
`void fib(int i, int limit)`
`{`
`  if(i<=limit)`
`  {`
`  c = a+b;`
`  System.out.print(c+" ");`
`  a = b;`
`  b = c;`
`  fib(i+1,limit);`
`  }`
`}`
 	
`void display()`
`{`
`  System.out.print("Enter the limit : ");`
`  int limit = sc.nextLine();`
`  System.out.print("The Series is : "+a+" "+b+" ");`
`  fib(3,limit);`
`}`
	`int a=0,b=1,c=0;`
	`void fib(int limit)`
	`{`
	`  int i=3;`
	`  while(i<=limit)`
	`  {`
	`  c = a+b;`
	`  System.out.print(c+" ");`
	`  a = b;`
	`  b = c;`
	`  i++;`
	`  }`
	`}`
	
	`void display()`
	`{`
	`  System.out.print("Enter the limit : ");`
	`  int limit = sc.nextLine();`
	`  System.out.print("The Series is : "+a+" "+b+" ");`
	`  fib(limit);`
	`}`

The above recursive code generates and prints all the Fibonacci series term, from the 3rd term onwards. The 1st and the 2nd term are stored in the variables 'a' and 'b' respectively, while the third term which is being recursively generated is stored in the variable 'c'. Since, the 1st and the 2nd terms are known to be 0 and 1 and hence they are directly printed in the display method. This is why we are sending the starting value as 3.

Students are advised to use the 1st method whenever possible.

Generating the Entire Fibonacci Series (ISC 2005)

```
import java.util.*;
class Recursion
{
   static Scanner sc = new Scanner(System.in);
   int a,b,c,limit;

   Recursion()
   {
      a = 0;
      b = 1;
      c = 0;
      limit = 0;
   }

   void input()
   {
      System.out.print("Enter the limit : ");
      limit = sc.nextInt();
   }

   int fib(int n)
   {
      if(n<=1)
         return a;
      else if(n==2)
         return b;
      else
         return (fib(n-1)+fib(n-2));
   }
   void generate_fibseries()
   {
      System.out.println("The Fibonacci Series is:");
      for(int i=1;i<=limit;i++)
      {
         c = fib(i);
         System.out.print(c+"  ");
      }
   }
   public static void main()throws Exception
   {
      Recursion ob = new Recursion();
      ob.input();
      ob.generate_fibseries();
   }
}
```

Note: All the functions used here are what was given in the question (ISC 2005)

Finding HCF or GCD of 2 numbers

Recursive Method 1	Recursive Method 2	Recursive Method 3	Recursive Method 4

int gcd(int p,int q)
{
if(q==0)
return p;
return gcd(q,p%q);
}
	int gcd(int p,int q)
	{
	if(p%q!=0)
	return gcd(q,p%q);
	else
	return q;
	}
		int gcd(int p,int q)
		{
		if(p==q)
		return p;
		else if(p>q)
		return gcd(p-q,q);
		else
		return gcd(p,q-p);
		}
			int res=1;
			int gcd(int p,int q,int i)
			{
			if(i<=p)
			{
			if(p%i==0&&q%i==0)
			res = i;
			gcd(p,q,i+1);
			}
			return res;
			}

Any of the above 4 methods can be used, for finding the GCD or HCF of any two numbers. We would advise the students to use any of the first three recursive method, whenever possible.

A program on finding HCF of 2 numbers using the recursive technique came in **ISC 2006**.

Finding the LCM of 2 numbers

```
int lcm=1;
int calcLCM(int a,int b)
{
if(lcm%a==0 && lcm%b==0)
   return lcm;
 lcm++;
 calcLCM(a,b);
 return lcm;
}
void display()
{
System.out.print("Enter the 1st number : ");
int x = sc.nextInt();
System.out.print("Enter the 2nd number : ");
int y = sc.nextInt();
findLCM(x,y);
System.out.println("LCM = "+lcm);
}
```

Checking for Prime Number

Recursive Method 1

```
int count=0;
int prime(int n,int i)
{
if(i<=n)  {
   if(n%i==0)
        count++;
return (prime(n,i+1));
 }
else
  return count;
}
void display()  {
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   int res=prime(n,1);
   if(res==2)
      System.out.println("The number is Prime");
   else
      System.out.println("The number is Not Prime");
}
```

Corresponding Iterative Method

```
int count=0,i=1;
int prime(int n)
{
while(i<=n)  {
   if(n%i==0) {
     count++;
     }
   i++;
 }
  return count;
}
void display()  {
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   int res = prime(n);
   if(res==2)
      System.out.println("The number is Prime");
   else
      System.out.println("The number is Not Prime");
}
```

Recursive Method 2 (Without Return Type)

```
int count=0;
void prime(int n,int i)
{
   if(i<=n)   {
      if(n%i==0)
         count++;
      prime(n,i+1);
   }
   else   {
      if(count==2)
         System.out.println("The number is Prime");
      else
         System.out.println("The number is not Prime");
}}
void display()
{
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   prime(n,1);
}
```

Corresponding Iterative Method

```
int count=0,i=1;
void prime(int n)
{
   while(i<=n)   {
      if(n%i==0)      {
         count++;
      }
      i++;
   }
   if(count==2)
      System.out.println("The number is Prime");
   else
      System.out.println("The number is not Prime");
}
void display()
{
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   prime(n);
}
```

Recursive Method 3	Corresponding Iterative Method

int i=2;
int isprime(int n)
{
if(i<=n) {
if(n%i==0) {
i++;
prime(n);
}
}
if(i==n)
return 1;
else
return 0;
}
void display() {
System.out.print("Enter any number : ");
int n = sc.nextInt();
if(isprime(n)==1)
System.out.println("The number is Prime");
else
System.out.println("The number is Not Prime");
}
	int count=0,i=2;
	int isprime(int n)
	{
	while(i<=n)
	{
	if(n%i!=0)
	i++;
	}
	if(i==n)
	return 1;
	else
	return 0;
	}
	void display() {
	System.out.print("Enter any number : ");
	int n = sc.nextInt();
	if(isprime(n)==1)
	System.out.println("The number is Prime");
	else
	System.out.println("The number is Not Prime");
	}

Note: In the above method, 'i' is increasing every time we did not get any factor, and the recursive method/loop will end when 'i' is equal to 'n'. Hence, if 'i' becomes equal to the number, then that means that the number is not divisible by any other number except 1 and itself. Hence it is Prime.

Another Recursive Method (Modified form of above code):

```
int i=2;
int isprime(int n)
  {
    if(i==n)
      return 1;
    else if(n%i==0 || n==1)
      return 0;
    else     {
      i++;
      return isprime(n);
  }}
```

Note: In the above method, 'i' is increasing every time we did not get any factor, and the recursive method/loop will end when 'i' is equal to 'n' or If ‘n’ is divisible by any value of ‘i’ or ‘n’ is equal to 1.

If 'i' becomes equal to the number, then that means that the number is not divisible by any other number except 1 and itself. Hence it is Prime.

If ‘n’ is divisible by any value of ‘i’ apart from 1 and ‘n’ or if ‘n’ is equal to 1, then in both case, the number is not prime.

Checking for Twin prime

```
import java.util.*;
class TwinPrime
{
  int isPrime(int n, int i)   {
    if(i==n)
      return 1;
    else if(n%i==0 || n==1)
      return 0;
    else
      return isPrime(n,i+1);
  }
  public static void main(String args[])
  {
    Scanner sc = new Scanner(System.in);
    TwinPrime ob = new TwinPrime();
    System.out.print("Enter the 1st number : ");
    int x = sc.nextInt();
    System.out.print("Enter the 2nd number : ");
    int y = sc.nextInt();
    int a = ob.isPrime(x,2);
    int b = ob.isPrime(y,2);
    if(a==1 && b==1 && Math.abs((x-y))==2)
      System.out.println(x+" and "+y+" are Twin primes");
    else
      System.out.println("They are not Twin Primes");
  }}
```

Checking for Emirp Number (ISC 2013)

An Emirp number is a number which is prime backwards and forwards. Example: 13 and 31 are both prime numbers. Thus, 13 is an Emirp number.

Design a class Emirp to check if a given number is Emirp number or not. Some of the members of the class are given below:

Class name : Emirp

Data members/instance variables:

n : stores the number

rev : stores the reverse of the number

f : stores the divisor

Member functions:

Emirp(int nn) : to assign n = nn, rev = 0 and f = 2

int isprime(int x) : check if the number is prime using the recursive technique and return 1 if prime otherwise return 0

void isEmirp() : reverse the given number and check if both the original number and the reverse number are prime, by invoking the function isprime(int) and display the result with an appropriate message.

```
import java.util.*;
class Emirp
{
   int n, rev, f;
   Emirp(int nn)   {
      n = nn;
      rev = 0;
      f = 2;
   }
 /* Note: The function isPrime(...) will check whether 'n' is
   prime or not and not 'x'. Variable 'x' is just a counter  */
   int isprime(int x)    {
      if(x==n)
         return 1;
      else if(n%x==0 || n==1)
         return 0;
      else
         return isprime(x+1);
   }
   void isEmirp()    {
       int copy = n, d;
       while(copy>0) // code for reversing a number
       {
          d = copy%10;
          rev = rev*10+d;
          copy = copy/10;
       }
       int a = isprime(f); //Checking original number 'n'
       n = rev; //Saving reverse in 'n'
       f = 2;
       int b = isprime(f); //Checking reverse of number 'n'
       if(a==1 && b==1)
       System.out.println("It is an Emirp Number");
       else
       System.out.println("It is Not an Emirp Number");
   }

   public static void main(String args[])
   {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter any number : ");
      int n = sc.nextInt();
      Emirp ob = new Emirp(n);
      ob.isEmirp();
   }  }
```

Generating Prime Triplets within a Range

The consecutive prime numbers are known as **Prime Triplets** if they satisfy the following conditions: (n, n+2, n+6) are all prime OR (n, n+4, n+6) are all prime, where n is an integer number.

```
import java.util.*;
class PrimeTriplets
{
   int isPrime(int n, int x)    {
      if(x==n)
         return 1;
      else if(n%x==0 || n==1)
         return 0;
      else
         return isPrime(n,x+1);
   }
   public static void main(String args[])
   {
      Scanner sc = new Scanner(System.in);
      PrimeTriplets ob = new PrimeTriplets();
      System.out.print("Enter the start limit : ");
      int p = sc.nextInt();
      System.out.print("Enter the end limit : ");
      int q = sc.nextInt();
      System.out.println("The Prime Triplets are : ");
      int a,b,c,d;
      for(int i=p; i<=q-6; i++)       {
         a = ob.isPrime(i,2);
         b = ob.isPrime((i+2),2);
         c = ob.isPrime((i+4),2);
         d = ob.isPrime((i+6),2);
         if(a==1 && b==1 && d==1)
            System.out.println(i+"\t"+(i+2)+"\t"+(i+6));
          if(a==1 && c==1 && d==1)
            System.out.println(i+"\t"+(i+4)+"\t"+(i+6));
   }}}
```

Extracting Digits from a number and performing any given operation

Operation from First digit onwards	Operation from Last digit onwards

void numOp(int n)
{
if(n>0)
{
int d = n%10;
numOp(n/10);
Write the operation you want to perform with the digits coming out from the beginning over here.
}
}
	void numOp(int n)
	{
	if(n>0)
	{
	int d = n%10;
	Write the operation you want to perform with the digits coming out from the end over here.
	numOp(n/10);
	}
	}

In the 1st code, we are taking out the digits from the end of the number, and without performing any operation with the digits extracted from the last, we are again calling the recursive function with the number reduced by 10 times. We then write the operation we want to perform after the line which is calling the function recursively, because by doing so, we would be using the LIFO property of the stack used in recursion. Digits extracted from the end, were placed in the beginning of the stack, hence, when we pop out the digits, we would be getting the First digit of the number first, then the second and so on. Thus in this case we get digits from the beginning of a number. In the 2nd code above code, we are taking out the digits from the end of the number, and are performing the operation with the digits extracted before calling the recursive function with the number reduced by 10 times. Thus in this case we get digits from the end of a number.	

Some Programs dealing with digits of a number

1. Finding the sum of the digits of a number

Recursive Method 1	Recursive Method 2

int s = 0;
int sumDig(int n)
{
if(n>0)
{
int d = n%10;
sumDig(n/10);
s = s+d;
}
return s;
}
 	
void display()
{
System.out.print("Enter any number : ");
int n = sc.nextInt();
int x = sumDig(n);
System.out.println("Sum of digits = "+x);
}
	int s = 0;
	int sumDig(int n)
	{
	if(n>0)
	{
	int d = n%10;
	s = s+d;
	sumDig(n/10);
	}
	return s;
	}
	
	In the 1st method we are adding digits from the beginning, while in the 2nd method, we are adding the digits from the end.
	
	The difference between the 2 method is of shifting the line **s=s+d;** which in the 1st method is written after the recursive call, while in the 2nd is written before the recursive call.

Important Note:

For finding the sum of the **square of the digits**, write the above code. Just change the line **s=s+d;** into **s=s+d*d;** or **s=s+(int)Math.pow(d,2);**

For finding the sum of the **cube of the digits**, write the above code. Just change the line **s=s+d;** into **s=s+d*d*d;** or **s=s+(int)Math.pow(d,3);**

Another Important Recursive Method (Method 3)

Use this method, when you are not provided with or are asked not to take any separate variable for storing the sum. In the above 2 examples we used a variable 's' as an instance variable for storing sum. in the below given example, we are not using any variable for storing the sum.

```
int sumDig(int n)
 {
  if(n==0)
   return 0;
  else   {
    int d = n%10;
    return (d+sumDig(n/10));
 }}
```

For finding the sum of the **square of the digits**, just change the line **return (d+sumDig(n/10));** into **return (d*d+sumDig(n/10));**

For finding the sum of the **cube of the digits**, just change the line **return (d+sumDig(n/10));** into **return (d*d*d+sumDig(n/10));**

2. Finding whether a number is a Magic Number or not [ISC 2009]

```
import java.util.*;
class Magic
{
   static Scanner sc = new Scanner(System.in);
   int n;
   Magic()   {
      n = 0;
   }
   void getnum()   {
      System.out.print("Enter any number: ");
      n = sc.nextInt();
   }
   int sumDig(int x)   {
      if(x==0)
         return 0;
      else
      {
         int d = x%10;
         return (d+ sumDig(x/10));
      }
   }
   void isMagic()   {
      int a = n;
      while(a>9)
      {
         a=sumDig(a);
      }
      if(a==1)
         System.out.print(n+" is a Magic Number");
      else
         System.out.print(n+" is Not a Magic Number");
   }
   public static void main(String args[])
   {
      Magic ob = new Magic();
      ob.getnum();
      ob.isMagic();
   }
}
```

Note: A Magic Number is a number whose eventual sum of the digits is equal to 1. This addition of digits is performed again and again till the number itself becomes a single digit number. Example, 28
In this program we have used the 3rd method of finding the sum of the digits.

Important Note: In such programs you may be asked to input the number in a separate function like we have used above.

We have taken the value of 'n' as input from the user in the function **getnum()**.

You can also be asked to initialize the variable 'n' inside a function with some parameter passed to that function. In such case you don't input 'n' in that function, but you input it inside main() and pass this input to that initializing function.

Example: In the above program you can write the getnum() method as:

```
void getnum(int num)
{
   n = num;
}
```

Then the main() method will be:

```
public static void main(String args[])
{
   Magic ob = new Magic();
   System.out.print("Enter any number: ");
   int b = Integer.parseInt(br.readLine());
   ob.getnum(b);
   ob.isMagic();
}
```

3. Finding whether a number is a Happy Number or not [ISC 2012]

```
import java.util.*;
class Happy
{
   static Scanner sc = new Scanner(System.in);
   int n;
   Happy()
   {
      n = 0;
   }
   void getnum(int nn)
   {
      n = nn;
   }
   int sum_sq_digits(int x)
   {
      if(x==0)
         return 0;
      else
      {
         int d = x%10;
         return (d*d+ sum_sq_digits(x/10));
      }
   }
   void ishappy()   {
      int a = n;
      while(a>9)
      {
         a = sum_sq_digits(a);
      }
      if(a==1)
         System.out.print(n+" is a Happy Number");
      else
         System.out.print(n+" is Not a Happy Number");
   }
   public static void main(String args[])
   {
      Happy ob = new Happy();
      System.out.print("Enter any number: ");
      int m = sc.nextInt();
      ob.getnum(m);
      ob.ishappy();
   }
}
```

Note: In this program we have used the 3rd method of finding the sum of the square of the digits.
All the functions used here are what was given in the question (Question 10 of ISC 2012)

4. Finding whether a number is an Armstrong Number or not

```
import java.util.*;
class Armstrong
{
   static Scanner sc = new Scanner(System.in);
   int n;
   Armstrong()
   {
      n = 0;
   }
   void readNum()
   {
      System.out.print("Enter any number: ");
      n = sc.nextInt();
   }
   int sumCubeDig(int x)
   {
      if(x==0)
         return 0;
      else
      {
         int d = x%10;
         return (d*d*d+ sumCubeDig(x/10));
      }
   }
   void isArm()
   {
      int a = sumCubeDig(n);
      if(a==n)
        System.out.print(n+" is an Armstrong Number");
      else
        System.out.print(n+" is Not an Armstrong Number");
   }
   public static void main(String args[])
   {
      Armstrong ob = new Armstrong();
      ob.readNum();
      ob.isArm();
   }
}
```

Note: An Armstrong Number is a number whose sum of the cube of the digits is equal to the original number. Example, $153 = 1^3+5^3+3^3$

In this program we have used the 3rd method of finding the sum of the cube of the digits.

5. Finding frequency of the digits of a number

```
int A[]={0,0,0,0,0,0,0,0,0,0};
void count(int n)
{
   if(n>0)    {
      int d = n%10;
      A[d]++;
      count(n/10);
   }
}
void display()  {
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   System.out.println("Digit\t\tFrequency");
   count(n);
   for(int i=0;i<10;i++)    {
      if(A[i]!=0)
      System.out.println(i+"\t\t"+A[i]);
}}
```

Note: In this program, we have taken an integer Array of size=10 for counting the frequency of each digit present in a number. This array has to be declared as an instance variable. Each cell of the array has been initialized to zero.

Cell with index 0 will store the frequency of digit 0, cell with index 1 will store the frequency of digit 1 and so on.

In the recursive function, we are extracting the digits one by one, and incrementing the corresponding cell of the array.

So after the recursive function ends, we have in the array the frequency of each digit. In the **display()** method, we are printing only those cells of the array whose value is not zero.

The "\t" is for giving tab spaces.

6. Printing the digits of a number in words [ISC 2007]

```
import java.util.*;
class Convert
{
   static Scanner sc = new Scanner(System.in);
   int n;
   Convert()    {
      n = 0;
   }
   void inpnum()
   {
      System.out.print("Enter any number: ");
      n = sc.nextInt();
      System.out.print("Output : ");
      extdigit(n);
   }
   void extdigit(int n)
   {
      if(n>0)
      {
         int d = n%10;
         extdigit(n/10);
         num_to_words(d);
      }
   }
   void num_to_words(int x)
   {
      switch(x)
      {
         case 0: System.out.print("Zero "); break;
         case 1: System.out.print("One "); break;
         case 2: System.out.print("Two "); break;
         case 3: System.out.print("Three "); break;
         case 4: System.out.print("Four "); break;
         case 5: System.out.print("Five "); break;
         case 6: System.out.print("Six "); break;
         case 7: System.out.print("Seven "); break;
         case 8: System.out.print("Eight "); break;
         case 9: System.out.print("Nine "); break;
      }
   }
   public static void main(String args[])
   {
      Convert ob = new Convert();
      ob.inpnum();
   }
}
```

Note: All the functions used here are what was given in the question (ISC 2007)

In the above program, we are inputting the number in the **inpnum()** method. After inputting, we are sending this number to the function **extdigit(int n)** which is extracting one digit at a time. Here we have used the 1st recursive method of extracting digits from the beginning. As soon as we are getting the digits from the stack using the LIFO property, we are sending it to the **num_to_words(int x)** method, which is printing the digit in words.

Reversing a Number

(Without Return Type)

Recursive Method 1

```
void rev(int n)
{
   if(n>0)    {
      int d = n%10;
      System.out.print(d);
      rev(n/10);
   }
}
```

Note : This method is printing the digits on screen in reverse order.

Recursive Method 2

```
int r = 0;
void rev(int n)
{
   if(n>0)    {
      int d = n%10;
      r = r*10+d;
      rev(n/10);
   }
   else
   System.out.println("Reverse : "+r);
}
```

Recursive Method 3

```
int r = 0;
int rev(int n)
{
   if(n>0)
   {
      int d = n%10;
      r = r*10+d;
      return (rev(n/10));
   }
   else
      return r;
}
```

Corresponding Iterative Code

```
int r = 0;
int rev(int n)
{
   while(n>0)
   {
      int d = n%10;
      r = r*10+d;
      n = n/10;
   }
   return r;
}
```

```
void display()
{
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   System.out.print("Reverse of the number = ");
   rev(n);
}
```

```
void display()
{
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   int x = rev(n);
   System.out.print("Reverse of the number = "+x);
}
```

Checking for Palindrome Number

```
int r = 0;
int rev(int n)
{
   if(n>0)    {
      int d = n%10;
      r = r*10+d;
      return (rev(n/10));
   }
   else
      return r;
}
```

```
void display()
{
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   int x = rev(n);
   if(x==n)
      System.out.println("The number is Palindrome");
   else
      System.out.println("The number is Not Palindrome");
}
```

Extracting Numbers from a limit upto another limit

Recursive Method

```
void extract(int p, int q)
{
  if(p<=q)
    {
```

Write the operation you want to perform with the numbers coming one by one in 'p'.

```
      extract(p+1,q);
    }
}
```

Corresponding Iterative Function

```
void extract(int p, int q)
{
  while(p<=q)
    {
```

Write the operation you want to perform with the numbers coming one by one in 'p'.

```
      p++;
    }
}
```

Some Programs dealing with numbers within a range

1. Printing all the Even Numbers starting from 'p' till 'q'

Recursive Method

```
void even(int p, int q)
{
  if(p<=q)
    {
     if(p%2==0)
         System.out.println(p);
     even(p+1,q);
    }
}
```

Corresponding Iterative Function

```
void even(int p, int q)
{
  while(p<=q)
    {
     if(p%2==0)
         System.out.println(p);
     p++;
    }
}
```

```
void display()
{
   System.out.print("Enter the lower limit : ");
   int p = sc.nextInt();
   System.out.print("Enter the upper limit : ");
   int q = sc.nextInt();
   System.out.println("The even numbers are :");
   even(p,q);
}
```

2. Finding Sum of all the Even and Odd Numbers separately from 'p' till 'q'

```
int so = 0,se = 0;
void sumOE(int p, int q)
{
   if(p<=q)
   {
      if(p%2==0)
         se = se+p;
      else
         so = so+p;
      sumOE(p+1,q);
   }
}
```

```
void display()
{
   System.out.print("Enter the lower limit : ");
   int p = sc.nextInt();
   System.out.print("Enter the upper limit : ");
   int q = sc.nextInt();
   sumOE(p,q);
   System.out.println("Sum of even numbers = "+se);
   System.out.println("Sum of odd numbers = "+so);
}
```

3. Printing prime numbers from 'p' till 'q'

```
int isPrime(int n, int i)
{
   if(i==n)
      return 1;
   else if(n%i==0 || n==1)
      return 0;
   else
      return isPrime(n,i+1);
}
void printPrime(int p, int q)
{
  if(p<=q)
    {
     if(isPrime(p,2)==1)
         System.out.println(p);
     printPrime(p+1,q);
    }
}
```

```
void display()
{
   System.out.print("Enter the lower limit : ");
   int p = sc.nextInt();
   System.out.print("Enter the upper limit : ");
   int q = sc.nextInt();
   printPrime(p,q);
}
```

Note: Using this technique, you can also check for any other number within a given range. Just pass on the value of 'p' to appropriate function.

Finding the factors of a number

```
void factors(int n, int i)
{
   if(i<=n)   {
      if(n%i==0)
         System.out.print(i+"  ");
      factors(n,i+1);
   }
}
```

```
void display()
{
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   System.out.print("Factors of the number are : ");
   factors(n,1);
}
```

1. Checking for Perfect Number

```
int sum = 0;
int factors(int n, int i) {
   if(i<n)   {
      if(n%i==0)
         sum = sum+i;
      return factors(n,i+1);
   }
   else
      return sum;
}
```

```
void display()
{
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   int f = factors(n,1);
   if(f==n)
   System.out.println("The Number is Perfect");
   else
   System.out.println("The Number is Not Perfect");
}
```

2. Checking for Composite Number

```
int count = 0;
int factors(int n, int i) {
   if(i<n)   {
      if(n%i==0)
         count++;
      return factors(n,i+1);
   }
   else
      return count;
}
```

```
void display()
{
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   int f = factors(n,1);
   if(f>2)
   System.out.println("The Number is Composite");
   else
   System.out.println("The Number is Not Composite");
}
```

Finding the Prime Factors of a number

Recursive Method	Corresponding Iterative Method

void primeFact(int n,int i)
{
if(n>1) {
if(n%i == 0) {
System.out.print(i+" ");
primeFact(n/i,i);
}
else
primeFact(n,i+1);
}}
void display(){
System.out.print("Enter the any number : ");
int n = sc.nextInt();
System.out.print("Prime Factors of the number : ");
primeFact(n,2);
}
	int primeFact(int n)
	{
	int i = 2;
	while(n>1) {
	if(n%i == 0) {
	System.out.print(i+" ");
	n = n/i;
	}
	else
	i++;
	}}
	void display(){
	System.out.print("Enter the any number : ");
	int n = sc.nextInt();
	System.out.print("Prime Factors of the number : ");
	primeFact(n);
	}

Some Programs related to prime factors of a number

1. Finding Sum of the Prime Factors of a number

```
int sum = 0;
int primeFact(int n,int i)
{
   if(n>1)
   {
      if(n%i == 0)
      {
         sum = sum+i;
         return (primeFact(n/i,i));
      }
      else
         return (primeFact(n,i+1));
   }
   else
      return sum;
}
void display()
{
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   int s = primeFact(n,2);
   System.out.print("Sum of prime factors : "+s);
}
```

2. Checking for Smith Number

```
int sum = 0;
int primeFact(int n,int i)
{
   if(n>1)
   {
      if(n%i == 0)
      {
         sum=sum+sumDig(i);
         return (primeFact(n/i,i));
      }
      else
         return (primeFact(n,i+1));
   }
   else
      return sum;
}
int sumDig(int n)
{
   if(n==0)
      return 0;
   else
   {
      int d = n%10;
      return (d+sumDig(n/10));
   }
}
void display()
{
   System.out.print("Enter the any number : ");
   int n = sc.nextInt();
   int sd = sumDig(n);
   int sf = primeFact(n,2);
   if(sd==sf)
      System.out.println("The Number is a Smith Number");
   else
      System.out.println("The Number is Not a Smith
Number");
}
```

Note: The recursive method **sumDig()** is returning us the sum of the digits of a number, while the recursive method **primeFact()** is returning us the sum of the prime factors of a number.

The method **primeFact()** is the same we used for finding the sum of the prime factors above, with the only addition being that we are first sending that prime factor to the **sumDig()** function and then adding it to the sum.

This is done to ensure that we meet with the condition of checking for a Smith Number and hence get the sum of the digits of all those prime factors which are more than one digit.

Conversion Between Number Systems

1. Decimal to Binary Conversion

<table>
<tr><th>Recursive Method 1
Without Return Type</th><th colspan="2">Recursive Method 2
With Return type</th><th>Corresponding Iterative Code</th></tr>
<tr><td>

```
void binary(int n)
{
   if(n>0)
   {
      int d = n%2;
      binary(n/2);
      System.out.print(d);
   }
}
```

</td><td>

```
int bin = 0;
int binary(int n)
{
   if(n>0)
   {
      int d = n%2;
      binary(n/2);
      bin = bin*10+d;
   }
   return bin;
}
```

</td><td>

```
int bin = 0,c = 0;
int binary(int n)
{
   if(n>0)
   {
      int d = n%2;
      bin =
bin+d*(int)Math.pow(10,c++
);
      return binary(n/2);
   }
   else
      return bin;
}
```

</td><td>

```
int bin = 0,c = 0;
int binary(int n)
{
   while(n>0)
   {
      int d = n%2;
      bin =
bin+d*(int)Math.pow(10,c++
);
      n = n/2;
   }
   return bin;
}
```

</td></tr>
<tr><td>

```
void display()
{
   System.out.print("Enter a
number : ");
   int n = sc.nextInt();
   System.out.print("Binary =
");
   binary(n);
}
```

</td><td colspan="3">

```
void display()
{
   System.out.print("Enter a number : ");
   int n=sc.nextInt();
   int x=binary(n);
   System.out.println("Binary = "+x);
}
```

</td></tr>
</table>

Note: In method 1, we have made use of the stack memory to reverse the remainders of the number when divided by 2.

<table>
<tr><th>Recursive Method 3 (Return type String)</th><th>Corresponding Iterative Method</th></tr>
<tr><td>

```
String bin = "";
String binary(int n)
{
   if(n>0)
   {
      int d = n%2;
      bin = (char)(d+48) + bin;
      return binary(n/2);
   }
   else
      return bin;
}

void display()
{
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   String x = binary(n);
   System.out.println("Binary = "+x);
}
```

</td><td>

```
String bin = "";
String binary(int n)
{
   while(n>0)
   {
      int d = n%2;
      bin = (char)(d+48) + bin;
      n = n/2;
   }
   return bin;
}

void display()
{
   System.out.print("Enter any number : ");
   int n = sc.nextInt();
   String x = binary(n);
   System.out.println("Binary = "+x);
}
```

</td></tr>
</table>

Note: In method 3, we are making use of the ASCII values to convert an integer digit to a character digit. We are converting an integer 0 to a character '0' and then adding them to a String.

Decimal to Binary Conversion - Solution to Question 12 of [ISC Theory 2007 (Repeat)]

```
import java.util.*;
class dec_Bin
{
   static Scanner sc = new Scanner(System.in);
   int n,s,i;
   dec_Bin()   {
      n = 0;
      s = 0;
      i = 0;
   }
   void getdata()   {
      System.out.print("Enter a decimal number : ");
      n = sc.nextInt();
   }
   void recursive(int x)   {
      if(x>0)      {
         int d = x%2;
         s = s+d*(int)Math.pow(10,i++);
         recursive(x/2);
      }
   }
   void putdata()   {
      System.out.println("Decimal Number = "+n);
      recursive(n);
      System.out.println("Binary Equivalent = "+s);
   }
   public static void main(String args[])   {
      dec_Bin ob=new dec_Bin();
      ob.getdata();
      ob.putdata();
   }
}
```

Note: All the functions are written as per the question 12 of ISC 2007 Computer Science Paper 1 (Theory).

Evil Number - Solution to Question 1 of [ISC Practical 2016 (Specimen)]

An Evil number is a positive whole number which has even number of 1's in its binary equivalent.

```
int c = 0;
int binary(int n)   {
   if(n>0)      {
      int d = n%2;
      if(d==1)
         c++;
      return binary(n/2);
   }
   return c;
}
void display()
{
   System.out.print("Enter a number = ");
   int n = sc.nextInt();
   int x = binary(n);
   if(x%2==0)
      System.out.println(n+" is an Evil Number.");
   else
      System.out.println(n+" is Not an Evil Number.");
}
```

2. Binary to Decimal Conversion

Recursive Method	Corresponding Iterative Method

int dec = 0,c = 0;
int decimal(long n)
{
  if(n>0) {
  int d = (int)n%10;
  dec = dec+d*(int)Math.pow(2,c++);
  return decimal(n/10);
  }
  else
  return dec;
}
void display(){
  System.out.print("Enter any number : ");
  long n = sc.nextLong();
  int x = decimal(n);
  System.out.println("Decimal Equivalent = "+x);
}
	int dec = 0,c = 0;
	int decimal(long n)
	{
	  while(n>0) {
	  int d = (int)n%10;
	  dec = dec+d*(int)Math.pow(2,c++);
	  n = n/10;
	  }
	  return dec;
	}
	
	void display(){
	  System.out.print("Enter any number : ");
	  long n = sc.nextLong();
	  int x = decimal(n);
	  System.out.println("Decimal Equivalent = "+x);
	}

3. Decimal to Octal Conversion [ISC 2011]

<table>
<tr><th>Recursive Method 1
Without Return Type</th><th colspan="2">Recursive Method 2
With Return type</th><th>Corresponding Iterative Code</th></tr>
<tr><td>

```
void octal(int n)
{
   if(n>0)
   {
      int d = n%8;
      binary(n/8);
      System.out.print(d);
   }
}
```

</td><td>

```
int oct = 0;
int octal(int n)
{
   if(n>0)
   {
      int d = n%8;
      binary(n/8);
      oct = oct*10+d;
   }
   return oct;
}
```

</td><td>

```
int oct = 0,c = 0;
int octal(int n)
{
   if(n>0)   {
      int d = n%8;
      oct =
oct+d*(int)Math.pow(10,c++
);
      return octal(n/8);
   }
   else
      return oct;
}
```

</td><td>

```
int oct = 0,c = 0;
int octal(int n)
{
   while(n>0)
   {
      int d = n%8;
      oct =
oct+d*(int)Math.pow(10,c++
);
      n = n/8;
   }
   return oct;
}
```

</td></tr>
<tr><td>

```
void display(){
   System.out.print("Enter a
number : ");
   int n = sc.nextInt();
   System.out.print("Octal =
");
   octal(n);
}
```

</td><td colspan="3">

```
void display()
{
   System.out.print("Enter a number : ");
   int n=sc.nextInt();
   int x=octal(n);
   System.out.println("Octal = "+x);
}
```

</td></tr>
</table>

Note: The above recursive methods having return types, and parameters can also be written without return types and parameters. In such a case, you need to take the decimal number 'n' as an instance variable. Example

```
int oct = 0;
void octal(){
   if(n>0)   {
      int d = n%8;
      n = n/8;
      octal();
      oct = oct*10+d;
   }
}
```

Then you can print the value of the variable 'oct' inside any function.

4. Octal to Decimal Conversion

Recursive Method	Corresponding Iterative Method

`int dec = 0,c = 0;`
`int decimal(int n)`
`{`
`  if(n>0)  {`
`  int d = n%10;`
`  dec = dec+d*(int)Math.pow(8,c++);`
`  return decimal(n/10);`
`  }`
`  else`
`  return dec;`
`}`
`void display()`
`{`
`  System.out.print("Enter any number : ");`
`  int n = sc.nextInt();`
`  int x = decimal(n);`
`  System.out.println("Decimal Equivalent = "+x);`
`}`
	`int dec = 0,c = 0;`
	`int decimal(int n)`
	`{`
	`  while(n>0)  {`
	`  int d = n%10;`
	`  dec = dec+d*(int)Math.pow(8,c++);`
	`  n = n/10;`
	`  }`
	`  return dec;`
	`}`
	`void display()`
	`{`
	`  System.out.print("Enter any number : ");`
	`  int n = sc.nextInt();`
	`  int x = decimal(n);`
	`  System.out.println("Decimal Equivalent = "+x);`
	`}`

5. Decimal to Hexadecimal Conversion

Recursive Method 1 Without Return Type	Recursive Method 2 With Return type	Corresponding Iterative Code

void hexa(int n)
{
if(n>0)
{
int d=n%16;
hexa(n/16);
if(d>=0 && d<=9)
System.out.print((char)(d+48));
else
System.out.print((char)(d+55));
}
}
	String hex="";
	String hexa(int n)
	{
	if(n>0)
	{
	int d=n%16;
	if(d>=0 && d<=9)
	hex = (char)(d+48) + hex;
	else
	hex = (char)(d+55) + hex;
	return hexa(n/16);
	}
	else
	return hex;
	}
		String hex="";
		String hexa(int n)
		{
		while(n>0) {
		int d=n%16;
		if(d>=0 && d<=9)
		hex = (char)(d+48) + hex;
		else
		hex = (char)(d+55) + hex;
		n = n/16;
		}
		return hex;
		}

void display()
{
System.out.print("Enter any number : ");
int n = sc.nextInt();
System.out.print("Hexadecimal Equivalent = ");
hexa(n);
}
	void display()
	{
	System.out.print("Enter any number : ");
	int n = sc.nextInt();
	String h = hexa(n);
	System.out.print("Hexadecimal Equivalent = "+h);
	}	

Note: In the above methods, we are making use of the ASCII values to convert an integer remainder to a character remainder. We are converting remainder 0 to a character '0', a remainder 10 to character 'A' and adding them to a String.

6. Hexadecimal to Decimal Conversion

Recursive Method	Corresponding Iterative Method

int dec = 0,c = 0;
int decimal(String n,int i)
{
if(i>=0)
{
char ch=n.charAt(i);
if(ch>='0' && ch<='9')
dec = dec + (ch-48)*(int)Math.pow(16,c++);
if(ch>='A' && ch<='F')
dec = dec + (ch-55)*(int)Math.pow(16,c++);
return decimal(n,i-1);
}
else
return dec;
}
void display()
{
System.out.print("Enter any hexadecimal number : ");
String n=sc.next();
n = n.toUpperCase();
int len=n.length();
int x=decimal(n,len-1);
System.out.println("Decimal Equivalent = "+x);
}
	int dec = 0,c = 0;
	int decimal(String n,int i)
	{
	while(i>=0)
	{
	char ch=n.charAt(i);
	if(ch>='0' && ch<='9')
	dec = dec + (ch-48)*(int)Math.pow(16,c++);
	if(ch>='A' && ch<='F')
	dec = dec + (ch-55)*(int)Math.pow(16,c++);
	i--;
	}
	return dec;
	}
	void display()
	{
	System.out.print("Enter any hexadecimal number : ");
	String n=sc.next();
	n = n.toUpperCase();
	int len=n.length();
	int x=decimal(n,len-1);
	System.out.println("Decimal Equivalent = "+x);
	}

Some operations based on Arrays

1. Linear Search

Assuming that there is an array A[ ], the below function searches for a value and returns '1' if the search is successful, otherwise returns '-1'.

```
int linearSearch(int i, int search)
{
   if(i==A.length) // condition if the search is unsuccessful
      return -1;
   else if(search==A[i]) // if the search is successful
      return 1;
   else
      return linearSearch(i+1,search);
}
void display()
{
   System.out.print("Enter a number to search : ");
   int v = sc.nextInt();
   int f = ob.linearSearch(0,v);
   if(f == 1)
      System.out.println("Number found");
   else
      System.out.println("Number Not found");
}
```

2. Binary Search [ISC 2015]

Assuming that there is an array A[ ] sorted in ascending order, the below function searches for a value and returns '1' if the search is successful, otherwise returns '-1'.

```
int binSearch(int l, int u, int search)
{
      int mid = (l + u)/2;
      if(u < l) // condition if the search is unsuccessful
         return -1;
      else if(search==A[mid]) // if the search is successful
         return 1;
      else if(search>A[mid])
         return binSearch(mid+1,u,v);
      else
         return binSearch(l,mid-1,v);
}
void display()
{
   System.out.print("Enter a number to search : ");
   int v = sc.nextInt();
   int f = ob.linearSearch(0, A.length-1, v);
   if(f == 1)
      System.out.println("Number found");
   else
      System.out.println("Number Not found");
}
```

3. Selection Sort (in Ascending Order)

```
void selectionSort(int i)
{
   if(i<A.length-1)
   {
      for(int j=i+1; j<A.length; j++)
         {
            if(A[i]>A[j]) //for descending change to A[i]<A[j]
            {
               int temp = A[i];
               A[i] = A[j];
               A[j] = temp;
         }}
      selectionSort(i+1);
}}
void display()
{
   selection(0);
   for(int i=0; i<A.length;i++)    {
      System.out.println(A[i]);
   }
}
```

4. Bubble Sort (in Ascending Order)

```
void bubbleSort(int i)
{
   if(i<A.length-1)
   {
      for(int j=0; j<A.length-1-i; j++)
         {
            if(A[j]>A[j+1]) //for descending A[j]<A[j+1]
            {
               int temp = A[j];
               A[j] = A[j+1];
               A[j+1] = temp;
         }}
      bubbleSort(i+1);
}}
void display()
{
   bubbleSort(0);
   for(int i=0; i<A.length;i++)    {
      System.out.println(A[i]);
   }
}
```