ESc101 : Fundamental of Computing

I Semester 2008-09

Lecture 17+18

- **Structured programming using methods**
- **Common Steps in Solving a problem**

Syntax of a method

return_type **method_name**(*input*)

{

Body

}

if there is no input parameter, we leave the parenthesis blank.

Methods offer a structured (top-down) way of writing the programs

- To solve a problem, divide it into smaller subproblem.
- Divide each smaller sub-problem into further smaller sub-sub-problems until they become easy to solve (code should be 10-20 lines).
- finally for each problem, sub-problems,sub-sub-problems, write a separate method.

Advantages of structured programming

- It is `an easier way to design solution` of the problem.
- It is `easy to code` since each method will be a small piece of code.
- The program becomes `more manageable`.
- It `reduces the chances of logical errors`.

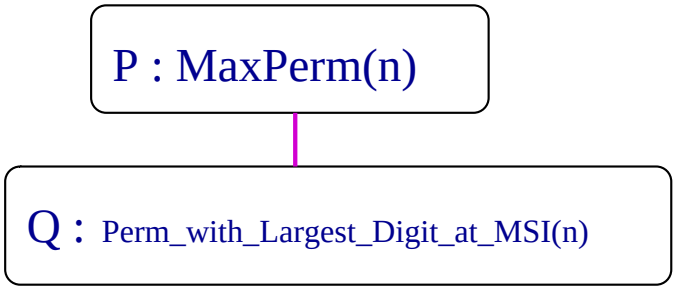
Two Examples of Structured Programming

- Printing a Diamond with a given half-width
- Computing the *Maximum value permutation* for an integer

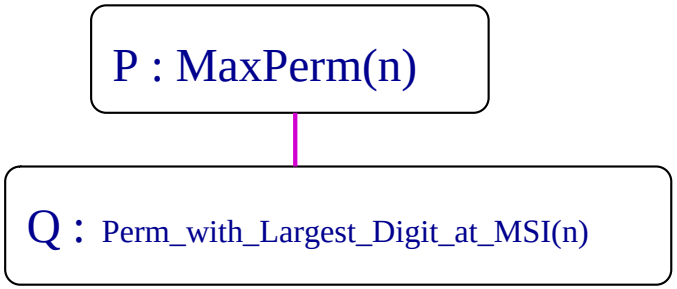
P : Computing Max_Perm for an integer

P : MaxPerm(n)

P : Computing Max_Perm for an integer



P : Computing Max_Perm for an integer



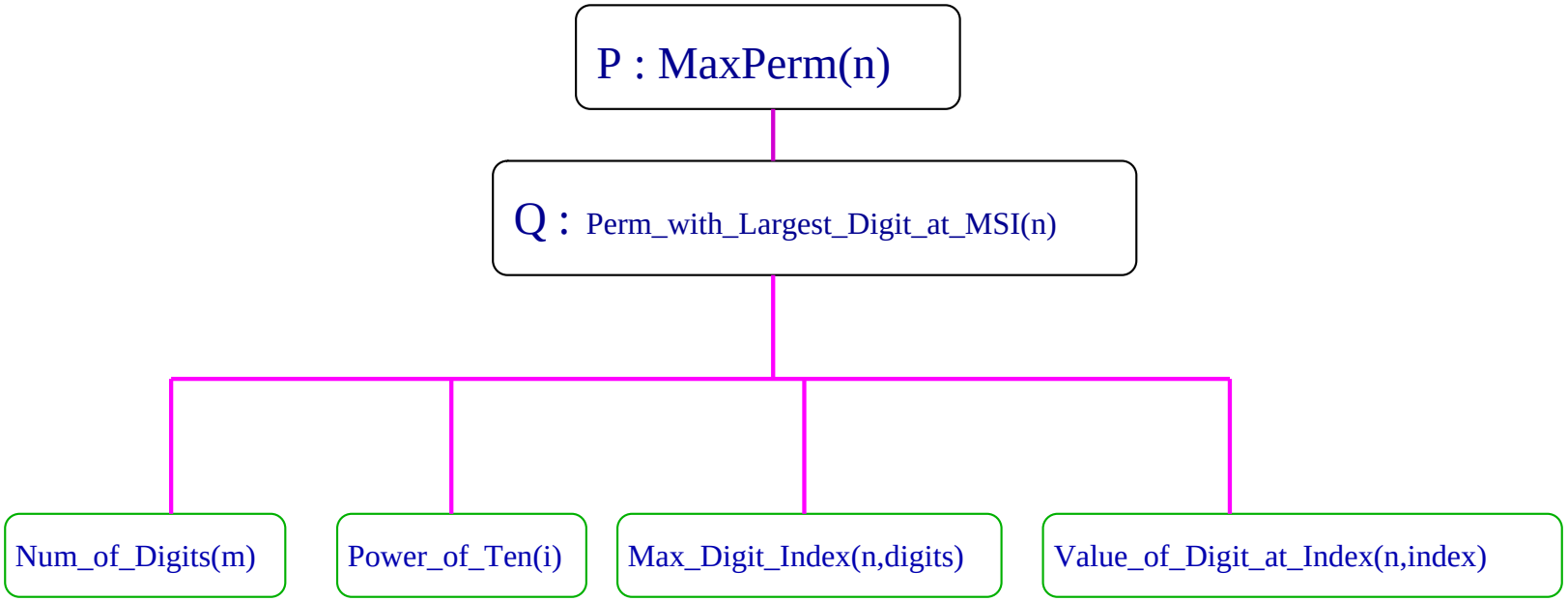
Num_of_Digits(m)

Power_of_Ten(i)

Max_Digit_Index(n,digits)

Value_of_Digit_at_Index(n,index)

P : Computing Max_Perm for an integer



The method Num_of_digits(int n)

```
// function for computing number of digits in n
    public static int Num_of_Digits(int n)
    {
        int digits = 0;
        while(n>0)
        {
            digits = digits + 1;
            n = n/10;
        }
        return digits;
    }
```

The method Power_of_Ten(int i)

```
// function for computing 10^i.
public static int Power_of_Ten(int i)
{
    int power = 1;
    while(i>0)
    {
        power = power*10;
        i = i-1;
    }
    return power;
}
```

The method for Max_Digit

```
public static int Max\_Digit\_Index(int n, int digits)
{    int max_digit = n%10;
     int max_digit_index=1;
     n = n/10;
     for(int i=2; i<=digits; i = i+1)
     {
         int current_digit = n%10;
         if(current_digit>max_digit)
         {
             max_digit = current_digit;
             max_digit_index = i;
         }
         n = n/10;
    }
}
```

The method : Value_of_Digit_of_n_at_Index()

```
// function for computing the digit of n at 'index'
    public static int Value_of_Digit_of_n_at_Index(int n, int index)
    {
        int power = Power_of_Ten(index-1);
        n = n/power;
        return(n%10);
    }
```

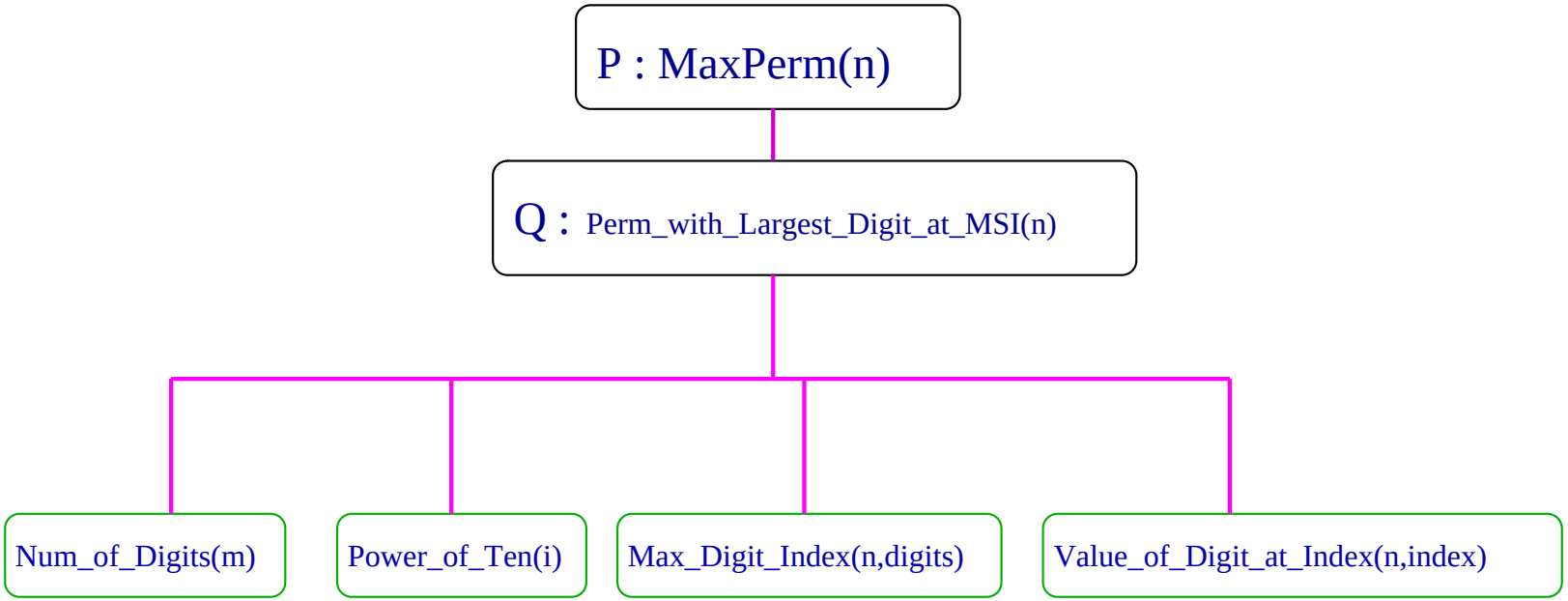
The method : Permutation_with_Max_Digit_at_MSI

```
public static int Permutation_with_Max_Digit_at_MSI(int n, int digits)
{
    int power  = Power_of_Ten(digits-1);
    int digit_at_MSI = n/power;
    int max_digit_index = Max_Digit_Index(n,digits);
    int value_of_max_digit =  Value_of_Digit_of_n_at_Index(n,max_digit_index);
    int new_n=n;
    if(max_digit_index != digits)
    {
      int difference_of_digits = value_of_max_digit - digit_at_MSI;
      int lower_power = Power_of_Ten(max_digit_index-1);
      new_n = n + (power-lower_power)*difference_of_digits;
    }
    return new_n;
}
```

The method Max_Perm(int n)

```
public static int Max_Perm(int n)
{   int upper_n;
    int lower_n;
    int digits = Num_of_Digits(n);
    int power= Power_of_Ten(digits);
    for(int i = digits; i>1; i =i-1)
    {   upper_n= n/power;
        lower_n= n%power;
        lower_n= Permutation_with_Max_Digit_at_MSI(lower_n,i);
        n = upper_n*power+ lower_n;
        power = power/10;
    }
    return n;
}
```

P : Computing Max_Perm for an integer



II : Common Steps in Problem Solving

1. Understanding the problem statement carefully
2. Working on simple examples
3. Working on related problems which are easier
4. Develop insight into the problem using steps 2 and 3
5. Solving the problem

An interesting example : Clever Prisoner problem

There are n prisoners in Jail.

Each prisoner is allocated fixed place in a line right in the beginning. Each day the following event happens. The warden of the jail is a tyrant.Each day the warden asks all the prisoners to line in front him and mth (Here m$>$m) person is executed. This continues till there is only one prisoner left. The last one remaining is pardoned and set free. Now we have a clever prisoner among n prisoners. He selects the position on the first day such that he escapes after n-1 days from the prison. `What position did he select ?` Let us denote the solution as Clever_Prisoner_Position(n,m)

An easier problem

```
// The following function computes the position of the prisoner
// who will be sentenced to life term when there are n prisoners
// in the row and the selection-number is m
public static int The_Unlucky_Position(int m, int n)
{

    ???


}
```

An easier problem

```
// The following function computes the position of the prisoner
// who will be sentenced to life term when there are n prisoners
// in the row and the selection-number is m
public static int The_Unlucky_Position(int m, int n)
{
   if(m%n==0)
       return n;
   else
       return m%n;
}
```

A simple instance : what is Clever_Prisoner_Position(2,m) ?

if m is odd, then it is 2, otherwise it is 1.

Extending to the case : n=3 for any m

Let Clever_Prisoner_Position(2,m) = `i`.

What is Clever_Prisoner_Position(3,m) ?

The solution lies in the following key idea :

> The clever prisoner should occupy that position such that after the execution of one (out of three) prisoner, his position becomes `i`

Extending to the case : n=3 for any m

Let Clever_Prisoner_Position(2,m) = i.

What should be Clever_Prisoner_Position(3,m) ?

Based on the key idea of previous slide there are two cases :

1. Unlucky_Position(3,m)$>$i : ???
2. Unlucky_Position(3,m)$<=$i : ???

Extending to the case : n=3 for any m

Let Clever_Prisoner_Position(2,m) = i.

What should be Clever_Prisoner_Position(3,m) ?

Based on the key idea of previous slide there are two cases :

1. Unlucky_Position(3,m)$>$i : Clever_Prisoner_Position(3,m)=i
2. Unlucky_Position(3,m)$<=$i : Clever_Prisoner_Position(3,m)=i+1

Solving the problem in full generality

Let Clever_Prisoner_Position(n,m) = i.

There are two cases :

1. Unlucky_Position(n+1,m)$>$i :

   Clever_Prisoner_Position(n+1,m)=i

2. Unlucky_Position(n+1,m)$<=$i :

   Clever_Prisoner_Position(n+1,m)=i+1

`Solve the problem bottom up using a for loop (the complete code is available on the course webpage, enjoy ...)`