SCSA1104 - Problem Solving Techniques with C and C++

UNIT – II

Algorithms: Reversing the digits of a number - Generation of Fibonacci sequence- Factorial Computation.

1. Reversing the digits of a number

Problem:

Design an algorithm that accepts a positive integer and reverses the order of its digits

Algorithm Development:

Input – 27953

Output – 35972

Example

27953

To get number 2795 do integer division

i.e.27953 div 10 - > 2795

To get remainder we can do modulo division

i.e 27953 mod 10 ->3

Therefore if we apply the following two steps

r = n mod 10   r=3

n = n div10   n=2795

We can obtain reverse by first extracting the 3, multiplying it by 10, and then adding 5 to give 35

3*10+5 - >35

Input 953 -> reversed number is 359

35*10+9 ->359

359*10+7 - > 3597

Step	Iteration	Value of dreverse

3597*10+2 -> 35972

1	dreverse= dreverse*10+3	3
2	dreverse= dreverse*10+5	35
3	dreverse= dreverse*10+9	359

That is,

dreverse= previous value of dreverse*10+most recently extracted rightmost digit

Logic for reversing the digits

- ✓ Reverse = reverse*10 +n mod 10;
- ✓ n =n div 10

Application

- Hashing
- Information Retreival
- Data base applications

//Program for reversing the digits of a given number

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int n, rev = 0, remainder;
   printf("Enter an integer: ");
   scanf("%d", &n);
   while (n != 0) {
       remainder = n % 10;

       rev = rev * 10 + remainder;

       n /= 10;

   }

   printf("Reversed number = %d", rev);

   return 0;

}
```

2. Generation of Fibonacci sequence

Problem:

Generate and print the first n terms of the Fibonacci sequence.
The first few terms are:

0, 1, 1, 2, 3, 5, 8, 13, ….

Each tem beyond the first two is derived from the sum of its two nearest predecessors.

Definition:

The Fibonacci sequence is a sequence where the next term is the sum of the previous two terms.

Algorithm Development:

From the definition

New term = preceding term + term before preceding term

Using this formula we can generate consecutive terms iteratively.

Let us define

a <- as the term before preceding term

b <- as the preceding term

c <- new term

Then to start with we have:
a = 0 <- First Fibonacci number

b = 1 <- Second Fibonacci number

c = a+b <- third Fibonacci number

To find Fourth Fibonacci number

1.New term (i.e third) assumes the role of the preceding term

2. Current preceding term must assume the role of the term before the preceding term.

That is

Step 1 : a = 0 -> term before preceding term

Step 2 : b = 1 -> preceding term

Step 3 : c = a+b -> new term

Step 4 : a = b -> term before preceding term becomes preceding term

Step 5 : b = c -> preceding term becomes new term

After making this assignment we are in a position where we can use the definition to generate the next Fibonacci number. A way to do this is to loop back to step [3] so this can be placed in a loop to iteratively generate Fibonacci numbers (for n>2). The essential mechanism we could use is:

```
a = 0;
b = 1
for( i=2;i<n;i++)
{
c = a+b;
a= b;
b = c
}
```

We can make some improvements in this algorithm. as each new term c is computed, the term before the preceding term, a, loses its relevance to the calculation of the next Fibonacci number. To restore its relevance we made the assignment.

We know that at all times only two numbers are relevant to the generation of the next Fibonacci number. In our computation, however, we have introduced a third variable, c. What we can therefore attempt to do is keep the two variables a and b always relevant. Because the first Fibonacci number becomes irrelevant as soon as the third Fibonacci is computed we can start with:

To make a relevant

```
        a : = 0;          [1]
```

```
          b : = 1;             [2]
          a : = a+b;            [3]   (this keeps a relevant to generate  the next Fibonacci number)
```

If we iterate on step [3] to generate successive Fibonacci numbers we run into trouble because we are not changing b. However, after step [3] we know the next (i.e. fourth) Fibonacci number can be generated using

```
          next: = a + b   [4] (fourth Fibonacci number)
```

When we do step [4], the old value of b, loses its relevance.

To make b relevant

b = a + b

The first four steps then become

```
a= 0;
b = 1;
a = a+b;  // this keeps a relevant to generate the next Fibonacci number
b = a+b;  // to keep b relevant
```

We find that a and b as defined are correct for generating the fifth Fibonacci number. Working through several more steps confirms that we can safely iterate on steps [3] and [4] to generate the required sequence.

Example

//Program to generate fibo series without third variable

```
#include <stdio.h>
#include <stdlib.h>
int main()
{

 int n1=0,n2=1,n3,i,number;
 printf("Enter the number of elements:");
 scanf("%d",&number);
 printf("\n%d\n%d\n",n1,n2);//printing 0 and 1
 for(i=2;i<number;i++)

 //loop starts from 2 because 0 and 1 are already printed
 {
  n3=n1+n2;
  printf("%d\n",n3);
  n1=n2;
  n2=n3;
 }
     return 0;
}
```

Try to write algorithm and pesudocode yourself

//Program to generate fibo series without third variable

```
#include<stdio.h>

void main()

{

Int  a=0,b=1,i=2,n;

Printf("Enter value of n");

Scanf("%d",&n);

printf("\n%d\n%d\n",n1,n2);

   while(i<number)

   {

      //printf("\n%d\n%d\n",n1,n2);

      n2=n1+n2;

      n1=n2-n1;

      i=i+1;

      printf("%d",n3);

}

}
```

3. **Factorial Computation**

Given Problem:

Given a number n, compute n factorial where n>=0

Algorithm Development:

Definition: Factorial of a positive integer n is product of all values from n to 1. For example, the factorial of 3 is (3 * 2 * 1 = 6).

We are given that

Eg. n=5! = 1*2*3*4*5

n! = 1*2*3*4…………*(n-1)*n for n>=1

and by definition

0! = 1

Applying the factorial definition we get

0! = 1

1! = 1*1

2! = 1*2

3! = 1*2*3

4! = 1*2*3*4

5! =1*2*3*4*5

We see that 5! contains all the factors of 4!. The only difference is the inclusion of the number 5. We can generalize this by observing that n! can always be obtained from (n-1)! By simply multiplying it by n.

n! = n*(n-1)! for n>=1

using this definition we can write the first few factorials as

1! = 1*0!

2! = 2*1!

3! = 3*2!

4! = 4*3!

If we start with f = 0! = 1 we can rewrite the first few steps in computing n! as

We know that 0! = 1 = f

f = 1	0!
f = f*1	1!
f = f*2	2!
f = f*3	3!
f = f*4	4!

From step (2) onwards we are actually repeating the same process over and over. For the general (i+1)th step we have,

f = f*i   1<i<n

This general step can be placed in a loop to iteratively generate n!.

Algorithm :

```
Step 1-> start
Step 2-> Declare the variables fact, i, n
Step 3->initialize fact = 1,i=1
Step 4->Read the value of n from user.
Step 5->Repeat step 4 to 6 until i<=n
Step 6-> fact = fact*i
Step 7-> i =i+1
Step 8-> Print fact
Step 9-> stop
```

Program:

```
#include<stdio.h>
#include<conio.h>   // preprocessor section
Void main()              // main function
```

```
{
  int n,i,fact=1;
printf(“Enter the value of n”);
scanf(“%d”,&n);
for(i=1;i<=n;i++)
{
fact = fact*I;
}
printf(“%d”,fact);
}
```