Birla Institute of Technology & Science, Pilani
Computer Programming (CSF111)
Second Semester 2015-2016
Lab-5

Objectives

1. Iterative constructs

Iterative Constructs (Loops)

There may be a situation when you need to execute a block of code several number of times. In general statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general from of a loop statement in most of the programming languages:

- **for** statement,
- **while** statement and
- **do-while** statement

A loop can either be a pre-test loop or be a post-test loop. In a pre-test loop the condition is checked before the beginning of each iteration. If the test expression evaluates to true the statement associated with the pre-test loop construct are executed and process is repeated until the test expression is true. On the other hand, if the test expression evaluates to false, the statements associated with the construct are skipped and the statement next to the loop is executed.

In the post-test loop, the code is always executed once. At the completion of the loop code, the expression is tested. If the test expression evaluates to true, the loop repeats; if the expression is false the loop terminates.

Difference between pre-test and post-test loop

	Pre-test loop	Post-test loop
Initialization	Once	Once
Number of tests	N+1	N
Action executed	N	N
Updating executed	N	N
Minimum iteration	Not even once	At least once

Event control loop/sentinel value / indefinite loop

An event changes the test expression of the loop from true to false. A sentinel value is a value that is not legitimate data value for a particular problem, but it is proper type, that is used to check for a 'stopping value' It is like a flag or an indicator. For example we take the input from user till user presses special character like 999 or 'y'.

Counter control loop / finite loop

When the number of repetitions is known, then a counter controlled loop is used. For example find the sum of ten natural numbers.

C provides three loops construct namely **for**, **while** and **do while**. For and while are pre-test loop and do-while is post-test loop.

Syntax of for loop is as follows:

*for **(initE; termE; updateE)***
*{ block of statements executed if **termE** is true; }*

Syntax of while loop is as follows:

initE
while(termE)
*{ block of statements executed if **termE** is true;*
updateE
}

Syntax of do while loop is as follows:

initE
do

{ block of statements executed;
updateE
} while(termE);

- ***initE*** is evaluated once before any iterations of the loop. Its value is not used - therefore it must assign one or more variables.
- ***termE*** must be an expression that evaluates to a boolean value and is used as termination check for the loop. It is evaluated once before every iteration; if the value is true then that iteration is executed; if the value is false, then no more iteration are executed.
- ***updateE*** is evaluated once at the end of every iteration. Its value is not used - therefore it must assign one or more variables.

Assignments

In the following problems flowcharts are depicted in order to better understand how to write a program that involves loop(s).

Assignment 1
Approximate the value of pi using the following series expansion.

$pi = 4 – 4/3 + 4/5 – 4/7 + … + (-1)^{n-1}\ 4/(2n+1) + …$

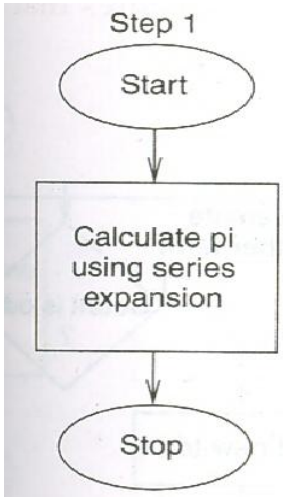

Fig 1(a)

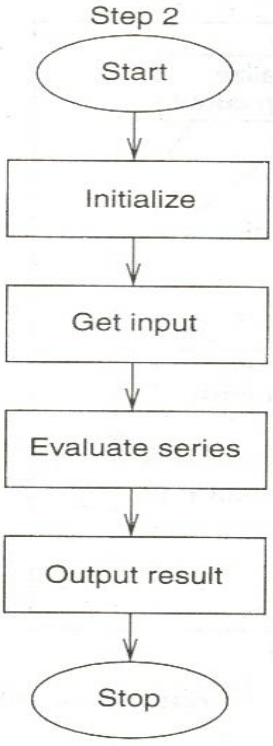

Fig 1(b)

Fig 1(a) shows the flowchart at a very high level. Fig1(b) expands with initialization, input and output. As we move over the flowcharts, its advised you start writing the necessary C statements for your program.

After the *Get input* step in Fig 1(b), in order to evaluate the series, use the refined subtask *Evaluate Series* as below that iterates a given number of times. Within this loop evaluate the terms for series expansion of pi.

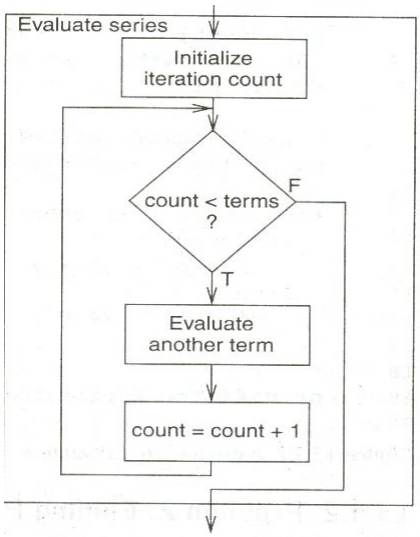

Fig 1(c)

Incorporate the current term based on whether it is odd or not based on Fig 1(d)

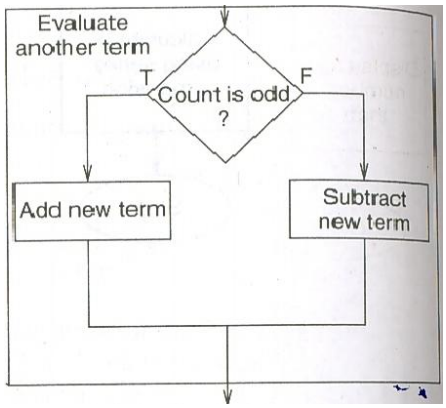

Fig 1(d)

```
#include <stdio.h>
int main()
{
int count, numOfTerms;
double pi = 0;
printf("\n Number of terms (must be 1 or larger):");
scanf("%d", &numOfTerms);

 for(count = 1; count <= numOfTerms; count++) {
    if(count % 2 == 0)
       pi += (4.0 / (2.0 * count -1) ) // Odd term
    else
       pi -= (4.0 / (2.0 * count -1) ) //  Even term
 }
printf("\n The approximate value of pi is %f", pi);
}
```

NOTE: Do not copy the above program. Rather, understand it and then do it of your own. Also, convert the above program using while loop.

Assignment 2
Find all prime numbers that are less than 100. Recall that a number is prime only if the only numbers that evenly divide it are 1 and itself.

First *decompose* the problem to compute the prime numbers less than 100. Fig 2(a) shows the first three steps that involve creating a loop that iterates between the 2 and 100.

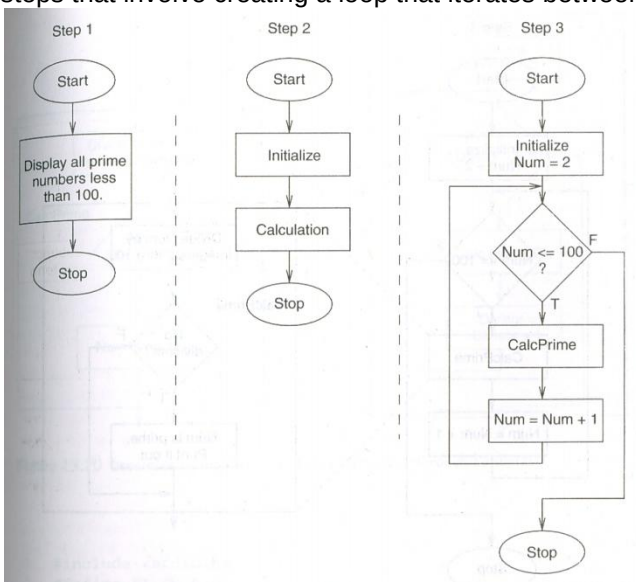

Fig 2 (a)

Next decompose the CalcPrime in step 3 as shown in Fig 2 (b). Basically we will determine if each number is divisible by an integer between 2 and 100 (be careful to exclude the number itself).

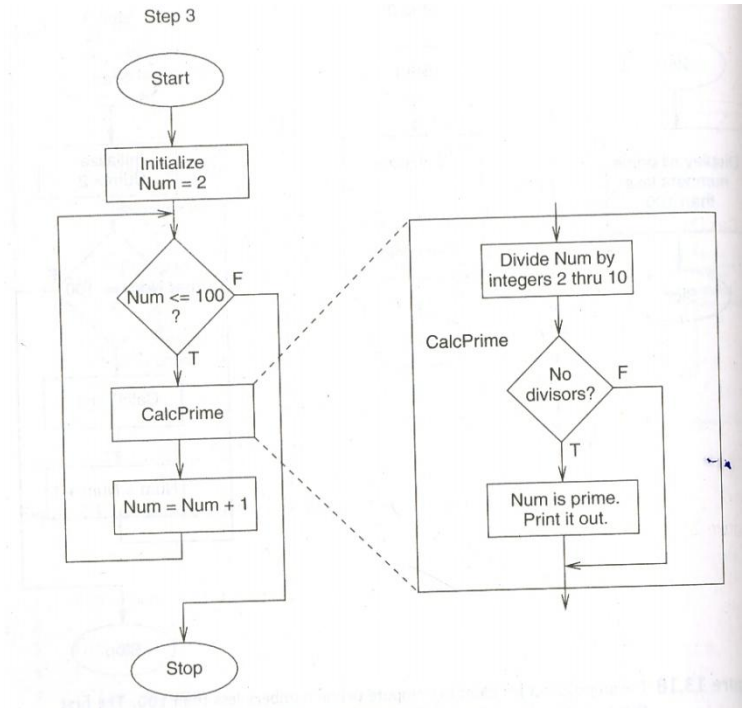

Fig 2(b)

Finally we need to refine the *Divide number by integers 2 through 100* subtask. A simple way is to use a counter controlled loop to cycle through all integers between 2 and 100 as shown in Fig 2(c).

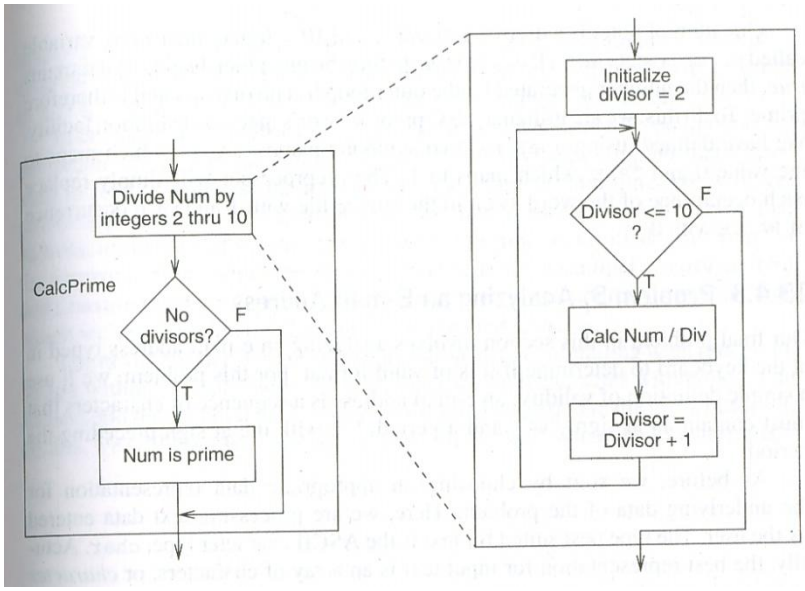

Fig 2(c)

```
#include <stdio.h>
# define FALSE 0
# define TRUE 1
int main()
{
  int num, divisor, prime;

  /* Start at 2 and go until 100 */
  for( num = 2; num <=100; num++) {
   prime = TRUE; // Assume the number is prime

      /* Test if the candidate number is prime */
      for( divisor = 2; divisor <=100; divisor++)
        if( ((num%divisor) == 0) && num != divisor)
           prime = FALSE;

      if(prime)
       printf("\n The number %d is prime", num);
   }

}
```

NOTE: Do not copy the above program. Rather, understand it and then do it of your own. Also, convert the above program using while loop.

Assignment 3

Write a program to find whether a positive integer entered by the user is a palindrome or not. E.g. 12321 is a palindrome whereas 112233 is not. Use the approach followed in the previous problems..

Additional problems

Write a program to print the average of first ten natural numbers.

```
#include<stdio.h>
int main()
{
int count,number=1,sum=0;
float average;
for(count=10;count>0;count--)  // reverse loop
{sum = sum+number;
number++;
}
average = sum/10;  // can we write count or number here ??
printf("\n average is %f  \t sum = %d \t count = %d \n", average,sum,count);
}
```

Write a program that prints an average of N integer numbers. Ask the user how many number he wants to enter? (Using for loop).

```
#include<stdio.h>
int main()
{
int count,N;
int number=1;
int sum=0;
float average;
printf("enter N\n");
scanf("%d",&N);
for(count=1;count<=N;count++)
{
printf("enter number \n");
scanf("%d",&number);
sum=sum+number;
}
average = sum/N;
printf("\n average is %f\n", average);
}
```

Write a program that prints the average of N float numbers. Ask the user how many number he/she wants to enter. (Using while loop)

```
#include<stdio.h>
int main()
{
int count,N;
float average,number,sum;
count=1;
sum=0.0;
printf("Enter N \n");
scanf("%d",&N);
while(count<=N)
{
  printf("enter number\n");
  scanf("%f",&number);
  sum+=number;
  count++;
}
average = sum / N;
printf("sum =%f \t count =%d \t average =%f",sum,count,average);
}
```

Write the same program using do-while construct.

```
#include<stdio.h>
int main()
{
int count,N;
```

```
float average,number,sum;
count=1;
sum=0.0;
printf("Enter N \n");
scanf("%d",&N);
do
{
  printf("enter number\n");
  scanf(" %f",&number);
  sum+=number;
  count++;
}while(count<N);
average = sum / N;
printf("sum =%f \t count =%d \t average =%f",sum,count,average);
}
```

Let us write the same program using sentinel value where sentinel value is -1.

```
#include<stdio.h>
int main()
{
int count;
float average,number,sum;
count=0;
number=sum=0.0;
do
{
  printf("enter number\n");
  scanf("%f",&number);
  if(number != -1)
  {
    sum+=number;
    count++;
  }
}while(number != -1);
average = sum /count ;
printf("sum =%f \t count =%d \t average =%f",sum,count,average);
}
```

The Nested for Loop: A nested loop refers to a loop that is contained within another loop.

Syntax: Nested for loop:

```
for(initialize;condition;increment)  // outer loop
{
  for(initialize;condition;increment) // inner loop
  {
    Body of the loop;
  }
}
```

Write a program to generate tables from 2 to 5 with first 20 terms.

```
#include<stdio.h>
int main()
{
  for( int i=1;i<=20;i++)
  {
     int j=2;
     while(j<=5)
     {
        printf("\t %d",j*i);
        j++;
     }
```

```
    printf("\n");
  }
return 0;
}
```

Ex1. Write a program that reads an integer and finds the sum of digits of the number.
Ex2. Write a program to convert a decimal number into corresponding binary equivalent.
Ex3. Write a program to compute and output the first ***N*** terms of the Fi*bonacci* series, whose first 8 terms are as follows: **0 1 1 2 3 5 8 13.** ***N*** will be an input value.
[**Hint:** Each term is dependent on two previous terms. **End of Hint**.]
Ex4. Write a program to calculate the sum of the following series up to ***N*** terms

x - (x^2/sqrt(2)) + (x^3/sqrt(3))-(x^4/sqrt(4)) ...

N and ***x*** will be input values. Use ***math*** library for ***sqrt*** function. Note that ***sqrt*** takes a ***double*** argument and returns a ***double*** value. To obtain better precision define ***x*** as a ***double*** and obtain the sum as a ***double*** value as well. Also note that the ***math*** library has to be linked explicitly.
Ex5. Write a program to find the N prime numbers.
Ex6. Write a program to print the following pyramid.

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

```
5 5 5 5 5
4 4 4 4
3 3 3
2 2
1
```

```
    0
   111
  22222
 3333333
444444444
```
