Engr 101 **Sept. 6, 2019**
Day 7

How many line of print do each of the following produce? The variables i, j, and k are integers.

```
i = 52;
while(i > 5)
 {printf("%d", i);
  i--;
 }
```

```
for(j=-1;j<3;j++)
  {printf("%d", j);
   for(k=1;k<5;k++)
      printf("%d", k);
  }
```

do-while Statement

- For conditions where we know that a loop must execute <u>at least one time</u>

1. Get a *data value*
2. If *data value* isn't in the acceptable range, go back to step 1.

do-while Syntax

```
do
      statement;
while (loop repetition condition);
```

Do while example:
Write a function to prompt the user to enter a positive even number from the keyboard.

```
#include <stdio.h>
int GetEven();
int main()
{
	int even;
	even = GetEven();
	printf("%d", even);
	return(0);
}
int GetEven()
{
	int status;
	int even;
	do
	{
		printf("Enter a positive even integer... ");
		status = scanf_s("%d", &even);
	}while(status > 0 && even % 2 != 0);
	return even;
}
```

Notes:
1. Do while must execute statements *at least* once since the loop condition is not checked until the end of the loop.
2. The scanf function returns the number of characters that was read and stored. If an error occurs or end-of-file is reached before any items could be read, it will return EOF. For example if you try to read a char with the %d format specifier the status will come back as 0.
3. The while statement at the end of a do-while loop ends in a semicolon. The normal while loop does not.

Iterative Approximations

Iterative approximations use the computer to effectively guess at results until it gets an answer within a suitable range of accuracy.

This program examines a function looking for real roots between START and END.

Roots are found when there is a zero crossing.

We start with a large increment of 0.1. When we find a zero crossing we back up, decrease the increment by 10 and continue searching until the increment gets to be less than the tolerance.

```c
#include<stdio.h>
#include<math.h>
#define START -10
#define END 10
double FindY(double x);
int main()
{
      double tolerance, x, xIncr;
      double left, right;
      tolerance = 0.001;
       x = START;
      while(x < END)
      {
            x = x + tolerance;
            xIncr = 0.1;
            left = FindY(x);
            right = FindY(x + xIncr);
            while(xIncr >= tolerance && x < END)
               {while(left*right > 0 && x < END)
                     {x = x + xIncr;
                        left = FindY(x);
                        right = FindY(x + xIncr);
                        }
                    x = x - xIncr;
                    xIncr = xIncr/10;
                    left = FindY(x);
                    right = FindY(x + xIncr);
                    }
            x = x + xIncr*10;
            if(x < END)
               printf("Root is between %10.6f and %10.6f\n", x, x+tolerance);
      }
}
double FindY(double x)
{
      return pow(x, 4) - 7*pow(x, 3) + 9*x*x + 7*x -10;
}
```

Numerical integration
In numerical integration using rectangles we break a function up into small rectangles and add up the area of each rectangle to get a value for the integral over the range.

```
#include "stdio.h"
#include <math.h>

const double INCR = 0.01;
double F(double x);
int main()
  {double newX, oldX;
   double sumRec = 0, sumTrap = 0;
   newX = INCR;
   while (newX <= 10)
     {sumRec = sumRec + INCR*F(newX);
      newX = newX + INCR;
     }
   printf("The true value of the integral is...%10.6f \n ", 14356.66667);
   printf("The integral for rectangular integration is... %10.6f\n ",
sumRec);
   //
       return 0;
  }
//
double F(double x)
  {return pow(x, 4) - pow(x, 3) -10*x*x +3*x + 4;
  }
```