To:
Tenouk

C LAB WORKSHEET 7a
Another C & C++ Repetition Construct: while Loop and do-while Loop 1

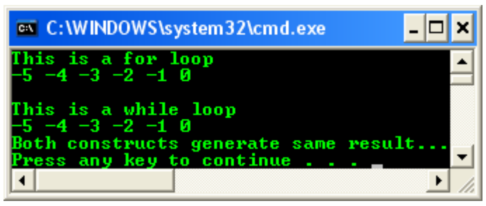
Items in this page:

1. A repetition - the while loop.
2. A repetition - the do-while loop.
3. The for loop control activities, questions and answers.
4. The related tutorial reference for this worksheet are: C/C++ program control 1 and C/C++ program control 2.

The while Loop

- The while loop in C/C++ is very similar to the for loop. The for statement contains two semicolons, which allows placement of the initialization statement, the negation of the termination condition and the iterative statement in it. However, the while loop allows placement of only the condition so the other two statements must be placed outside the while statement.

```
#include <stdio.h>

int main(void)
{
     int i, j;
     // for loop
     printf("This is a for loop\n");
     for(i = -5; i <= 0; i = i +1) // initial, terminal condition and iteration
          printf("%d ", i);
     printf("\n");
     printf("\nThis is a while loop\n");
     j = -5; // initial condition
     // while loop
     while(j <= 0) // terminal condition
     {
          printf("%d ", j);
          j = j + 1;  // iteration
     }
     printf("\nBoth constructs generate same result...\n");
     return 0;
}
```



```
This is a for loop
-5 -4 -3 -2 -1 0

This is a while loop
-5 -4 -3 -2 -1 0
Both constructs generate same result...
Press any key to continue . . .
```

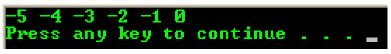
- The for loop in the program prints out all the integers from -5 to 0. The while loop in the second part of the program also does the same thing. Notice that the statements i = -5; and i = i + 1; are removed from the for statement because we converted it to the while loop. Since we now have two statements in the while loop, we needed to add a pair of braces, as we had to with the for loop. Just as the for statement doesn't take a semicolon after it, neither does the while statement. The only difference between a while loop and a for loop construct are a couple of semicolons as shown below.

```
while ( condition )

for( ; condition ; )
```

- In more complete comparison, we may have:

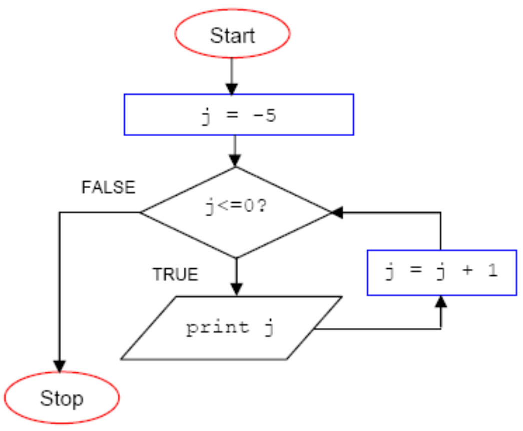
```
initial condition
while ( terminal condition )
   iteration

for(initial condition ; terminal condition ; iteration)

or

initial condition
for( ; terminal condition; )
   iteration
```

- Create a flowchart and/or tracechart for the following while loop.



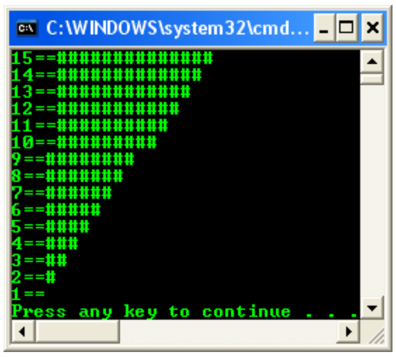
```
-5 -4 -3 -2 -1 0
Press any key to continue . . .
```

```
#include <stdio.h>

int main(void)
{
     int j;
     j = -5;
     // while loop
     while(j <= 0)
     {
            printf("%d ", j);
            j = j + 1;
        }
        return 0;
    }
```



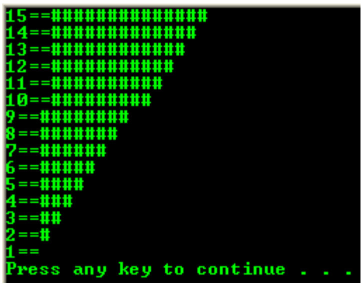
Convert the following programs that using for loop to while loop.

```
// a program to show the nested for loops
#include <stdio.h>

int main()
{
      // variables for counter…
      int i, j;
      // outer loop, execute this first...
      // for every i iteration, execute the inner loop
      for(i=15; i>0;)
      {
            // display i==
            printf("%d==", i);
            // then, execute inner loop with loop index j,
            // the initial value of j is i - 1
            for(j=i-1; j>0; )
            {
                  // display #
                  printf("#");
                  // decrement j by 1 until j>10, i.e j = 9
                  j = j - 1;
            }
               // go to new line, new row
               printf("\n");
               // decrement i by 1, repeat until i > 0 that is i = 1
               i = i - 1;
      }
      return 0;
}
```



```
15==##############
14==#############
13==############
12==###########
11==##########
10==#########
9==########
8==#######
7==######
6==#####
5==####
4==###
3==##
2==#
1==
Press any key to continue . . .
```

```
// program to show the nested while loops
#include <stdio.h>

int main()
{
   // variables for counter…
   int i = 15, j;
   // outer loop, execute this first...
   // for every i iteration, execute the inner loop
   while(i>0)
   {
      // display i==
      printf("%d==", i);
      // then, execute inner loop with loop index j,
      // the initial value of j is i - 1
      j=i-1;
      while(j>0)
      {
         // display #
         printf("#");
         // decrement j by 1 until j>10, i.e j = 9
         j = j - 1;
      }
      // go to new line, new row
      printf("\n");
      // decrement i by 1, repeat until i > 0 that is i = 1
      i = i - 1;
   }
   return 0;
}
```

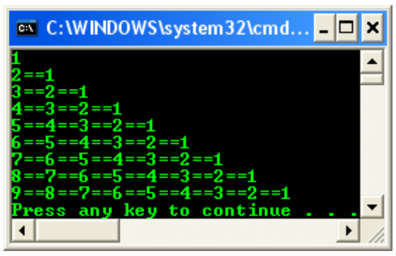


```
15==##############
14==#############
13==############
12==###########
11==##########
10==#########
9==########
8==#######
7==######
6==#####
5==####
4==###
3==##
2==#
1==
Press any key to continue . . .
```

```
// a program to show the nested for loops
#include <stdio.h>

int main()
{
      // variables for counter…
      int i, j;
      // outer loop, execute this first...for every i iteration,
      // execute the inner loop
      for(i = 1; i <= 9;)
      {
            // display i
            printf("%d", i);
            // then, execute inner loop with loop index j,
            // the initial value of j is i - 1
            for(j = i-1; j>0; )
            {
                  // display ==j
                  printf("==%d", j);
                  // decrement j by 1 until j>0, i.e j = 1
                  j = j - 1;
            }
               // go to new line
               printf("\n");
```

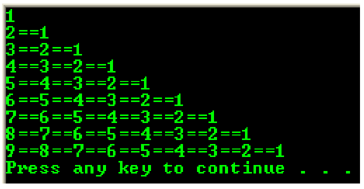
```
// a program to show the nested while loops
#include <stdio.h>

int main()
{
   // variables for counter…
   int i = 1, j;
   // outer loop, execute this first...
   // for every i iteration, execute the inner loop
   while(i <= 9)
   {
      // display i
      printf("%d", i);
      // then, execute inner loop with loop index j,
      // the initial value of j is i - 1
      j = i-1;
      while( j>0 )
      {
         // display ==j
         printf("==%d", j);
         // decrement j by 1 until j>0, i.e j = 1
         j = j - 1;
       }
      // go to new line
```

```
                // increment i by 1, repeat until i<=9, i.e i = 9
                i = i + 1;
        }
        return 0;
}
```



```
1
2==1
3==2==1
4==3==2==1
5==4==3==2==1
6==5==4==3==2==1
7==6==5==4==3==2==1
8==7==6==5==4==3==2==1
9==8==7==6==5==4==3==2==1
Press any key to continue . . .
```

```
        printf("\n");
        // increment i by 1, repeat until i<=9, i.e i = 9
        i = i + 1;
    }
    return 0;
}
```

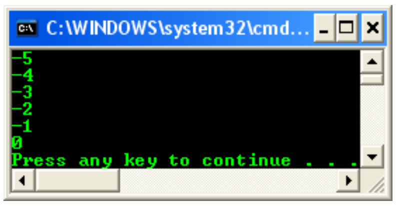


```
1
2==1
3==2==1
4==3==2==1
5==4==3==2==1
6==5==4==3==2==1
7==6==5==4==3==2==1
8==7==6==5==4==3==2==1
9==8==7==6==5==4==3==2==1
Press any key to continue . . .
```

The do-while Loop

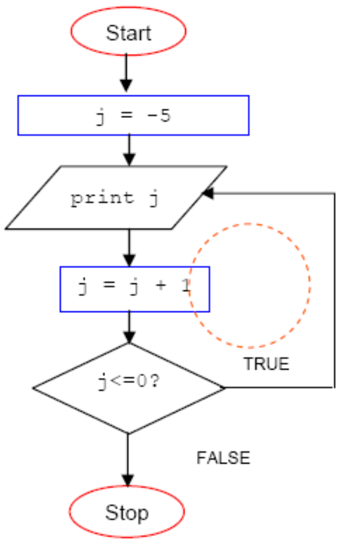
The following example uses do-while loop, another C/C++ construct that can be used for repetition. The main difference here is the **condition is tested after the body of the loop** and the **statement in the body will be executed at least once whether the condition is true or false**. This is not the case with the other two loops where if the condition is false at the beginning, then the body of the loop is not executed at all. Notice the semicolon at the end of the while line of code.

```
// a program to show the use of do-while
#include <stdio.h>

int main()
{
     int j = -5; // initialization
     do
     {
          printf("%d\n", j);
          j = j + 1;
     }
     while(j <= 0);  // condition
     return 0;
}
```



```
-5
-4
-3
-2
-1
0
Press any key to continue . . .
```

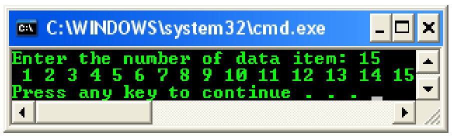
Create a flowchart and/or tracechart for the previous do-while loop.



More for Loop Variation

When combining scanf() with loops, there are two types of loops that should be mastered. It is **count loop** and the **delimiter loop**. With the count loop, the programmer knows before the loop begins how many iterations that loop will perform. However with the delimiter loop, the programmer doesn't know that thing, instead, when a certain data item is encountered, the loop will stop. The certain data item that will terminate the loop is called the **delimiter** for example x != 0. Here is the general form of the count loop.

```
// a program to show the use count
#include <stdio.h>

int main()
{
     int i, count;
     printf("Enter the number of data item: ");
     scanf_s("%d", &count, 1);
     for(i = 1; i <= count; i = i + 1)
          printf(" %d", i);
     printf("\n");
      return 0;
}
```



```
Enter the number of data item: 15
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Press any key to continue . . .
```

Notice that firstly the programmer finds out the number of data items to be read. That number is read and stored in a variable called count. Then the loop is set up so that it will be performed that many times. The loop could have also been set up this way:

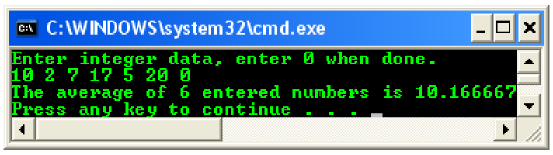
```
for( ; count != 0; count = count – 1)
```

It would have the same effect. However if for some reason the value of count were necessary after the loop, it would not be available as it would be in the first instance. The general form of the delimiter loop is shown in the following code snippet.

```
printf("Enter integer data, 0 when done.\n");
scanf_s("%d", &in_data);
for( ; in_data != 0; )
{
    // other code for the data processing...
    scanf("%d", &in_data);
}
```

The working program example for the delimiter is shown below.

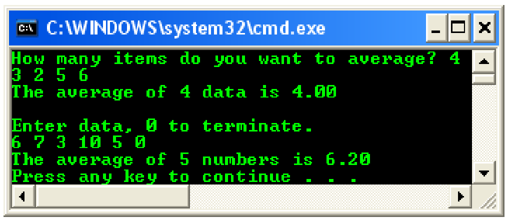
```
// for loop, delimiter
#include <stdio.h>

int main()
{
    float sum;
    int count, in_data;
    printf("Enter integer data, enter 0 when done.\n");
    scanf_s("%d", &in_data, 1);
    for(count = 0, sum = 0.0; in_data != 0; count= count + 1)
    {
        // calculate the sum...
        sum = sum + in_data;
        // read and save the next data...
        scanf_s("%d", &in_data, 1);
        // increment count by 1 and check the
        // in_data != 0 or not, if != 0, repeat
        // else stop the loop and calculate the average...
    }
    printf("The average of %d entered numbers is %f\n", count, sum/count);
   return 0;
}
```



```
Enter integer data, enter 0 when done.
10 2 7 17 5 20 0
The average of 6 entered numbers is 10.166667
Press any key to continue . . .
```

The following is a program example that having both count and delimiter loops.

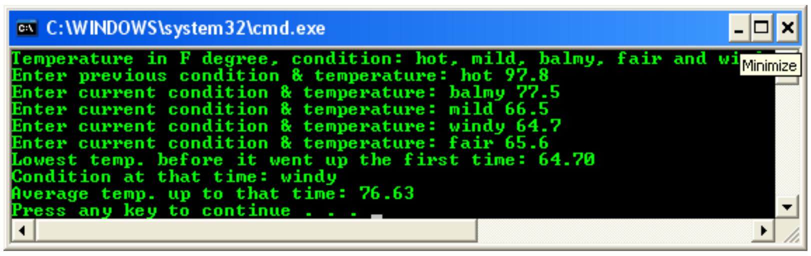
```
// for loop, delimiter
#include <stdio.h>

int main()
{
    float sum = 0.0;
    int i, count, in_data;
    printf("How many items do you want to average? ");
    scanf_s("%d", &count, 1);
    for(i=1; i <= count; i = i+1)
    {
        scanf_s("%d", &in_data);
        sum = sum + in_data;
    }
    printf("The average of %d data is %.2f\n", count, sum / count);
    // redo the previous code
    printf("\nEnter data, 0 to terminate.\n");
    scanf_s("%d", &in_data);
    for(count = 0, sum = 0.0; in_data != 0; count = count + 1)
    {
        sum = sum + in_data;
        scanf_s("%d", &in_data);
    }
    printf("The average of %d numbers is %.2f\n", count, sum/count);
    // we have to make sure count doesn't stay 0 to avoid divide by 0.
   return 0;
}
```



```
How many items do you want to average? 4
3 2 5 6
The average of 4 data is 4.00

Enter data, 0 to terminate.
6 7 3 10 5 0
The average of 5 numbers is 6.20
Press any key to continue . . .
```

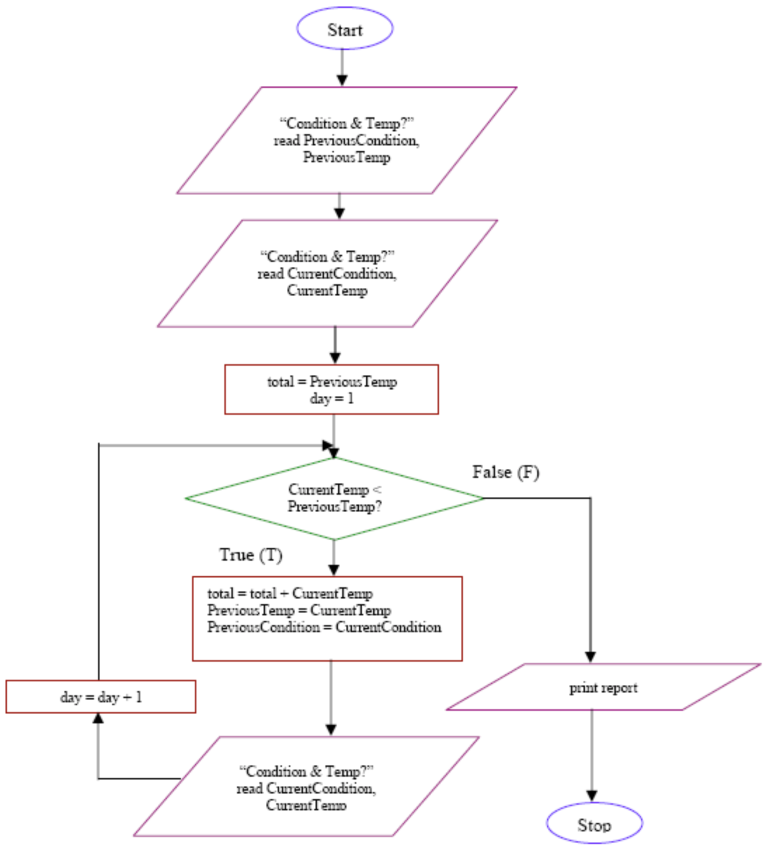
The following is more complex example. The temperature was reported as going down each day, so we want to know the first day that it went up. The temperature and the weather condition for each day are what the program has to read and it has to report the temperature and the weather condition for the day when the temperature dipped to a minimum before it went up.

```
// for loop, delimiter and count
#include <stdio.h>
#include <string.h>

int main()
{
     float CurrentTemp, PreviousTemp, total = 0.0;
     char CurrentCondition[15], PreviousCondition[15];
     int day;
     // initialize the loop, set up the current & previous values
     printf("Temperature in F degree, condition: hot, mild, balmy, fair and windy\n");
     printf("Enter previous condition & temperature: ");
     scanf_s("%s %f", &PreviousCondition, 15, &PreviousTemp);
     printf("Enter current condition & temperature: ");
     scanf_s("%s %f", &CurrentCondition, 15, &CurrentTemp);
     total = PreviousTemp;
     // do the looping
     for(day = 1; CurrentTemp < PreviousTemp; day = day + 1)
     {
          total = total + CurrentTemp;
          // now make the current values become the old or previous ones
          PreviousTemp = CurrentTemp;
          strcpy_s(PreviousCondition, 15, CurrentCondition);
          // and get the new or current values.
          printf("Enter current condition & temperature: ");
          scanf_s("%s %f", &CurrentCondition, 15, &CurrentTemp);
     }
     // print the report
     printf("Lowest temp. before it went up the first time: %.2f\n", PreviousTemp);
     printf("Condition at that time: %s\n", PreviousCondition);
     printf("Average temp. up to that time: %.2f\n", total/day);
   return 0;
}
```



```
Temperature in F degree, condition: hot, mild, balmy, fair and wi
Enter previous condition & temperature: hot 97.8
Enter current condition & temperature: balmy 77.5
Enter current condition & temperature: mild 66.5
Enter current condition & temperature: windy 64.7
Enter current condition & temperature: fair 65.6
Lowest temp. before it went up the first time: 64.70
Condition at that time: windy
Average temp. up to that time: 76.63
Press any key to continue . . .
```

A flowchart for the previous program is given below.


Start
"Condition & Temp?" read PreviousCondition, PreviousTemp
"Condition & Temp?" read CurrentCondition, CurrentTemp
total = PreviousTemp
day = 1
CurrentTemp < PreviousTemp?
False (F)
True (T)
total = total + CurrentTemp
PreviousTemp = CurrentTemp
PreviousCondition = CurrentCondition
day = day + 1
print report
"Condition & Temp?" read CurrentCondition, CurrentTemp
Stop

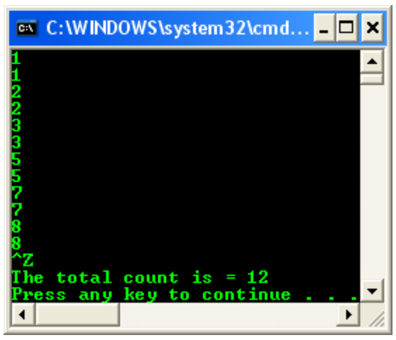
The following is the tracechart.

PreviousCondition	PreviousTemp	CurrentCondition	CurrentTemp	total	True/False	day
hot	97.8					
		balmy	77.5			
				97.8		1
					True	
				175.3		
balmy	77.5					
		mild	66.5			
						2
					True	
				241.8		
mild	66.5					
		windy	64.7			
						3
					True	
				306.5		
windy	64.7					
		fair	65.6			
						4
					False	

The EOF Character

A special character called **EOF** (**E**nd **O**f **F**ile) is placed at the end of a file. When a computer system is typing out a file and it encounters this character, it realizes that this is the end of the file and that the typing of the file is complete. In UNIX based OS, this character is **CTRL-D** and in PC system, it is **CTRL-Z**. EOF is defined in stdio.h file. Its value is -1. The following shows the usage of EOF. The getchar() function get a character from the user and store it in i. If i is not equal to EOF, then we use the putchar() function, print it on the screen and get another one to place it in i. This is done until EOF is encountered, when the loop stops. Notice that the character is read into i, an integer. It could also have been a char. In the output, a total of 4 is shown because the fourth character, which we cannot see, is the carriage return character.

```
#include <stdio.h>

int main()
{
      int i, j;
      // read the user input, getchar() can read any char
      // here we declare int
      i = getchar();
      for(j = 0; i != EOF; j = j + 1)
      {
            // display to screen...
            putchar(i);
            i = getchar();
      }
      printf("The total count is = %d\n", j);
      return 0;
}
```



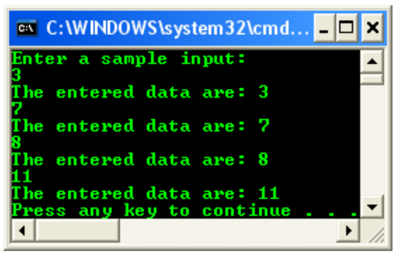
```
1
1
2
2
3
3
5
5
7
7
8
8
^Z
The total count is = 12
Press any key to continue . . .
```

More Activities And Questions

Try the following program example, show the output and answer the questions. The inputs sample when the program allows you to do so is also shown or stated. Enter the input one at a time and may not all on one line. A program may not need all the values shown here.

Sample input: 3 7 8 11 14 10 9 9 6

```
#include <stdio.h>

int main()
{
   int i, k;
   printf("Enter a sample input:\n");
   for(i = 1; i <= 4; i = i + 1)
   {
     scanf_s("%d", &k, 1);
     printf("The entered data are: %d \n", k);
   }
   return 0;
}
```

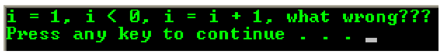


```
Enter a sample input:
3
The entered data are: 3
7
The entered data are: 7
8
The entered data are: 8
11
The entered data are: 11
Press any key to continue . . .
```

a. How many numbers were **accepted** by the program?
b. If the <= were changed to <, how may data would have been **processed**?
c. If it were changed to !=, how many data items would have been **processed**?

a. 4. Based on the i <= 4; condition, 4 is included.
b. 3. Based on the i < 4; condition, 4 is not included.
c. 3. Based on i != 4; condition, when i == 4, the iteration stops.

```
#include <stdio.h>

int main()
{
   int i, k;
   printf("i = 1, i < 0, i = i + 1, what wrong???\n");
   for(i = 1; i <= 0; i = i + 1)
   {
     scanf_s("%d", &k, 1);
     printf("The entered data are: %d \n", k);
   }
   return 0;
}
```



```
i = 1, i < 0, i = i + 1, what wrong???
Press any key to continue . . .
```

a. How many numbers were accepted by the program?

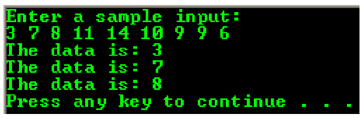
a. None. The program or the loop just terminate.
b. loop not executed at all. This is because of the terminal condition, i <= 0 already took effect (FALSE) when the for statement been evaluated while the initial value is i = 1.

b. How many times was the loop executed? Why?

The following statements are executed in the following order: Statement 1, 2, 3, 4, 2, 3, 4, 2, etc. until k = 11. Remember that the scanf()/scanf_s() will terminate when there is a whitespace. The next input will be read by the next scanf()/scanf_s().

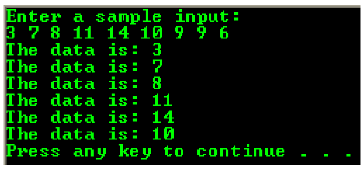
```
#include <stdio.h>

int main()
{
   int k;
   printf("Enter a sample input:\n");
   scanf_s("%d", &k, 1);                    // Statement 1
   for( ; k != 11; )                        // Statement 2
   {
     printf("The data is: %d \n", k);    // Statement 3
     scanf_s("%d", &k, 1);               // Statement 4
   }
   return 0;
}
```

a. How many numbers are accepted by the program?
b. How many times was the scanf_s() function called? And the printf()?
c. Was 11 ever **read** in as data? Why wasn't it **print**?
d. Did the **program** or the **data** determine the **number of times** the loop was to be performed?
e. If you wanted all the numbers up to 9 to be printed, how would you have changed the for statement?
f. Draw a flowchart and/or a tracechart for this program.



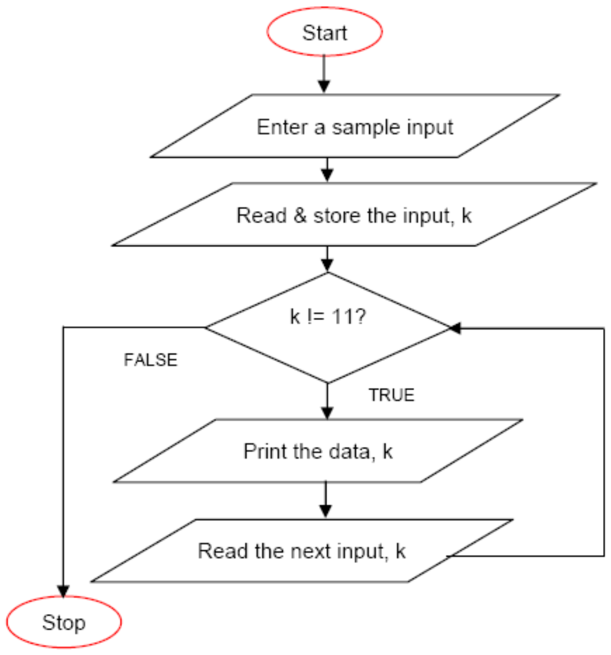
```
Enter a sample input:
3 7 8 11 14 10 9 9 6
The data is: 3
The data is: 7
The data is: 8
Press any key to continue . . .
```

a. 3.
b. scanf() and printf() both called 3 times.
c. Yes. Because the TRUE condition for the for loop is k != 11, so k == 11 will be FALSE and terminate the loop.
d. The data.
e. Change k != 11 to k != 9.



```
Enter a sample input:
3 7 8 11 14 10 9 9 6
The data is: 3
The data is: 7
The data is: 8
The data is: 11
The data is: 14
The data is: 10
Press any key to continue . . .
```

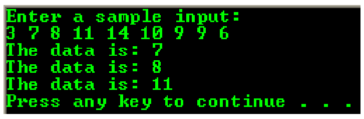
f. See below.



Change the position of the printf() and scanf_s() in the for body of the previous example.

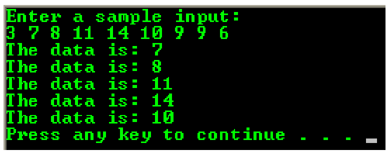
```
#include <stdio.h>

int main()
{
   int k;
   printf("Enter a sample input:\n");
   scanf_s("%d", &k, 1);                    // Statement 1
   for( ; k != 11; )                        // Statement 2
   {
     scanf_s("%d", &k, 1);                // Statement 4
     printf("The data is: %d \n", k);    // Statement 3
   }
   return 0;
}
```

a. Did the 11 get printed? Why or why not?
b. Did the first number that was read get printed? Why or why not?
c. If you wanted all the numbers up to 9 to be printed, how would you have change the for statement?
d. Draw a flowchart for this program.



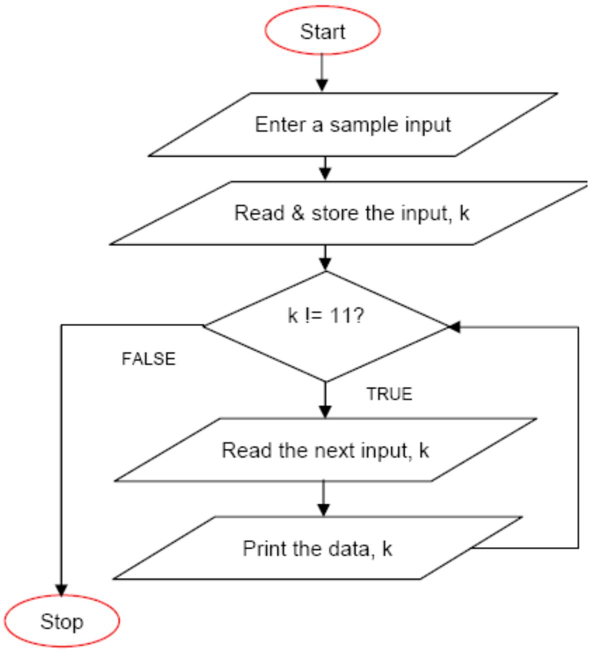
```
Enter a sample input:
3 7 8 11 14 10 9 9 6
The data is: 7
The data is: 8
The data is: 11
Press any key to continue . . .
```

a. Yes 11 was printed. This is because the scanf_s() in the for loop will read the next input, that is the second input, 7. Then the input was printed. When the 11 was read, it is printed first before been evaluated in the decision diamond box.
b. No. It is because the scanf_s() in the for loop will read the next input data, that is 7 and then print it.
c. Change k != 11 to k != 10.



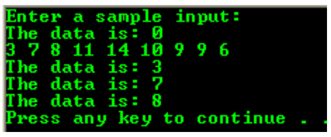
```
Enter a sample input:
3 7 8 11 14 10 9 9 6
The data is: 7
The data is: 8
The data is: 11
The data is: 14
The data is: 10
Press any key to continue . . .
```

d. See below.



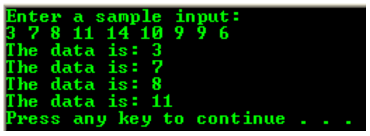
```c
#include <stdio.h>

int main()
{
   int k = 0;
   printf("Enter a sample input:\n");
   for( ; k != 11; )                        // Statement 2
   {
     printf("The data is: %d \n", k);    // Statement 3
     scanf_s("%d", &k, 1);               // Statement 4
   }
   return 0;
}
```

a. Did the 11 get printed? Why or why not?
b. Why didn't the first data item, 3, get printed first?
c. Would the 3 have been printed if the scanf_s() and the printf() were switched?



```
Enter a sample input:
The data is: 0
3 7 8 11 14 10 9 9 6
The data is: 3
The data is: 7
The data is: 8
Press any key to continue . .
```

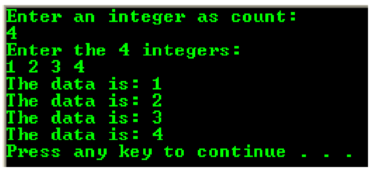
a. No. 11 wasn't printed. After 11 was read by the scanf_s() in the for loop, the decision k!=11 was evaluated, making it FALSE and the loop terminate.
b. From the output, we can see that the initial value of k = 0 has been printed before the program prompt for inputs. So, the next input to be read by the scanf_s() is 3. When there are inputs, the program will find the first scanf_s() to read and store those values.
c. Yes as shown in the following output. The initial value of k = 0 has been overwritten by the first input, 3 and next 3 been overwritten by 7 and so on.



```
Enter a sample input:
3 7 8 11 14 10 9 9 6
The data is: 3
The data is: 7
The data is: 8
The data is: 11
Press any key to continue . . .
```

```c
#include <stdio.h>

int main()
{
   int k = 0, m;
   printf("Enter an integer as count: \n");
   scanf_s("%d", &m);
   printf("Enter the %d integers: \n", m);
   for( ; m != 0; m = m - 1)
   {
     scanf_s("%d", &k);
     printf("The data is: %d \n", k);
   }
   return 0;
}
```

a. How many times was the body of the loop performed?
b. How many times was k assigned a new value through the scanf_s()?
c. What in the data determined how many times the loop was performed?



```
Enter an integer as count:
4
Enter the 4 integers:
1 2 3 4
The data is: 1
The data is: 2
The data is: 3
The data is: 4
Press any key to continue . . .
```

a. 4 times.
b. 4 times.
c. The data stored in m, that is 4.