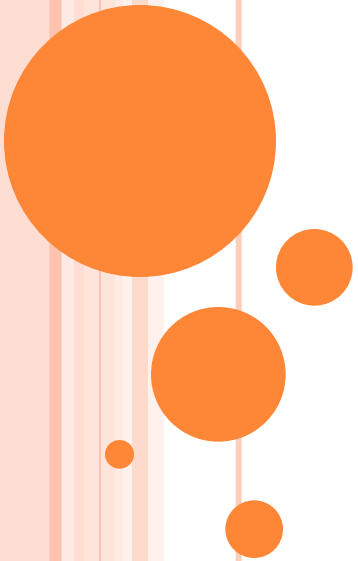
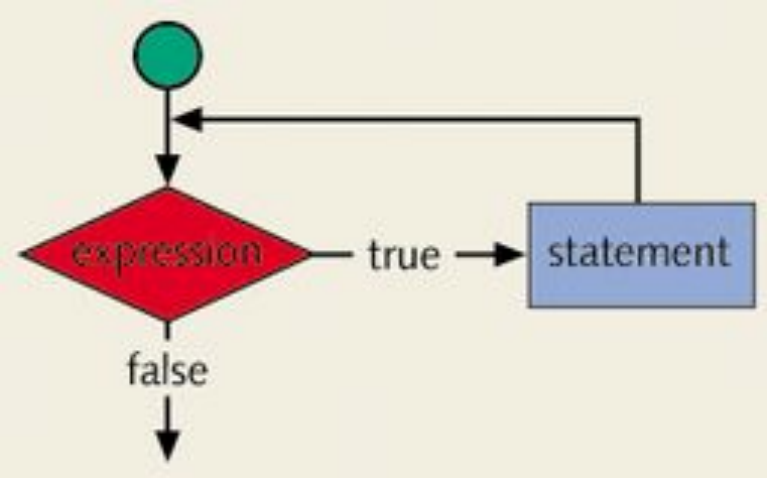


Control Structures II (REPETITION OR ITERATION OR LOOPING)

CONCEPT OF LOOPING (REPETITION) STRUCTURE

- One of the basic structured programming concepts
- The real power of computers is in their ability to repeat an operation or a series of operations many times
- When action is repeated many times, the flow is called a loop



WHY IS REPETITION NEEDED?

- Suppose we want to add five numbers (say to find their average).
- From what you have learned so far, you could proceed as follows.

```
scanf(%d %d %d %d  %d,
&num1,&num2,&num3,&num4,&num5);
sum = num1+num2+num3+num4+num5;
average = sum/5;
```

- Suppose we wanted to add and average `100`, or `1000`, or more numbers.
- We would have to declare that many variables, and list them again in `scanf` statement, and perhaps, again in the output statement.

SEQUENCE PROGRAM

- Suppose the numbers we want to add are the following:

```
5 3 7 9 4
```

Consider the following statements

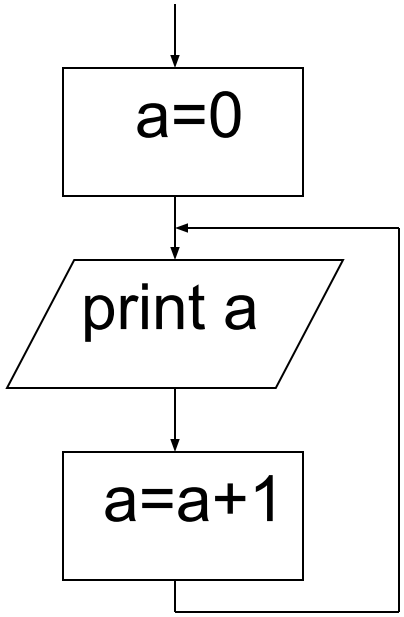
(1) `sum = 0;`
(2) `scanf(%d,&num);`
(3) `sum = sum + num;`

- Assume `sum` and `num` are variables of the type `int`.
- The first statement initializes `sum` to `0`.
- Execute statements 2 and 3.
- Statement 2 stores `5` in `num` and statement 3 updates the value of `sum` by adding `num` to it.
- After statement 3 the value of `sum` is `5`.

SEQUENCE PROGRAM

- To add 10 numbers, you can repeat statements 2 and 3 10 times.
- To add 100 numbers we can repeat statements 2 and 3 100 times.
- You would not have to declare any addition variables as in the first code.
- There are three repetition or looping structures in C that lets us repeat statements over and over until certain conditions are met.
  - **`while`** Looping Structure
  - **`for`** Looping Structure
  - **`do`**…**`while`** Looping Structure

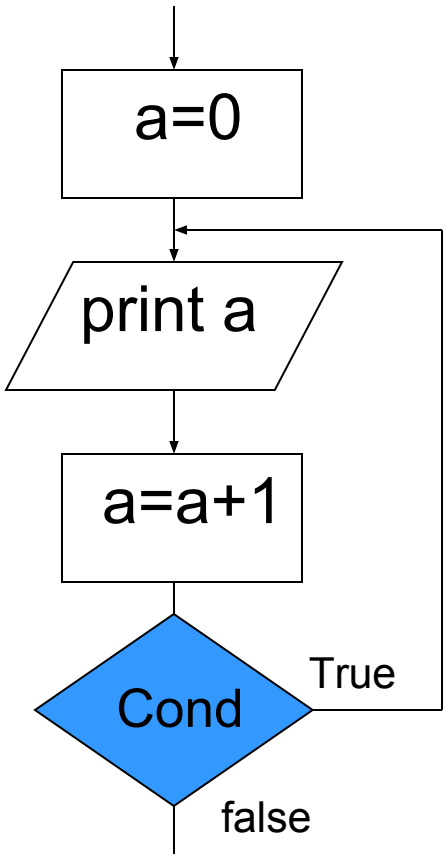
WHAT IS A LOOP



0 1 2 3 4 5 6 7 ……..?

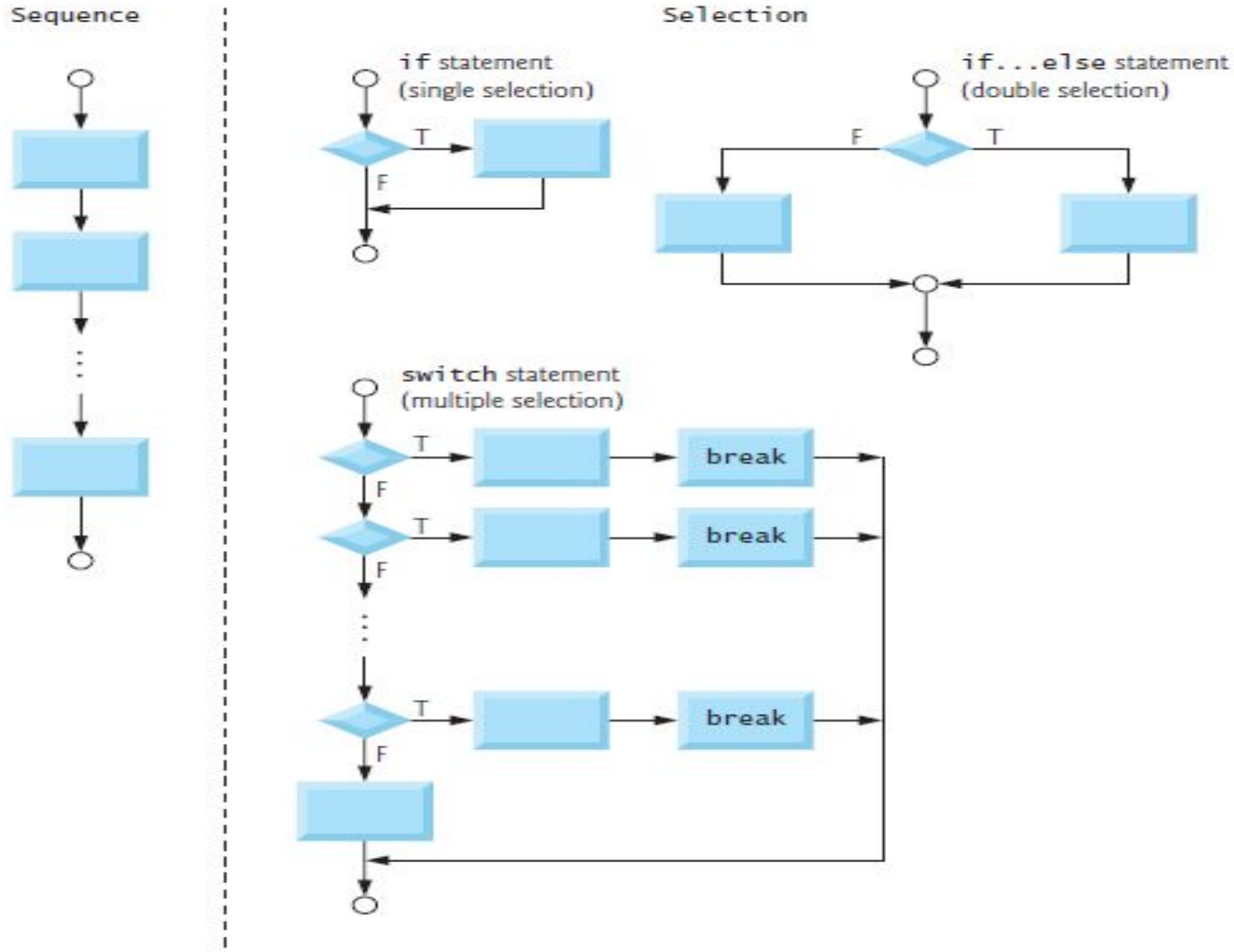
- Loops repeat a statement, a block, a function or a set of any combination of these construct a number of times
- The figure on the left demonstrates the repetition process of the statement printf and a=a+1.
- The question about the figure is that when this loop will stop?
- The loop will never stop and will infinitely repeat these statements until the program is forced to quit by the user

WHAT IS A LOOP

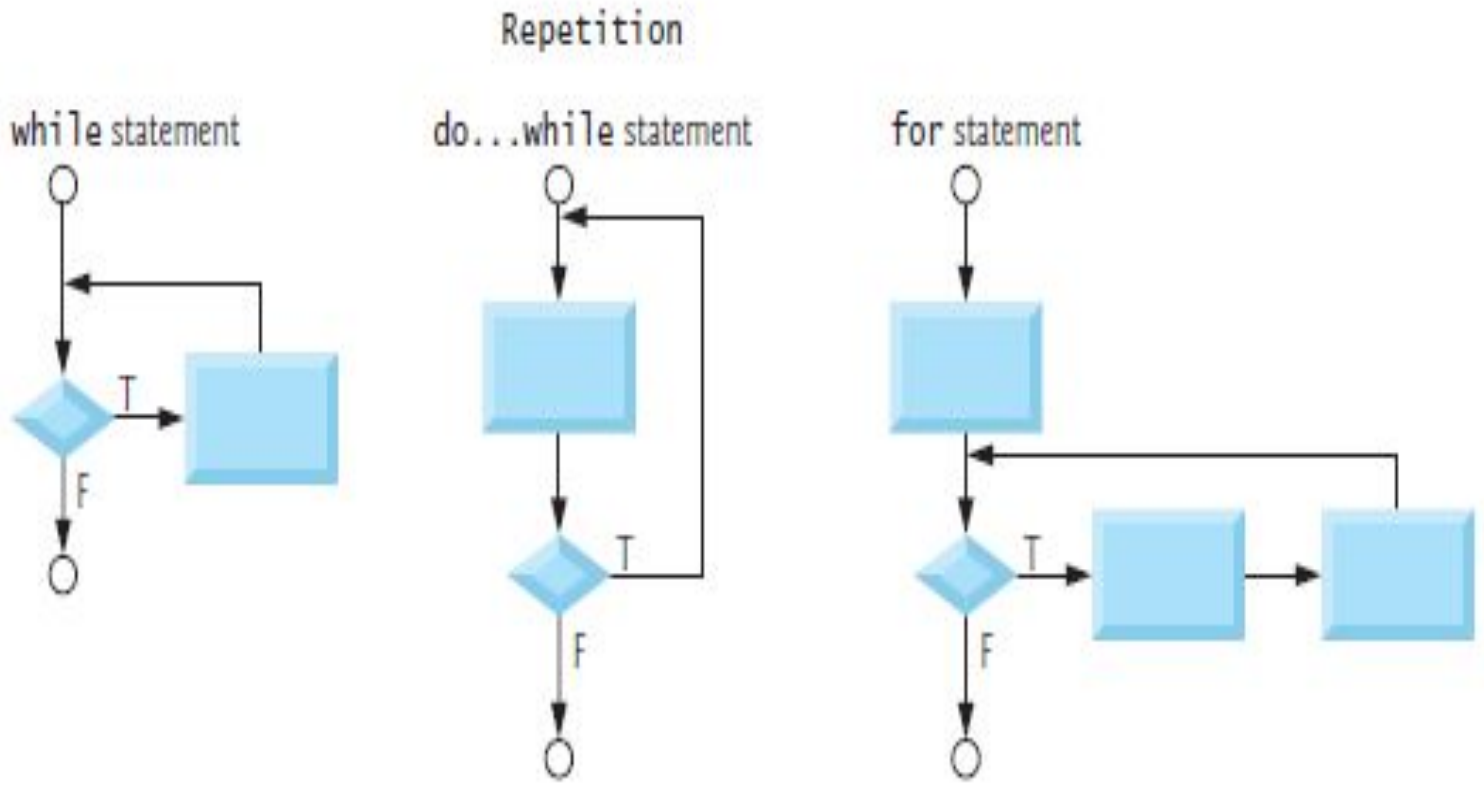


- A loop must be controlled by a condition or an event in order to limit the number of iterations and prevent infinite loops from occurring
- The figure to the left shows a loop controlled by a condition.
- If the condition is true the loop will go on repeating the statements otherwise it breaks the cycle and continues with the rest of the program
- The condition can be any logical or relational expression in C language

SELECTION STATEMENTS



REPETITION STATEMENT

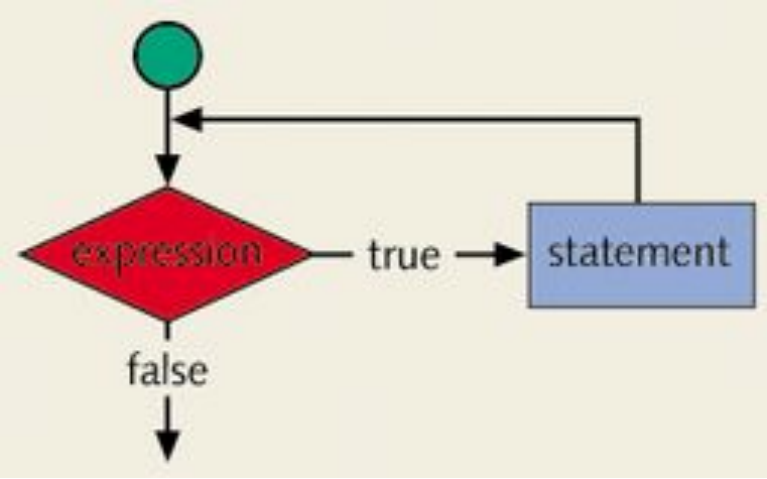


THE `WHILE` LOOPING (REPETITION) STRUCTURE

- The general form of the `while` statement `is`

```
while(expression)
    statement
```

- The `statement` can be either a simple or compound statement.
- The `statement` is called the body of the loop.
- The `expression` provides an entry condition.



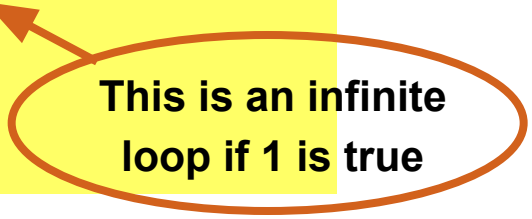
WHILE LOOPS

```
while ( expression )
{
    loop body
}
```

```
while ( a>b )
{
}
```

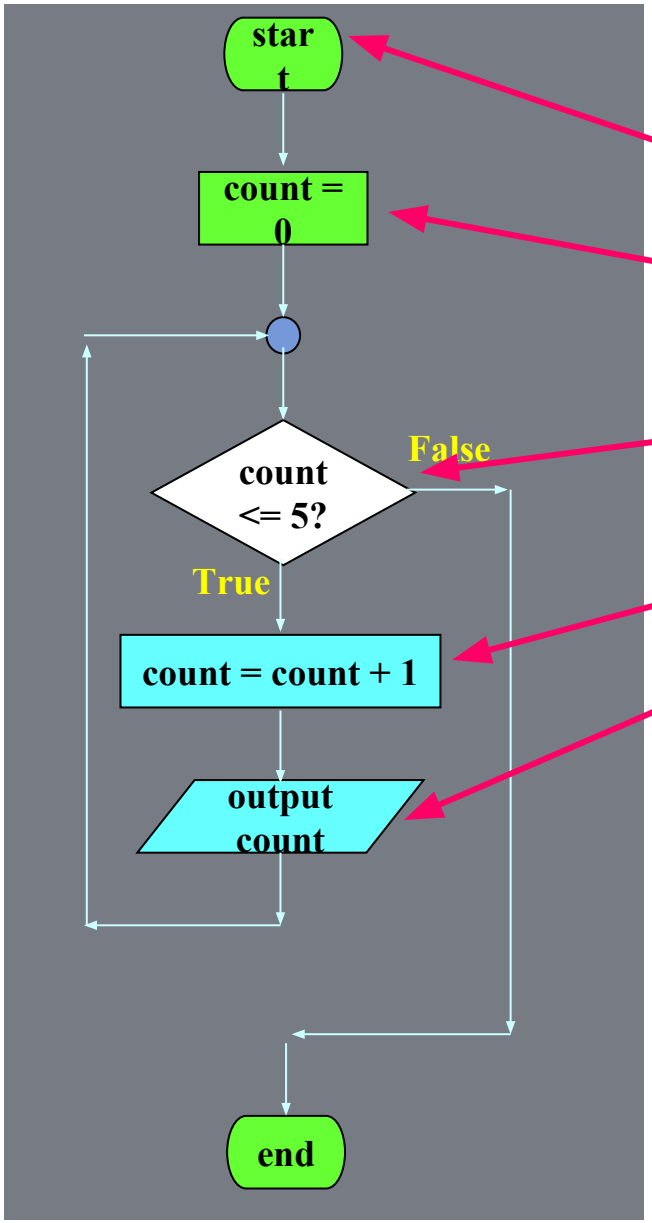
```
while ( a>b && c<=78 )
{
}
```

```
while ( 1 )
{
}
```



- The "loop body" can be either a simple or compound statement.
- The "expression" provides an entry condition.
- A while loop checks the expression before entering the loop body
- The expression is evaluated first (must produce true or false) then if the value is true (none zero) the loop will continue otherwise it exits the loop body proceeding with the rest of the program.

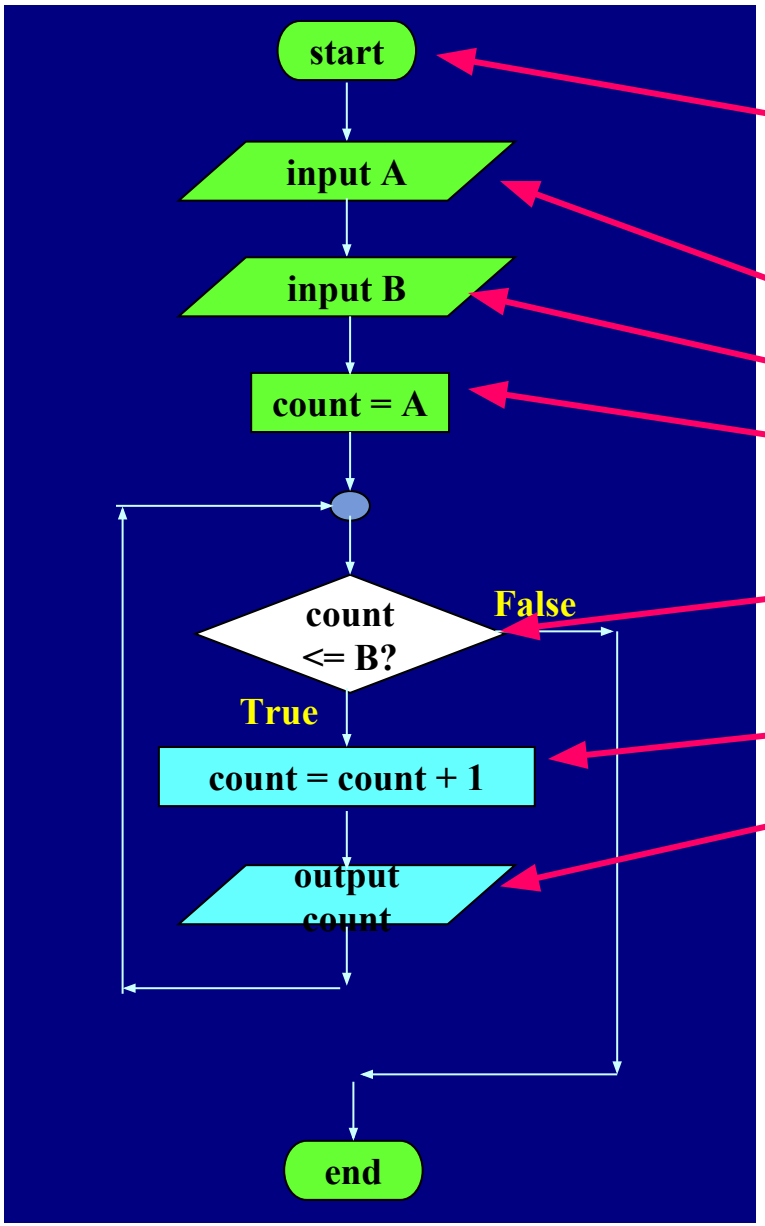
WHILE LOOPS: EXAMPLES (1)




```
int main()
{
   int count = 0;

   while(count <= 5)
   {
      count = count + 1;
    printf("%d ",count);
    }
}
```

```
1 2 3 4 5 6
```

WHILE LOOPS: EXAMPLES (2)



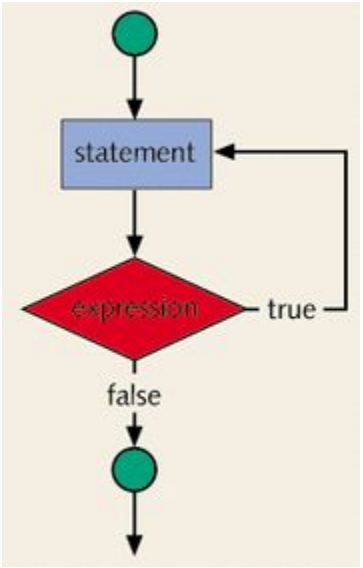

```
int main()
{
   int count, A, B;
   scanf("%d",&A);
   scanf("%d",&B);
   count = A;

   while(count <= B)
    {
     count = count + 1;
      printf("%d ",count);
    }
}
```

```
3 8

4 5 6 7 8 9
```

THE DO…WHILE LOOPING (REPETITION) STRUCTURE

- The general form of a `do...while statement is:`

```
do
   statement
while(expression);
```



- The `statement` executes first, and then the `expression` is evaluated.
- If the `expression` evaluates to `true`, the `statement` executes again.
- As long as the `expression` in a `do...while` statement is `true`, the `statement` executes.

DO..WHILE LOOPS

```
do
{
   loop body
} while ( expression );
```

```
do
{
} while ( a>b );
```

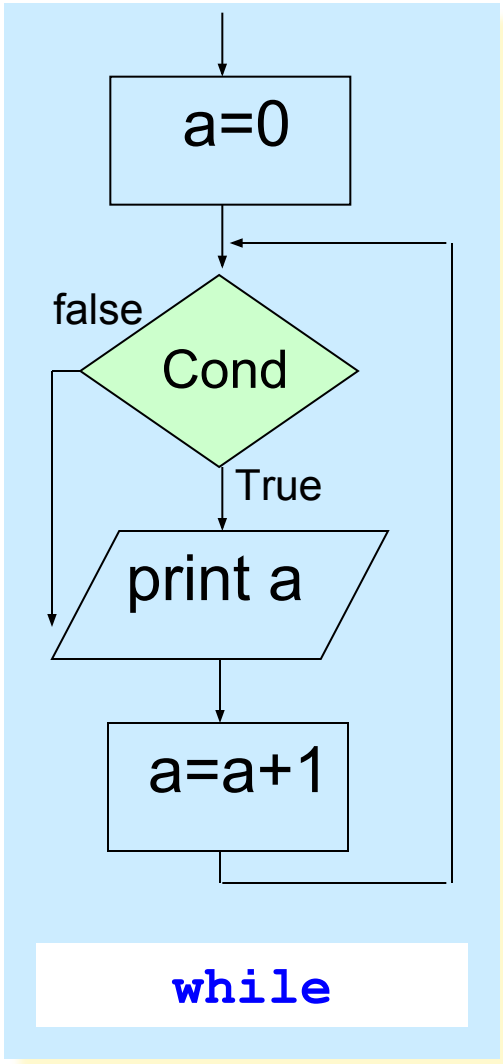
```
do
{
} while ( a>b && c<=7 );
```

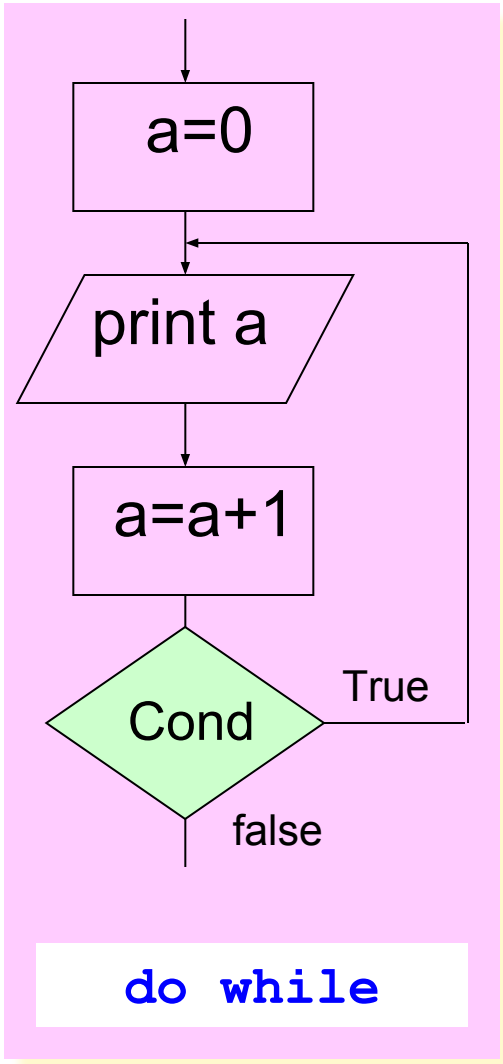
```
do
{
} While ( 1 );
```

This is an infinite loop if 1 is true

- A do...while loop will allow the execution of the body once then checks the expression, if the expression is true then it will repeat again otherwise it exits the loop.
- The expression is evaluated first (must produce true or false) then if the value is true (none zero) the loop will continue otherwise it exits the loop body proceeding with the rest of the program.
- To avoid an infinite loop, make sure that the loop body contains a statement that ultimately makes the `expression` `false` and assures that it exits.

WHILE LOOP AND DO..WHILE LOOP





- Loops in C language fall under two major categories, these are
  - **while** , **For** loop

    The while loop, has the condition checked before entering the loop
  - **do while** loop

    This type of loops will enter the loop body at least once then checks the condition
- The two figures on the left show those two types

DO…WHILE LOOPING STRUCTURE

- Because the `while` and `for` loop has an entry condition, it may never activate, whereas, the `do...while` loop, which has an exit condition, always goes through at least once.

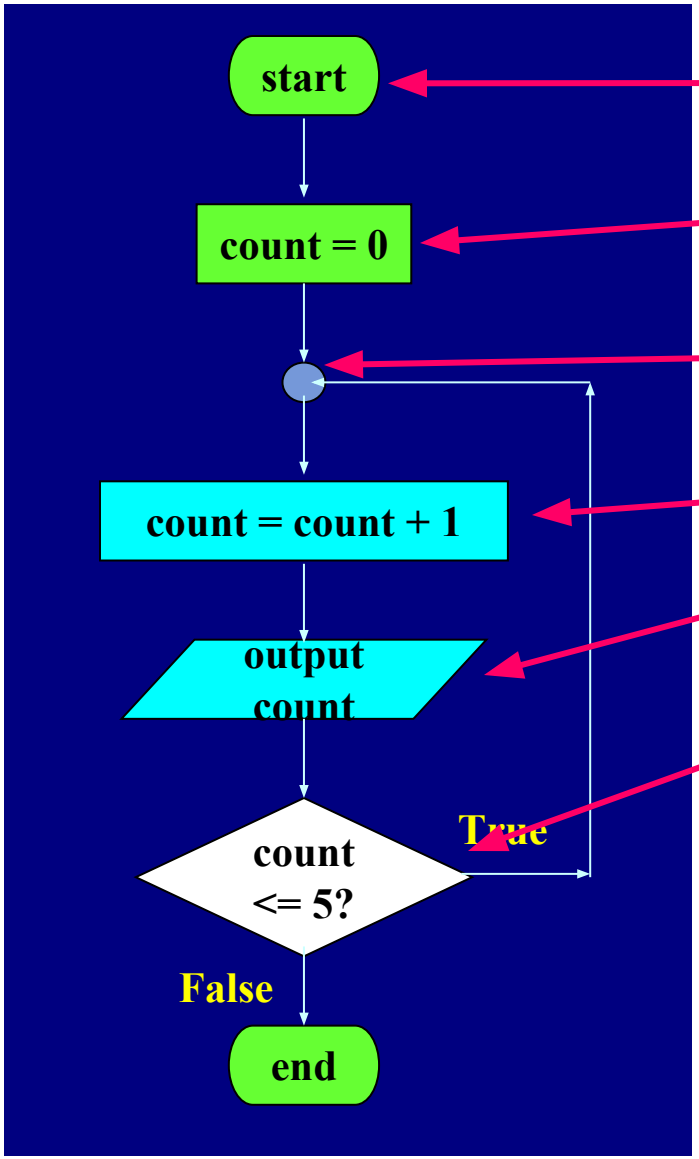
```
i = 11;
while(i <= 10)
{
  printf("%d ",i);
  i++;
}
```

In the `while` loop, produces nothing.

```
i = 11;
do
{
  printf("%d ",i);
  i++;
}
while(i <= 10);
```

the `do...while` loop, outputs the number `11`.

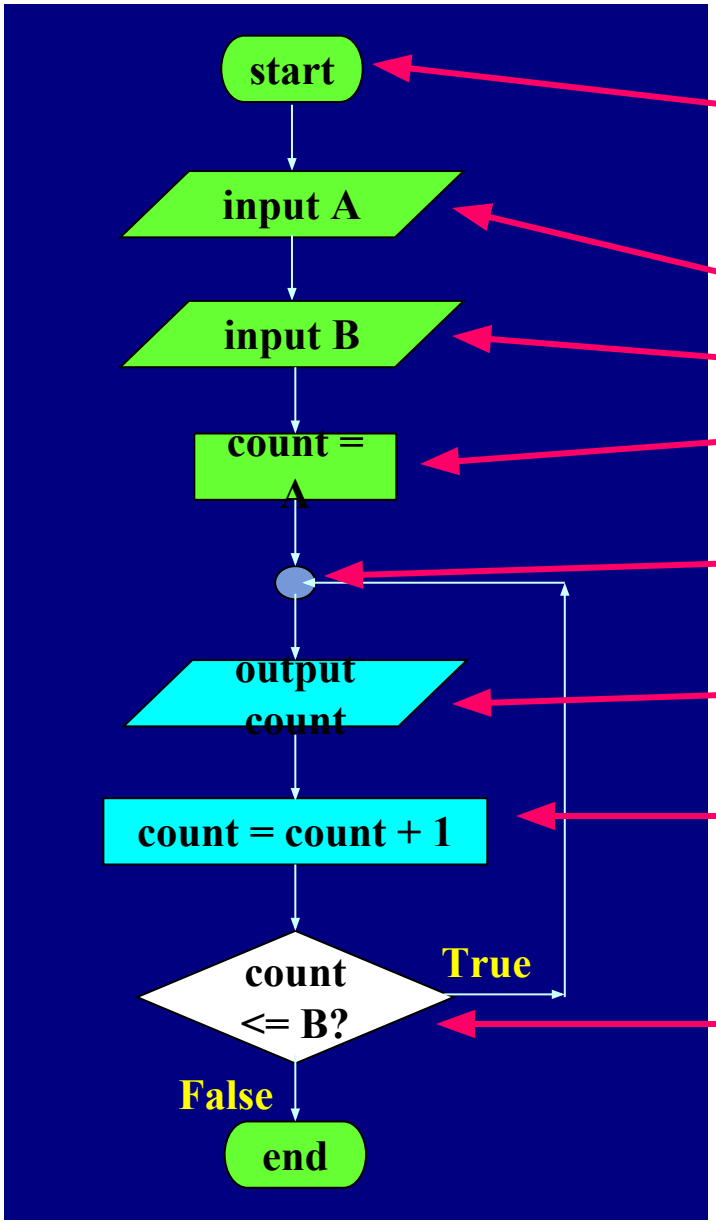
DO WHILE EXAMPLE(3)




```
int main()
{
    int count = 0;

    do
    {
        count = count + 1;
        printf("%d ",count);
    } while(count <= 5);
}
```

```
1 2 3 4 5 6
```

DO WHILE EXAMPLE(4)




```
int main()
{
   int count, A, B;
   scanf("%d",&A);
   scanf("%d",&B);
   count = A;

   do
    {
      count = count + 1;
      printf("%d ",count);
    } while(count <= B)
}
```

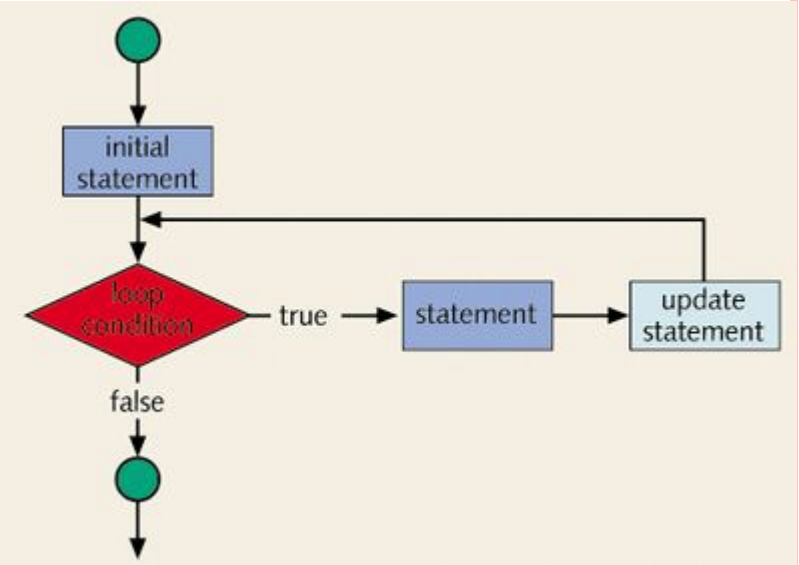
```
3 8

4 5 6 7 8 9
```

THE **FOR** LOOPING (REPETITION) STRUCTURE

- The general form of the `for` statement is

```
for(initial statement; loop condition; update statement)
    statement
```

- The `initial statement`, `loop condition` and `update statement` (called `for` loop **control statements**) that are enclosed with in the parentheses controls the body (the statement) of the `for` statement.



FOR LOOPING STRUCTURE

- The `for` loop executes as follows:
  1. The `initial statement` executes.
  2. The `loop condition` is evaluated. If the `loop condition` evaluates to `true`
     1. Execute the `for` loop `statement`.
     2. Execute the `update statement`.
  3. Repeat Step 2 until the `loop condition` evaluates to `false`.

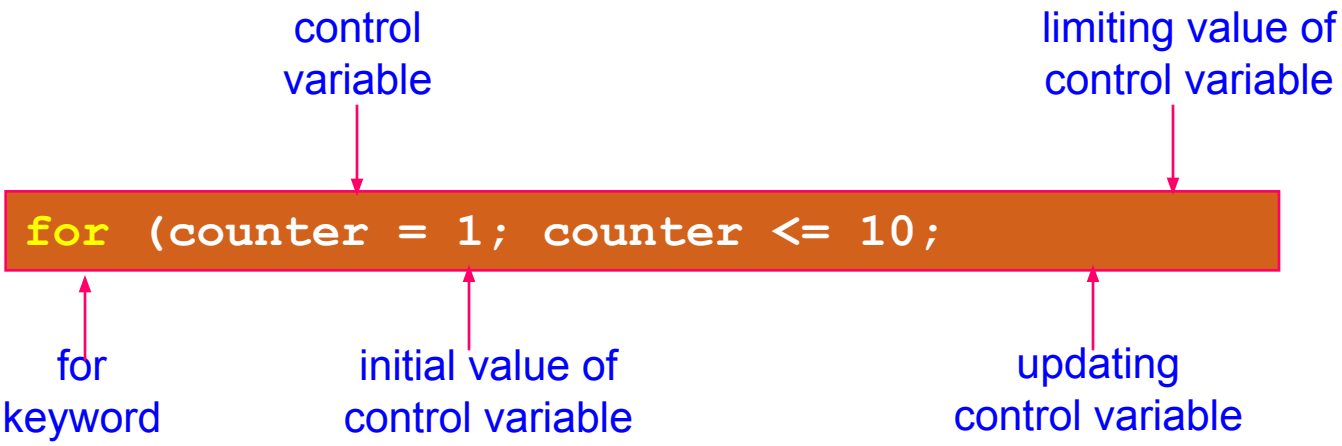
- The `initial statement`
  - usually initializes a variable
  - is the first statement to be executed and is executed only once.

THE **FOR** LOOPING (REPETITION) STRUCTURE

- **Problem:** print numbers 1 through 10 on standard output
- Solution using **for** statement

```
1 2 3 4 5 6 7 8 9 10
```

```
int counter;

for (counter = 1; counter <= 10; counter++)
   printf("%d ",counter);
```



FOR LOOPING STRUCTURE

- The statement of a `for` loop may be a simple or a compound statement.

simple	compound

`for(i = 1; i <= 5; i++)`
`  printf("Output of stars.\n");`
`printf("*");`
	`for(i = 1; i <= 5; i++)`
	`{`
	`  printf("Output of stars.\n");`
	`  printf("*\n");`
	`}`

```
Output a line of stars.
Output a line of stars.
Output a line of stars.
Output a line of stars.
Output a line of stars.
*
```

```
Output a line of stars.
*
Output a line of stars.
*
Output a line of stars.
*
Output a line of stars.
*
Output a line of stars.
*
```

```
for(i = 1; i <= 5; i++);
    printf("*");
```

```
*
```

FOR LOOPING STRUCTURE

- All three statements can be omit—initial statement, loop condition, and update statement.

```
for(;;)
   printf("Hello\n");
```

```
Hello
Hello
.
.
```

Infinite output; press break to stop the program running

- Count backward

```
for(i = 10; i >= 1; i--)
     printf("%d ",i);
```

```
10 9 8 7 6 5 4 3 2 1
```

- Increment (or decrement) the loop control variable

```
for(i = 1; i <= 20; i = i + 2)
    printf("%d ",i);
```

```
1 3 5 7 9 11 13 15 17 19
```

Example 5

FOR LOOPING STRUCTURE

```
// reads five numbers and find their sum and average
#include <stdio.h>

int main()
{
    int i, newNum, sum, average;

    sum = 0;
    for(i = 1; i <= 5; i++)
    {
      scanf("%d",&newNum);
      sum = sum + newNum;
    }
average = sum / 5;
printf("The sum is %d\n",sum);
printf("The average is %d",average);
return 0;
}
```

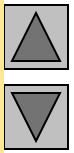
```
5 4 3 2 1
The sum is 15
The average is 3
```

3.8 FORMULATING ALGORITHMS (COUNTER-CONTROLLED REPETITION)

- Counter-controlled repetition
  - Loop repeated until counter reaches a certain value
  - Definite repetition: number of repetitions is known
  - Example: A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz
  - Pseudocode:

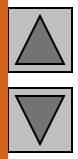
Set total to zero
Set grade counter to one
While grade counter is less than or equal to ten
Input the next grade
Add the grade into the total
Add one to the grade counter
Set the class average to the total divided by ten
Print the class average

```
1  /* Fig. 3.6: fig03_06.c
2     Class average program with
3     counter-controlled repetition */
4  #include <stdio.h>
5
6  int main()
7  {
8     int counter, grade, total, average;
9
10    /* initialization phase */
11    total = 0;
12    counter = 1;
13
14    /* processing phase */
15    while ( counter <= 10 ) {
16       printf( "Enter grade: " );
17       scanf( "%d", &grade );
18       total = total + grade;
19       counter = counter + 1;
20    }
21
22    /* termination phase */
23    average = total / 10;
24    printf( "Class average is %d\n",
25
26    return 0;   /* indicate program ended
27 }
```



Outline

1. Initialize Variables
2. Execute Loop
3. Output results



Outline

Program Output

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```

3.9 FORMULATING ALGORITHMS WITH TOP-DOWN, STEPWISE REFINEMENT

- Problem becomes:

  Develop a class-averaging program that will process an arbitrary number of grades each time the program is run.

  - Unknown number of students
  - How will the program know to end?
- Use sentinel value
  - Also called signal value, dummy value, or flag value
  - Indicates "end of data entry."
  - Loop ends when user inputs the sentinel value
  - Sentinel value chosen so it cannot be confused with a regular input (such as `-1` in this case)

3.9 FORMULATING ALGORITHMS WITH TOP-DOWN, STEPWISE REFINEMENT

- Top-down, stepwise refinement
  - Begin with a pseudocode representation of the *top*:

    Determine the class average for the quiz
  - Divide *top* into smaller tasks and list them in order:

    Initialize variables
    Input, sum and count the quiz grades
    Calculate and print the class average
- Many programs have three phases:
  - Initialization: initializes the program variables
  - Processing: inputs data values and adjusts program variables accordingly
  - Termination: calculates and prints the final results

3.9 FORMULATING ALGORITHMS WITH TOP-DOWN, STEPWISE REFINEMENT

- Refine the initialization phase from *Initialize variables* to:

  Initialize total to zero
  Initialize counter to zero

- Refine *Input, sum and count the quiz grades* to

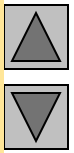
  Input the first grade (possibly the sentinel)
  While the user has not as yet entered the sentinel
  Add this grade into the running total
  Add one to the grade counter
  Input the next grade (possibly the sentinel)

3.9 FORMULATING ALGORITHMS WITH TOP-DOWN, STEPWISE REFINEMENT

- Refine *Calculate and print the class average* to

```
If the counter is not equal to zero
    Set the average to the total divided by the counter
    Print the average
else
    Print "No grades were entered"
```


```
1  /* Fig. 3.8: fig03_08.c
2     Class average program with
3     sentinel-controlled repetition */
4  #include <stdio.h>
5
6  int main()
7  {
8     float average;                /* new
9     int counter, grade, total;
10
11    /* initialization phase */
12    total = 0;
13    counter = 0;
14
15    /* processing phase */
16    printf( "Enter grade, -1 to end: " );
17    scanf( "%d", &grade );
18
19    while ( grade != -1 ) {
20       total = total + grade;
21       counter = counter + 1;
22       printf( "Enter grade, -1 to end: "
23       scanf( "%d", &grade );
24    }
```



Outline

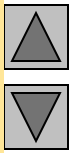
1. Initialize Variables

2. Get user input

2.1 Perform Loop

```
25
26      /* termination phase */
27      if ( counter != 0 ) {
28         average = ( float ) total /
29         printf( "Class average is %.2f",
30      }
31      else
32         printf( "No grades were entered\n"
33
34      return 0;    /* indicate program ended
35   }
successfully */
```

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50
```



Outline

3. Calculate Average

3.1 Print Results

Program Output

3.10 NESTED CONTROL STRUCTURES

- Problem
  - A college has a list of test results (**1** = pass, **2** = fail) for 10 students
  - Write a program that analyzes the results
    - If more than 8 students pass, print "Raise Tuition"
- Notice that
  - The program must process 10 test results
    - Counter-controlled loop will be used
  - Two counters can be used
    - One for number of passes, one for number of fails
  - Each test result is a number—either a **1** or a **2**
    - If the number is not a **1**, we assume that it is a **2**

3.10 NESTED CONTROL STRUCTURES

- Top level outline
  - *Analyze exam results and decide if tuition should be raised*
- First Refinement
  - *Initialize variables*
  - *Input the ten quiz grades and count passes and failures*
  - *Print a summary of the exam results and decide if tuition should be raised*
- Refine *Initialize variables* to
  - *Initialize passes to zero*
  - *Initialize failures to zero*
  - *Initialize student counter to one*

3.10 NESTED CONTROL STRUCTURES

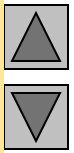
- Refine *Input the ten quiz grades and count passes and failures* to

```
While student counter is less than or equal to ten
   Input the next exam result
   If the student passed
      Add one to passes
   else
      Add one to failures
   Add one to student counter
```

- Refine *Print a summary of the exam results and decide if tuition should be raised* to

```
Print the number of passes
Print the number of failures
If more than eight students passed
   Print "Raise tuition"
```


```
1   /* Fig. 3.10: fig03_10.c
2      Analysis of examination results */
3   #include <stdio.h>
4
5   int main()
6   {
7      /* initializing variables in
8      int passes = 0, failures = 0, student
9
10     /* process 10 students;
11     while ( student <= 10 ) {
12        printf( "Enter result (
13        scanf( "%d", &result );
14
15        if ( result == 1 )          /*
16           passes = passes + 1;
17        else
18           failures = failures + 1;
19
20        student = student + 1;
21     }
22
23     printf( "Passed %d\n", passes );
24     printf( "Failed %d\n", failures );
25
26     if ( passes > 8 )
27        printf( "Raise tuition\n" );
28
29     return 0;    /* successful termination
30  }
*/
```

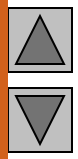



Outline

1. Initialize variables

2. Input data and count passes/failures

3. Print results



Outline

Program Output

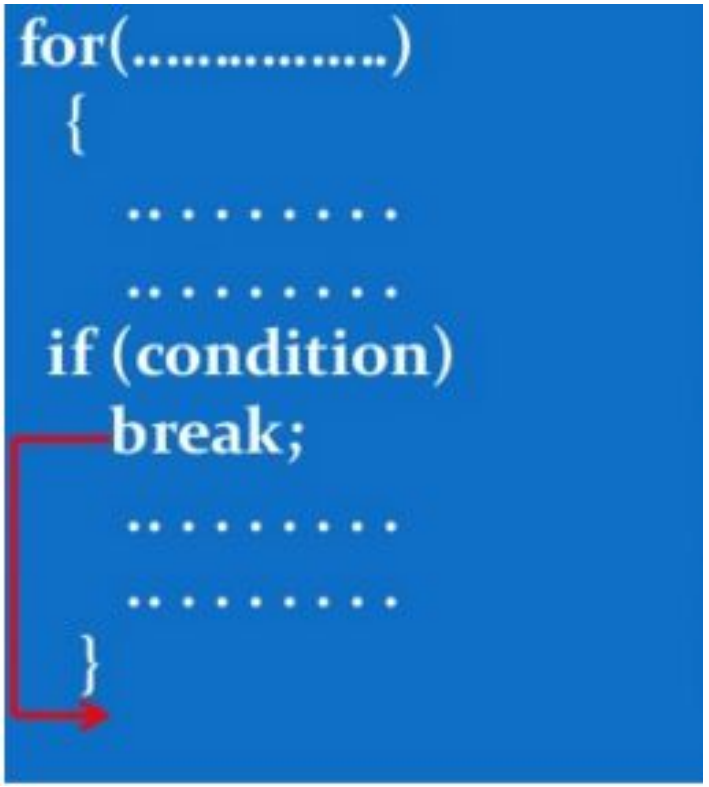
```
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Passed 6
Failed 4
```

BREAK STATEMENT

- A `break` and `continue` statement alters the flow of control.
- The `break` statement, when executed in a `switch` structure, provides an immediate exit from the `switch` structure.
- You can use the `break` statement in `while`, `for`, and `do...while` loops.
- When the `break` statement executes in a repetition structure, it immediately exits from these structures.
- The `break` statement is typically used for two purposes:
  1. To exit early from a loop
  2. To skip the remainder of the `switch` structure
- After the `break` statement executes, the program continues to execute with the first statement after the structure.

LOOPS AND BREAK KEYWORD

```
int i;
for(i=1;i<=10;i++)
{
  if(i==3)
  {
        break;
  }
      printf("%d\t",i);
}
```

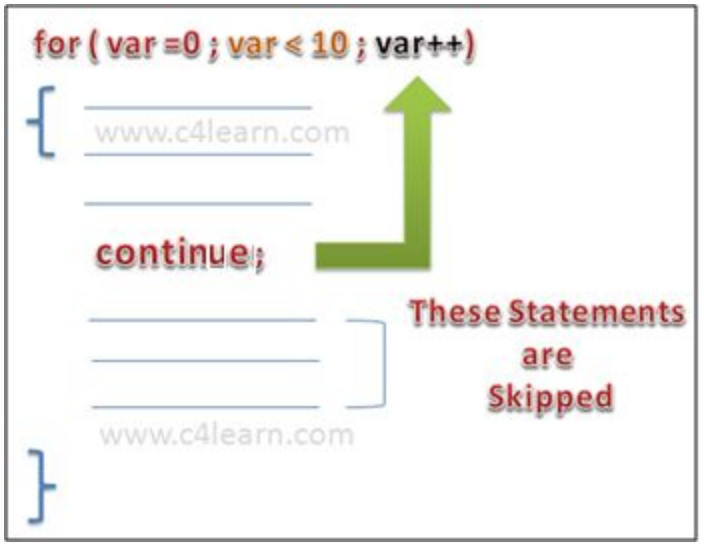


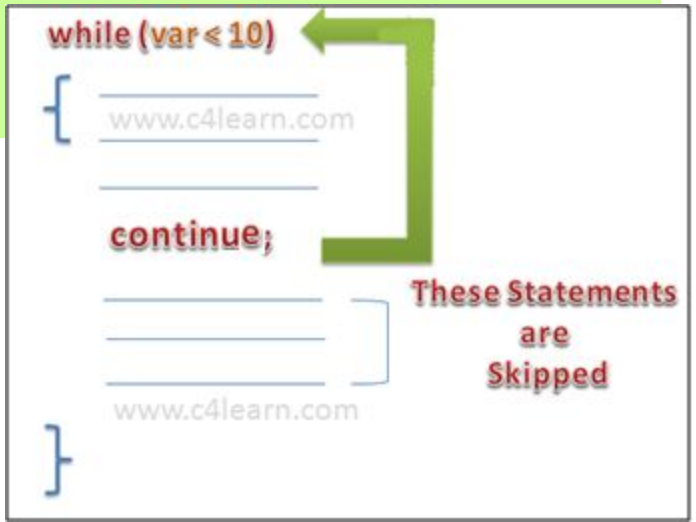
CONTINUE STATEMENT

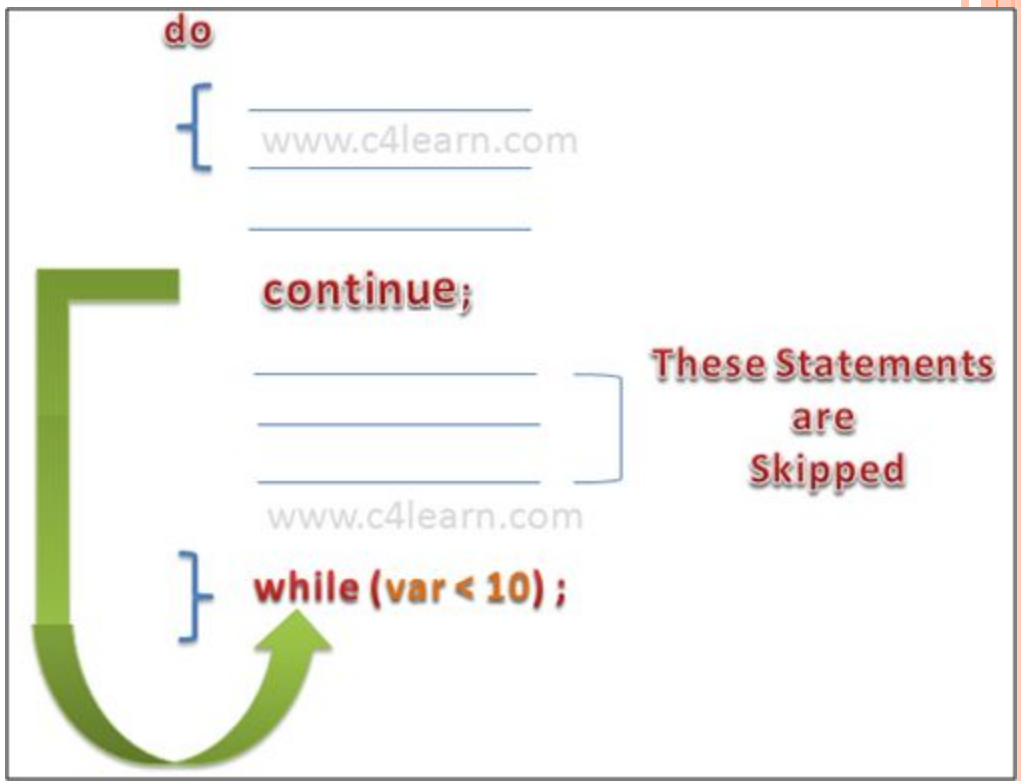
- The `continue` statement is used in `while`, `for`, and `do-while` structures.
- When the `continue` statement is executed in a loop, it skips the remaining statements in the loop and proceeds with the next iteration of the loop.
- In a `while` and `do-while` structure, the `expression` (that is, the loop-continue test) is evaluated immediately after the `continue` statement.
- In a `for` structure, the `update statement` is executed after the `continue` statement, and then the `loop condition` (that is, the loop-continue test) executes.

CONTINUE STATEMENT

```
int i;
for(i=1;i<=10;i++)
{
    if((i==3)||(i==7))
    {
        continue;
    }
     printf("%d\t",i);

}
```







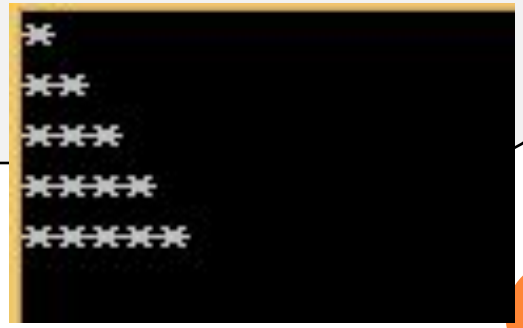
NESTED LOOPING

- A loop (e.g., **while**, **do-while**, **for**) can be placed within the body of another loop.
- When one loop is nested within another, several iterations of the inner loop are performed for every single iteration of the outer loop.
- The inner and outer loops need not to be generated by the same type of control structures.

NESTED FOR LOOP (EXAMPLE 6)

```
for ( expr1a; expr2a; expr3a )
{
      :
      for ( expr1b; expr2b; expr3b )
       {
            :
        }
      :
}
```

```
#include<stdio.h>
int main()
{
 int i,j;
for (i=1; i<=5;i++)          //Line 1
{
  for (j=1;j<=i;j++)         //Line 2
  {
      printf("*");           //Line 3
  }
printf("\n");              //Line 4
}
 }
```



NESTED WHILE (EXAMPLE 7)

```
while (expr1)

{
    :
    while (expr2)
    {
        :
        Update expr2;
    }
:
update expr1;
}
```

```
#include<stdio.h>
int main()
{
 int r,c,s;

 r=1;
 while(r<=5)  /*outer loop*/
 {
  c=1;
  while(c<=2)  /*inner loop*/
  {
   s=r+c;
   printf("r=%d c=%d sum=%d\n",r,c,s);
   c++;
  }
  printf("\n");
  r++;
 }
 }
```

```
r=1 c=1 sum=2
r=1 c=2 sum=3

r=2 c=1 sum=3
r=2 c=2 sum=4

r=3 c=1 sum=4
r=3 c=2 sum=5

r=4 c=1 sum=5
r=4 c=2 sum=6

r=5 c=1 sum=6
r=5 c=2 sum=7
```

NESTED DO WHILE (EXAMPLE 8)

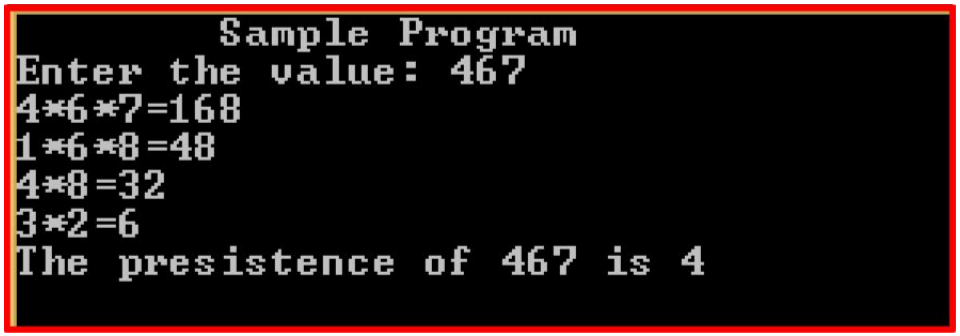
```
do

{
  :
  do
  {
    :
    Update expr;
  }
    while(expr);
:
update expr;
}
while(expr);
```

```
#include<stdio.h>
int main()
{
int i=1,j=0,sum;
do {
      sum=0;
      do{
            sum=sum+j;
            printf("%d",j);
             j++;
            if(j<=i)
            {
          printf("+");
             }
      }while(j<=i);
printf("=%d\n",sum);
j=1;
i++;
}while(i<=10);
}
```

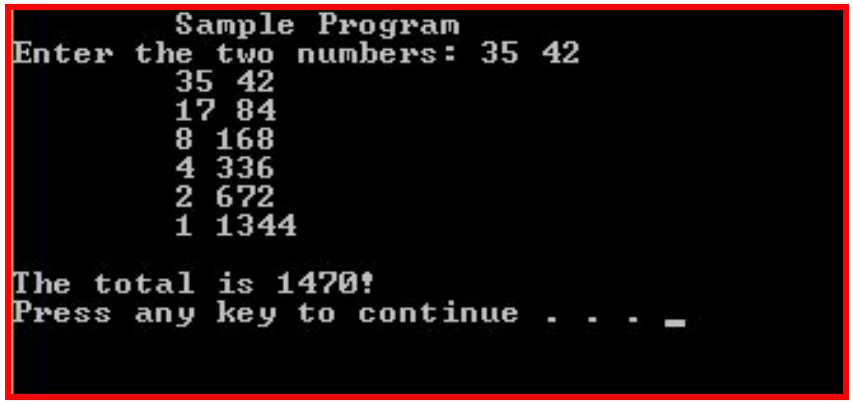
```
0+1=1
1+2=3
1+2+3=6
1+2+3+4=10
1+2+3+4+5=15
1+2+3+4+5+6=21
1+2+3+4+5+6+7=28
1+2+3+4+5+6+7+8=36
1+2+3+4+5+6+7+8+9=45
1+2+3+4+5+6+7+8+9+10=55
```

EXERCISES

1. Find the persistence of a number. It is the number, if times you can multiple the digits (iteratively) before you get a single digit number.



2. Given two numbers multiply them using the following method. Successively divide the smaller number by 2 (ignore any remainder) until the quotient is 1 and multiple the larger number by 2. Add to a total only those multiples of the larger which correspond to an odd quotient of the smaller.



3. Find the digital root of a number (repeatedly find the sum of the digits until you get a number 1 to 9.

   285:

   2+8+5=15

   1+5=6

4. A ball is dropped from a height of x. It decreases 2/3 on height x at each bounce. How many bounces the ball will have until height is less than y? (Use while loop and if condition to break, when height x becomes less than y).

5. Write a C program that receives the total number of integers (N). Then, the program will ask for N real numbers. By applying for loop, your program should find and display the largest and the lowest of the numbers. Give a proper message for invalid user input (Don’t use arrays).