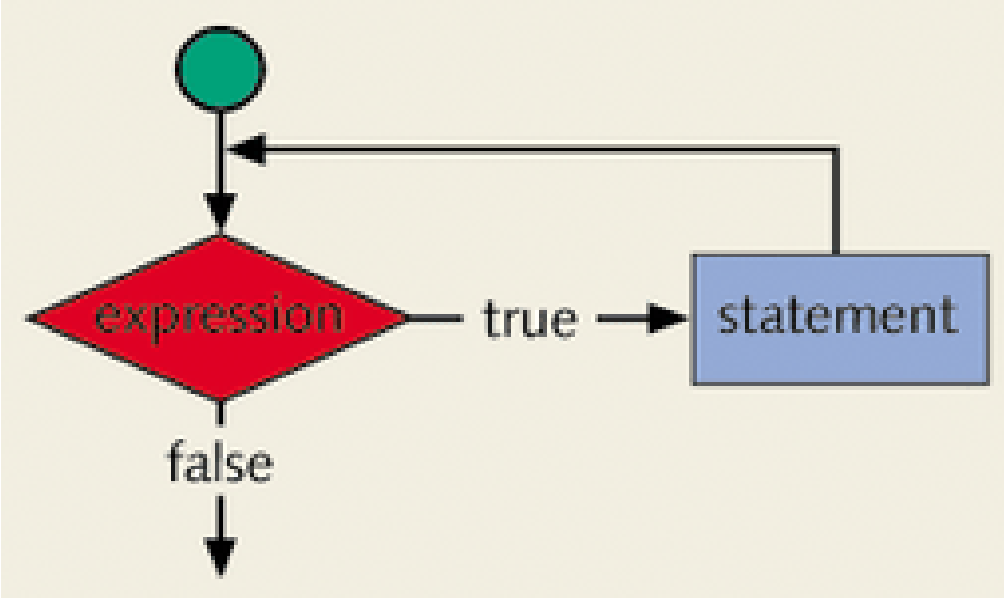
Control Structure/ Iterative /Repetition Structure

By: Atiya Jokhio

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.

Repetition

- There is two means of repetition
  1. Counter-controlled repetition
  2. Sentinel-controlled repetition

- 1. **Counter-controlled repetition:** is sometimes called definite repetition because we know in advance exactly how many times the loop will be executed.
- In counter-controlled repetition, a control variable is used to count the number of repetitions. The control variable is incremented (usually by 1) each time the group of instructions is performed. When the value of the control variable indicates that the correct number of repetitions has been performed, the loop terminates and execution continues with the statement after the repetition statement.

Repetition [Cont.]

2. **Sentinel-controlled repetition**

- Sentinel-controlled repetition is sometimes called indefinite repetition because it's not known in advance how many times the loop will be executed.
- Sentinel values are used to control repetition when:

  - The precise number of repetitions isn't known in advance, and

  - The loop includes statements that obtain data each time the loop is performed.

Repetition [Cont.]

Counter-controlled repetition

```
sum = 0;

n = 1;

while (n <= 10)

{

sum = sum + n*n;

n = n+ 1;

}
```

Sentinel-controlled repetition

```
do

{

printf("Input a number.\n");

scanf("%d", &num);

}

while(num>0);
```

Counter-controlled repetition

Counter-controlled repetition requires:

- The name of a control variable (or loop counter).

- The initial value of the control variable.

- The increment(or decrement) by which the control variable is modified each time through the loop.

- The condition that tests for the final value of the control variable (i.e., whether looping should continue).

Loop Types

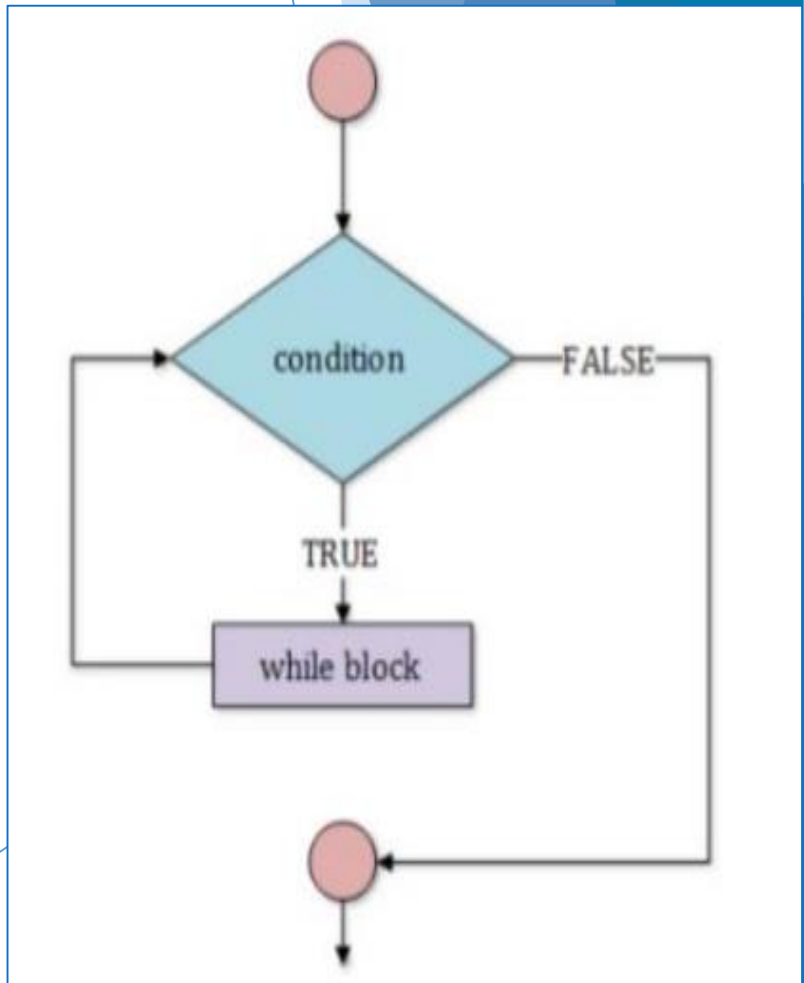
- There are three repetition or looping structures in C that lets us repeat statements over and over until certain conditions are met.
  - **while** Looping Structure
  - **for** Looping Structure
  - **do…while** Looping Structure

Loop Types [Cont.]

- **While Loop:**

In while loop, a condition is evaluated before processing a body of the loop. If a condition is true then and only then the body of a loop is executed. After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the same process is executed until the condition becomes false. Once the condition becomes false, the control goes out of the loop.

In while loop, if the condition is not true, then the body of a loop will not be executed, not even once.



Loop Types [Cont.]

Syntax

The syntax of a while loop is –

```
initialize loop counter;

    while(condition) {
            statement(s);
            loop counter
    }
```

Loop Types [Cont.]

Example:

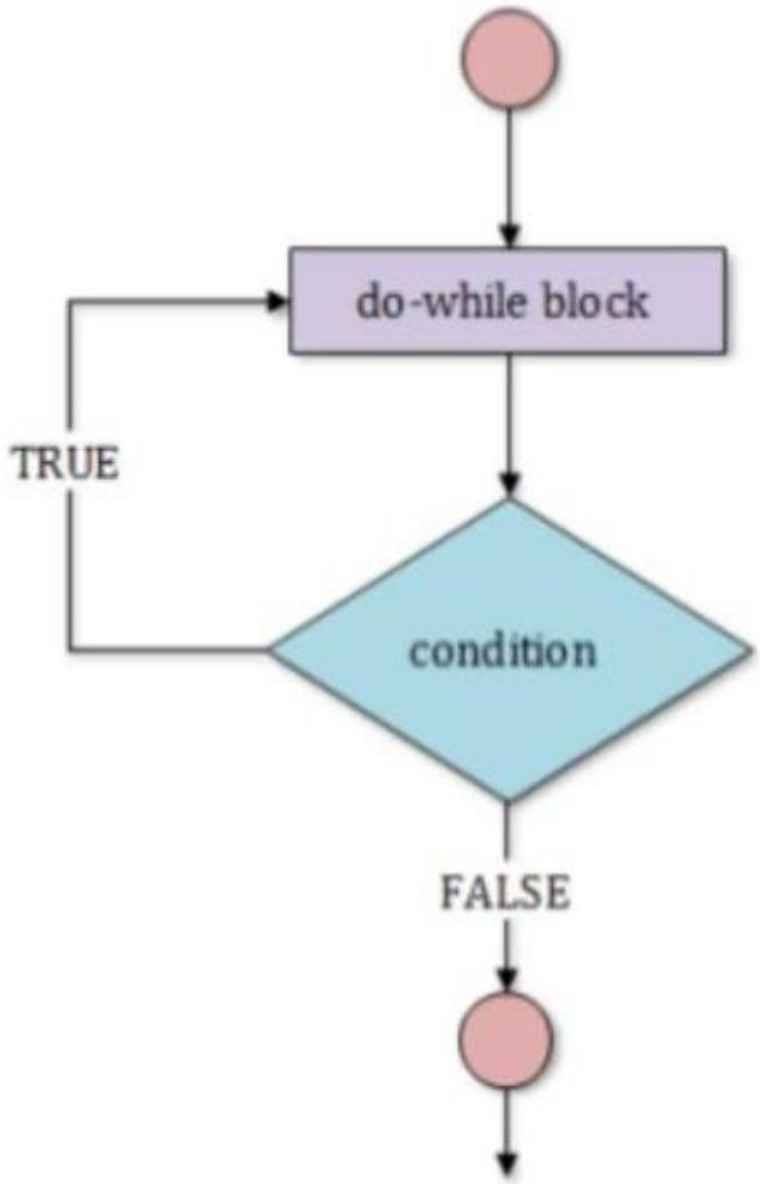
```c
#include <stdio.h>

// function main begins program execution
int main( void )
{
   unsigned int counter = 1; // initialization

   while ( counter <= 10 ) { // repetition condition
      printf ( "%u\n", counter ); // display counter
      ++counter; // increment
   } // end while
} // end function main
```

Loop Types [Cont.]

- ✓ **Do- while Loop:**

The do while repetition statement is similar to the while statement.

- The do…while statement tests the loop-continuation condition *after* the loop body is performed. Therefore, the loop body will be executed *at least once*. When a do…while terminates, execution continues with the statement after the while clause.
- It's not necessary to use braces in the do…while statement if there's only one statement in the body. However, the braces are usually included to avoid confusion between the while and do…while statements.



Loop Types [Cont.]

The syntax of while loop is as follows:

```
do {
 statements
} while (expression);
```

Loop Types [Cont.]

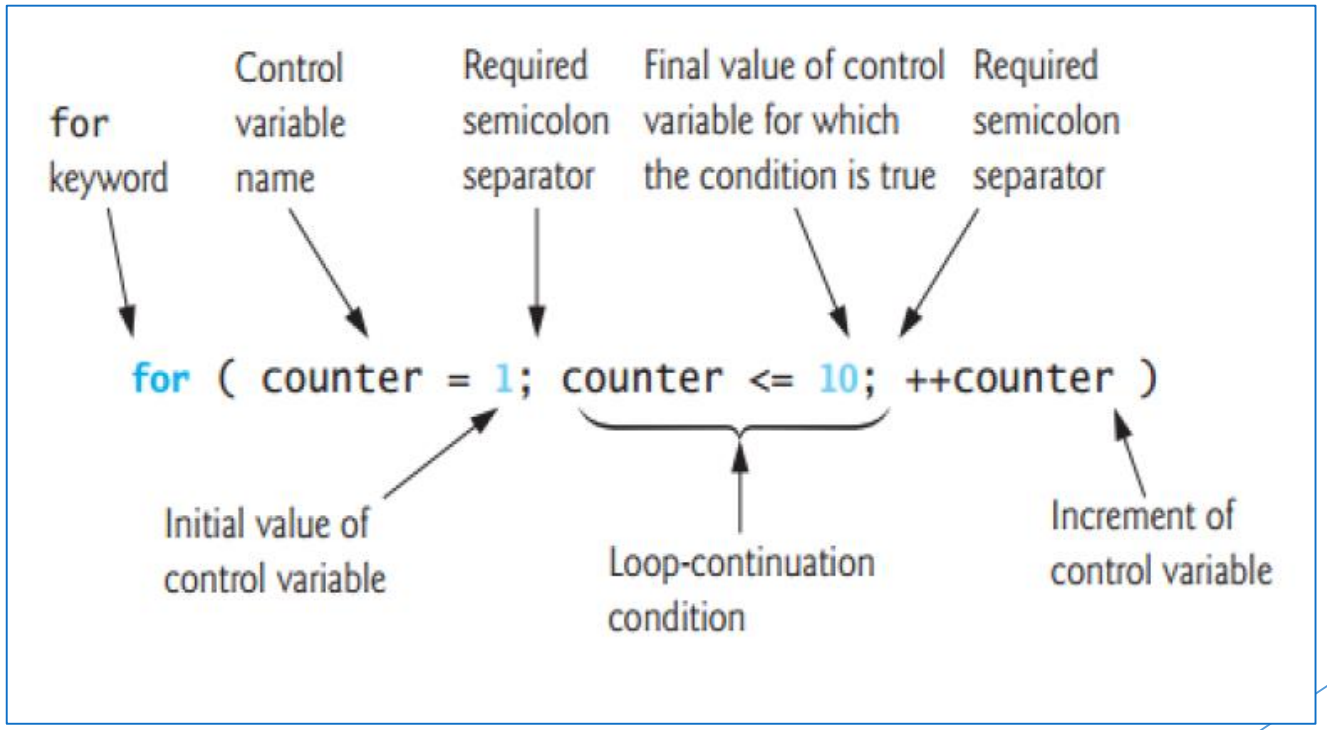
Examples:

```c
#include<stdio.h>
#include<conio.h>
int main()
{
    int num=1;
    do
    {
        printf("%d\n",3*num);
        num++;
    }while(num<=10);
    return 0;
}
```

Loop Types [Cont.]

- **For Loop:**

  A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.



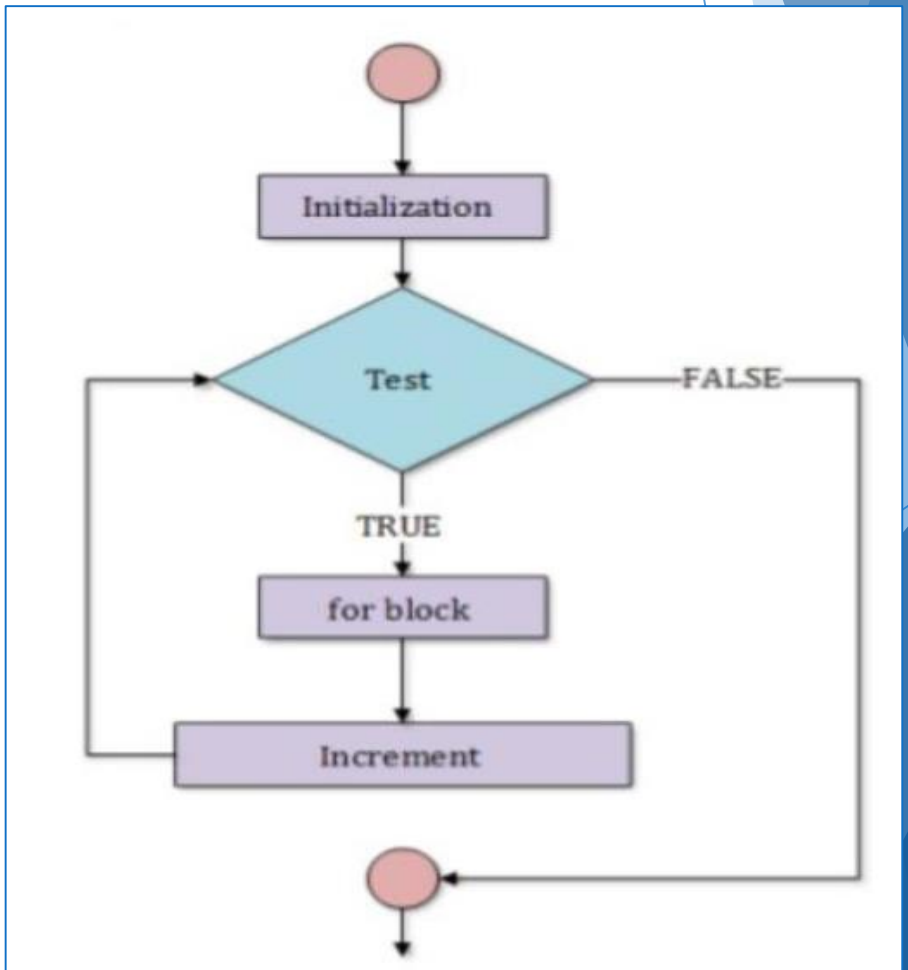
Loop Types [Cont.]

- **For Loop:**

  The for allows us to specify three things about a loop in a single line.

Syntax

```
for (initialization; condition; increment)
{
  statement(s);
}
```



Loop Types [Cont.]

Here is the flow of control in a 'for' loop −

- The **initialization** step is executed first, and only once. This step allows you to declare and initialize any loop control variables.
- Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
- After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update any loop control variables.
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

Loop Types [Cont.]

The initialization, loop-continuation condition and increment can contain arithmetic expressions. For example, if x = 2 and y = 10, the statement

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

is equivalent to the statement

```
for ( j = 2; j <= 80; j += 5 )
```

Loop Types [Cont.]

TASK:

Write a program to Calculate and Print Product of n odd integers. N will be inputted by user.

Loop Types [Cont.]

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

    int count,range;
    int product=1;
    scanf("%d",&range);
    printf( "\nThe odd numbers 1-15 are\n\n");

    for( count=1; count<=range; count+=2 ){
      product =count*product;
       printf("%d\n", count);
       }
    printf( "\nProduct is \n%d", product);
return 0;
}
```

Loop Types [Cont.]

TASK:

Write a program Logic to find ones complement of a binary number.