
UNIT-3

Topics to be covered in UNIT-3

Repetitive control structures – Pre-test and post-test loops, initialization and updation, event and counter controlled loops, while, do..while, for, break and continue statements, comma expression

Functions – User-defined functions, Function definition, arguments, return value, prototype, arguments and parameters, inner-function communication.

Standard functions – Math functions, Random numbers.

Scope – local global.

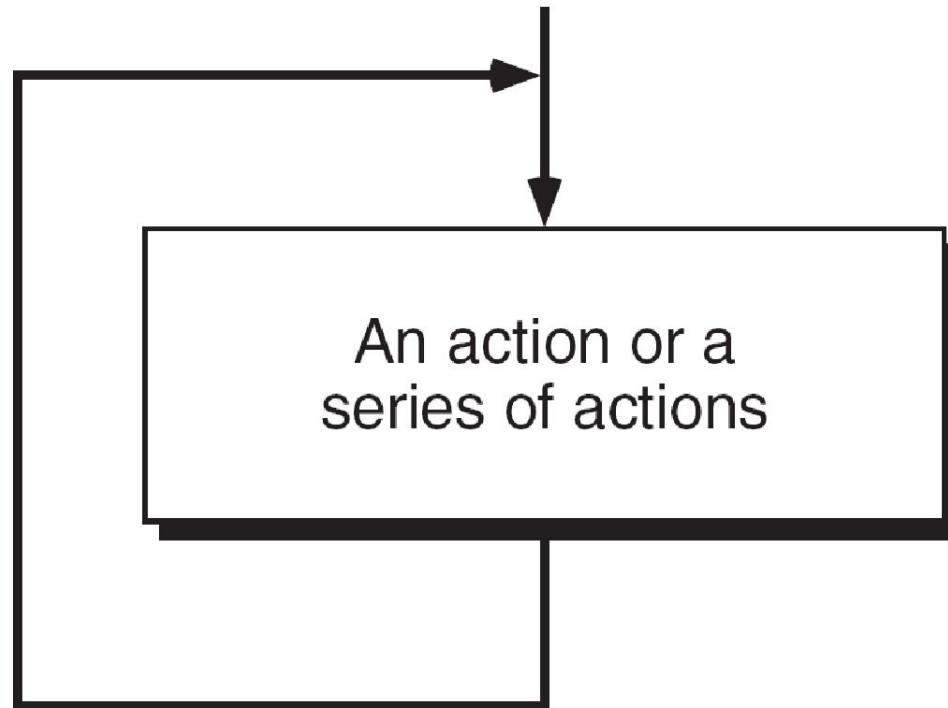
Parameter passing – Call by value and call by reference.

Recursive functions – Definition, examples, advantages and disadvantages.

Macros – Definition, examples, comparison with functions.

Concept of a loop

- The real power of computers is in their ability to repeat an operation or a series of operations many times.
- This repetition, called looping, is one of the basic structured programming concepts.
- Each loop must have an expression that determines if the loop is done.
- If it is not done, the loop repeats one more time; if it is done, the loop terminates.



Concept of a Loop

Pretest and Post-test Loops

- We need to test for the end of a loop, but where should we check it—before or after each iteration? We can have either a pre- or a post-test terminating condition.
- In a pretest loop , the condition is checked at the beginning of each iteration.
- In a post-test loop, the condition is checked at the end of each iteration.

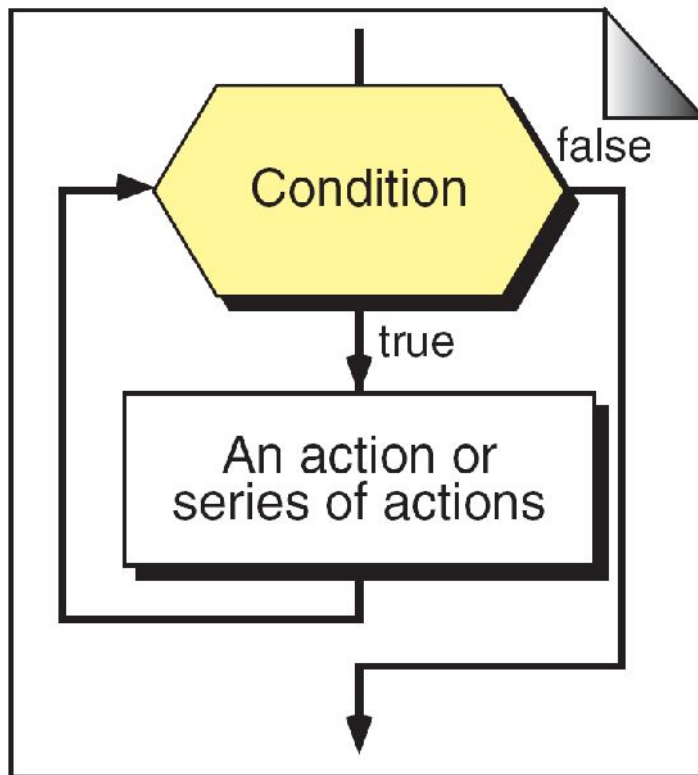
Note

Pretest Loop

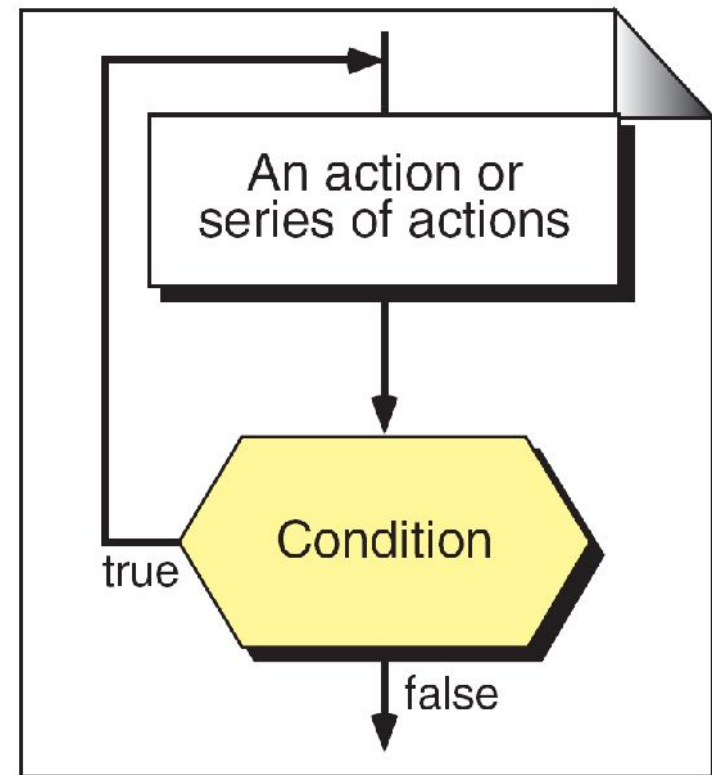
In each iteration, the control expression is tested first. If it is true, the loop continues; otherwise, the loop is terminated.

Post-test Loop

In each iteration, the loop action(s) are executed. Then the control expression is tested. If it is true, a new iteration is started; otherwise, the loop terminates.

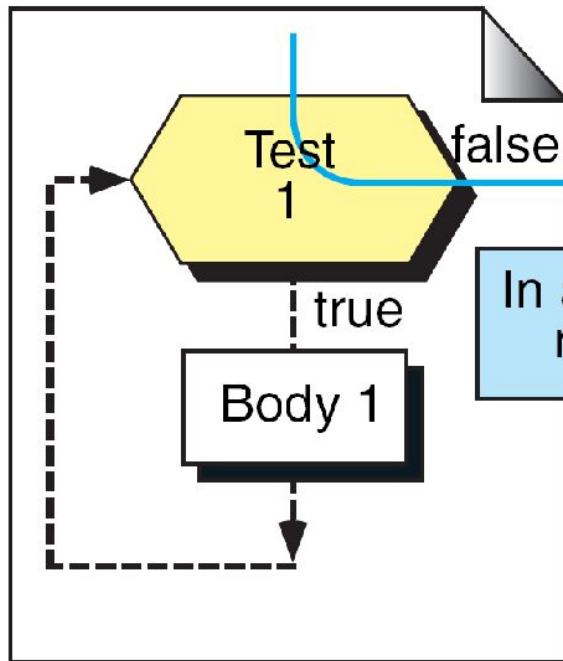


(a) Pretest Loop



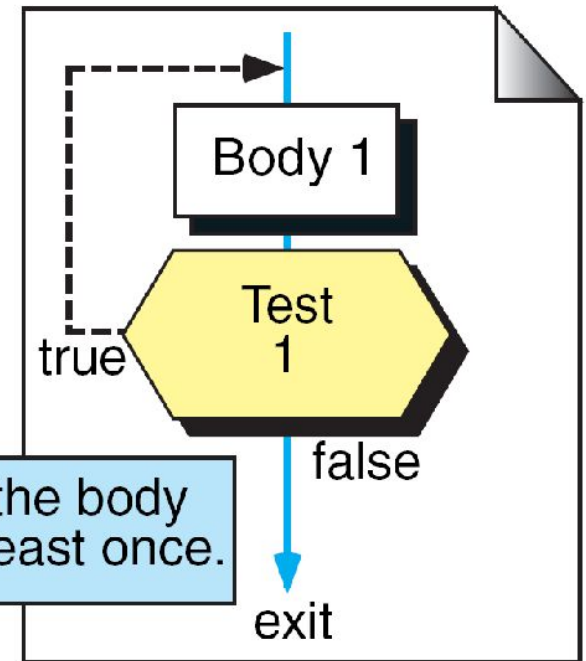
(b) Post-test Loop

Pretest and Post-test Loops



(a) Pretest

In a pretest loop, the body may not be executed.



(b) Post-test

In a post-test loop, the body must be executed at least once.

Minimum Number of Iterations in Two Loops

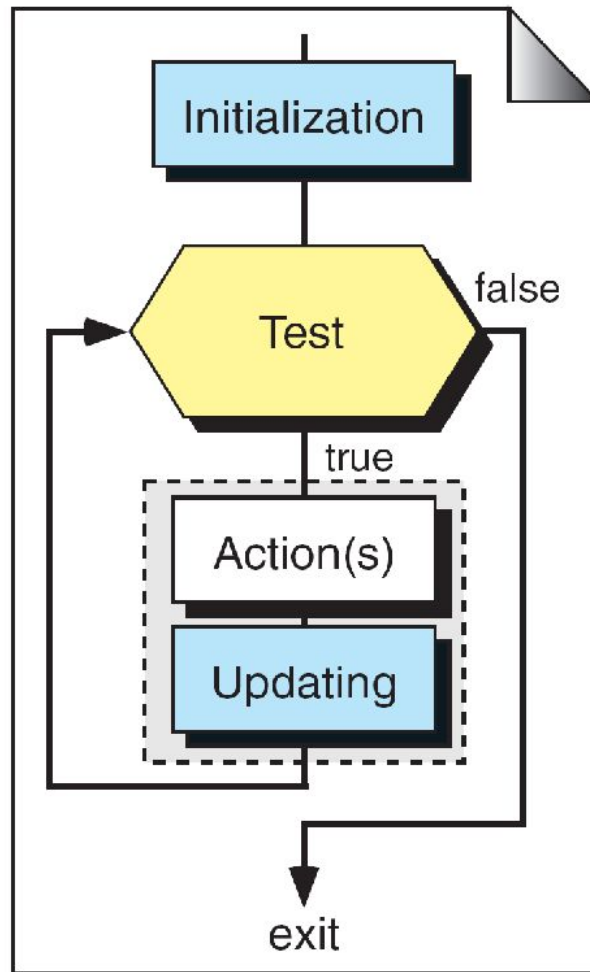
Initialization and Updating

- In addition to the **loop control expression**, two other processes, initialization and updating, are associated with almost all **loops**.

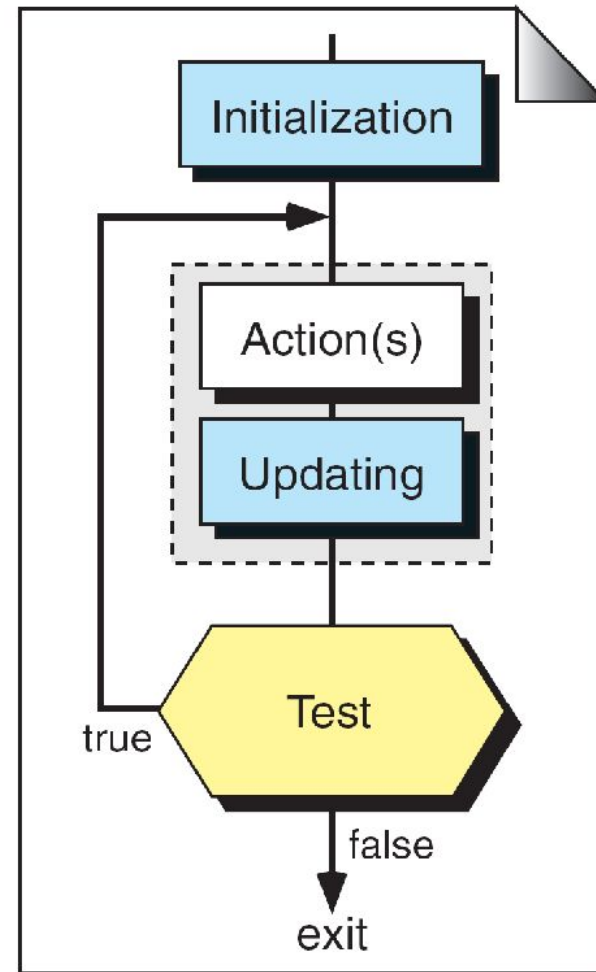
- **Loop Initialization**

- **Loop Update**

- Control expression is used to decide whether the loop should be executed or terminated.
- Initialization is place where you can assign some value to a variable.
- Variable's value can be updated by incrementing a value by some amount.



(a) Pretest Loop

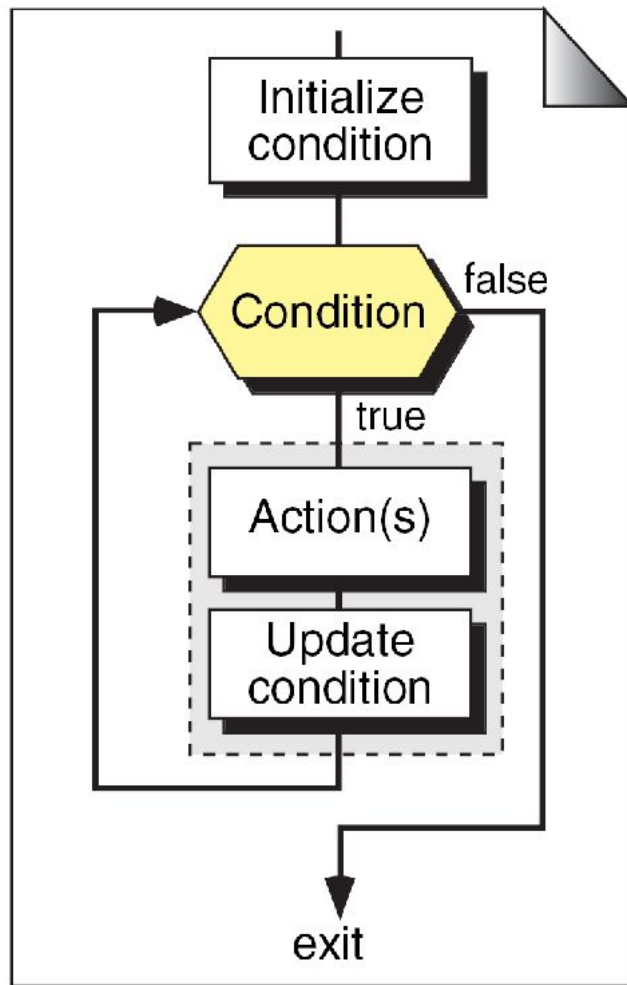


(b) Post-test Loop

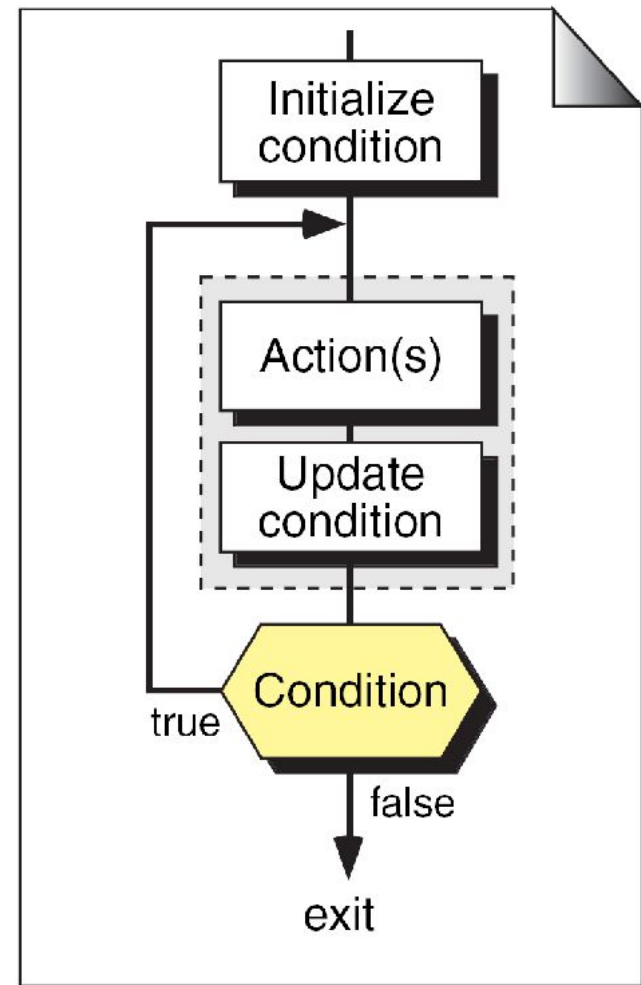
Loop Initialization and Updating

Event- and Counter-Controlled Loops

- All the possible expressions that can be used in a loop limit test can be summarized into two general categories:
 - 1. Event-controlled loops and**
 - 2. Counter-controlled loops.**
- In event-controlled loops, loop execution depends on the given condition.
- In counter-controlled loops, loop execution depends on the counter variable value.

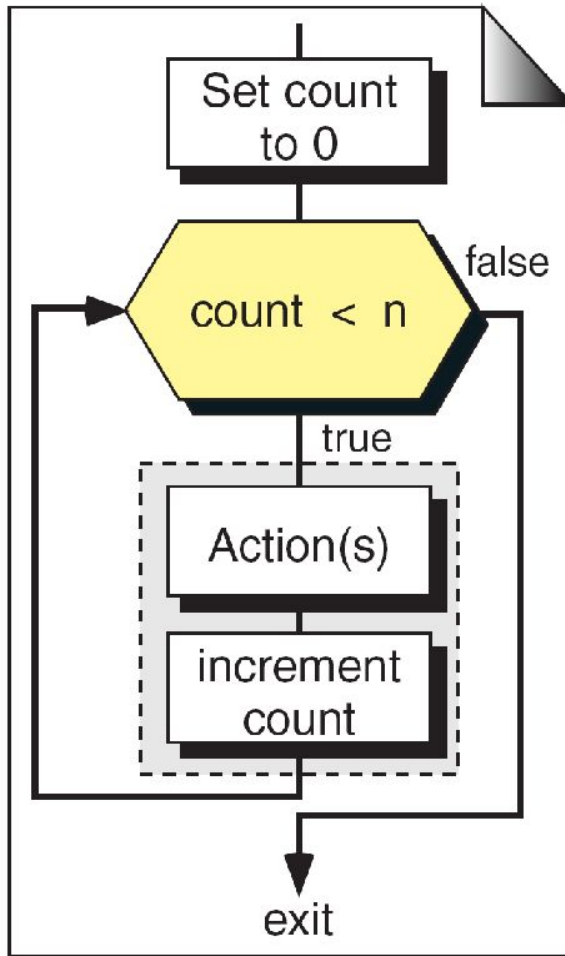


(a) Pretest Loop

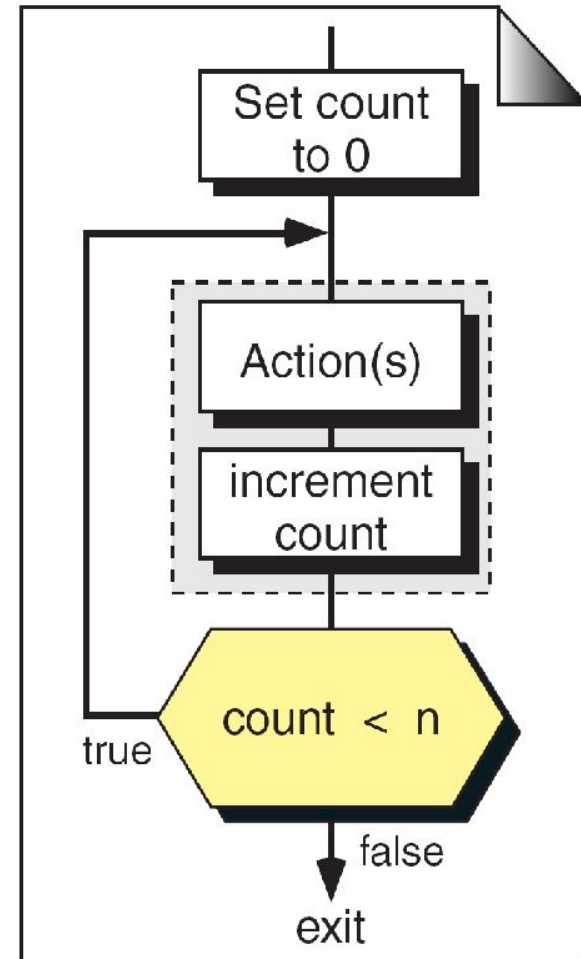


(b) Post-test Loop

Event-controlled Loop Concept



(a) Pretest Loop



(b) Post-test Loop

Counter-controlled Loop Concept

Loop Comparisons

Pretest Loop		Post-test Loop	
Initialization:	1	Initialization:	1
Number of tests:	$n + 1$	Number of tests:	n
Action executed:	n	Action executed:	n
Updating executed:	n	Updating executed:	n
Minimum iterations:	0	Minimum iterations:	1

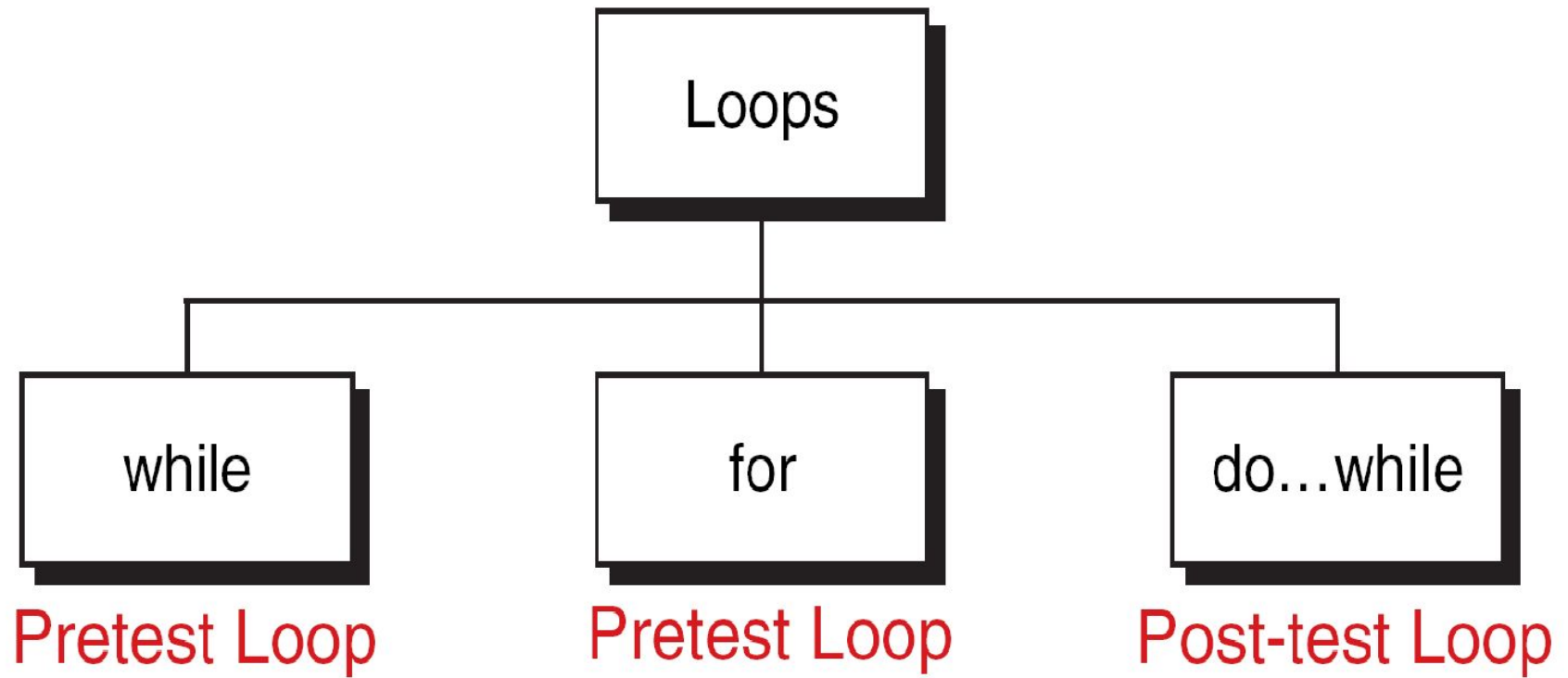
Loops in C

- C has three loop statements: the while, the for, and the do...while. The first two are pretest loops, and the third is a post-test loop.

- We can use all of them for event-controlled and counter-controlled loops.

A looping process, in general, would include the following four steps:

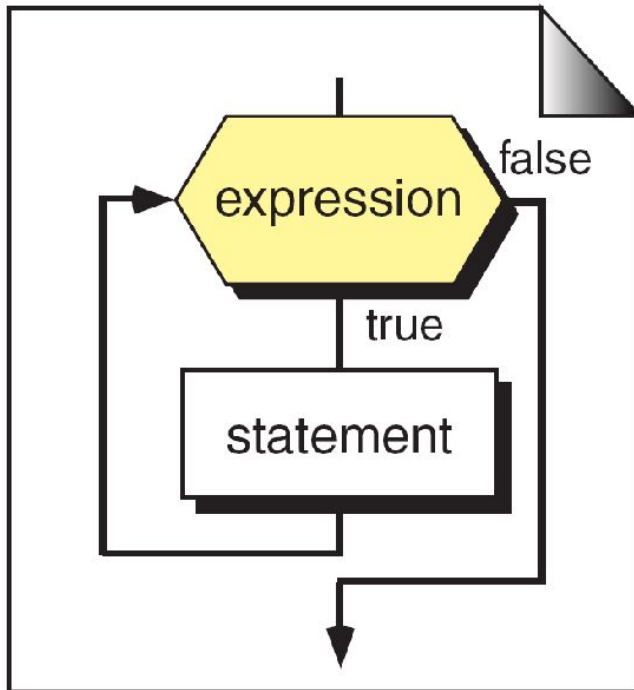
- Before a loop start, the loop control variable must be **initialized**; this should be done before the first execution of loop body.
- Test for the specified condition for execution of the loop, known as **loop control expression**.
- **Executing** the body of the loop, known as actions.
- **Updating** the loop control variable for performing next condition checking.



C Loop Constructs

while

- ❑ The "while" loop is a generalized looping structure that employs a variable or expression for testing the condition.
- ❑ It is a repetition statement that allows an action to be repeated while some conditions remain true.
- ❑ The body of **while** statement can be a **single** statement or **compound** statements.
- ❑ It **doesn't** perform even a single operation if **condition fails**.

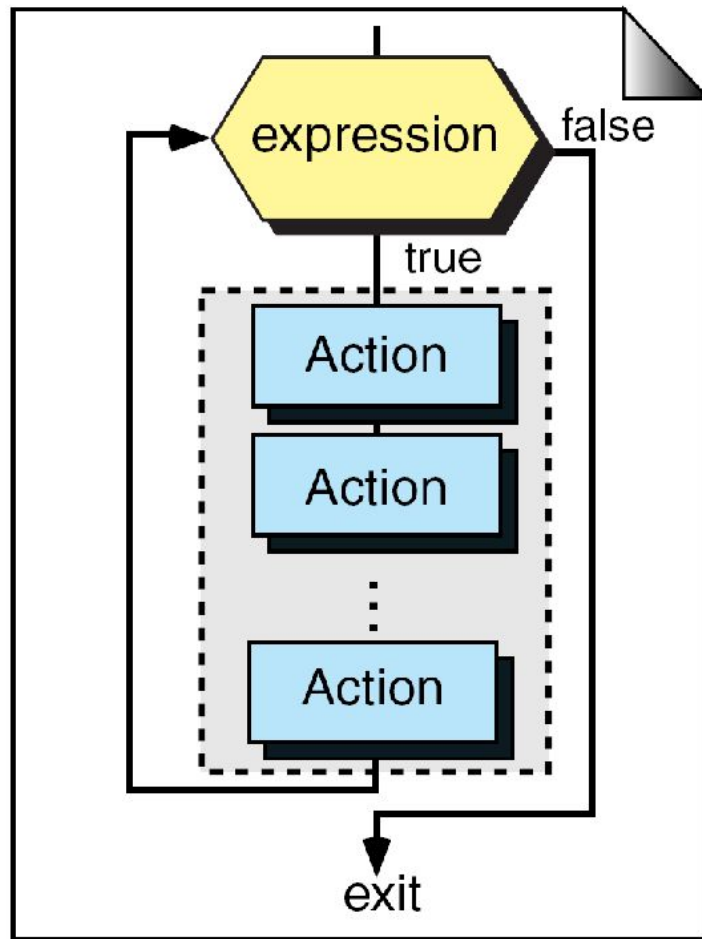


(a) Flowchart

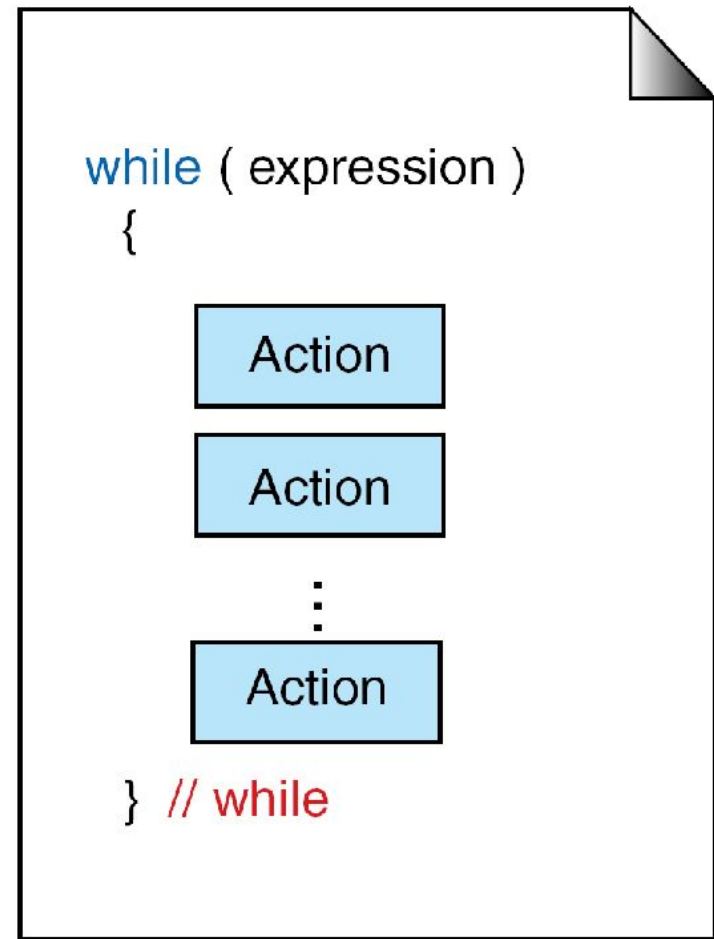
```
while (expression)  
    statement
```

(b) Sample Code

The while Statement



(a) Flowchart



(b) C Language

Compound while Statement

Example 1: To print 1 to 10 natural numbers

```
#include<stdio.h>
main()
{
    int i;
    i=1;
    while (i<=10)
    {
        printf(“%d\n”,i);
        i++;
    }
}
```

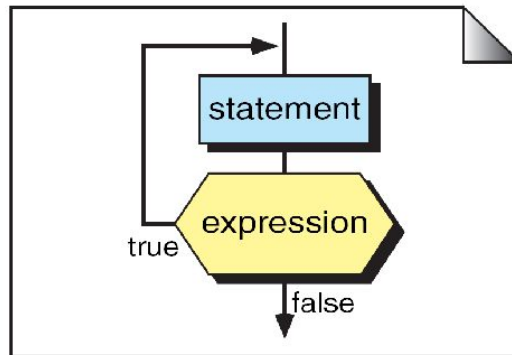
Example 2: To print the reverse of the given number.

```
void main()
{
    int n, rem, rev = 0;
    printf("\n Enter a positive number: ");
    scanf("%d",&n);
    while(n != 0)
    {
        rem = n%10;
        rev = rev*10+rem;
        n = n/10;
    }
    printf("The reverese of %d is %d",n,rev);
}
```

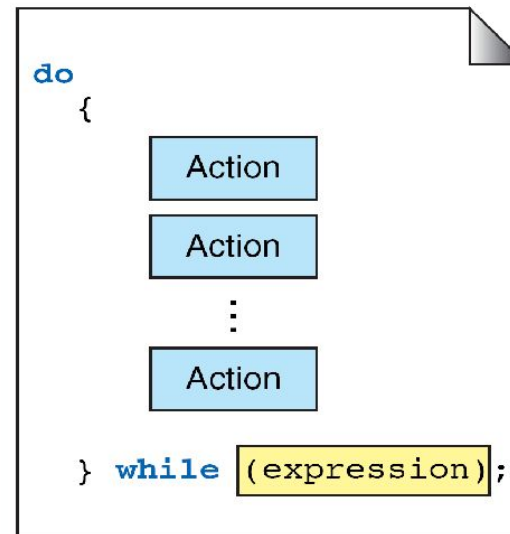
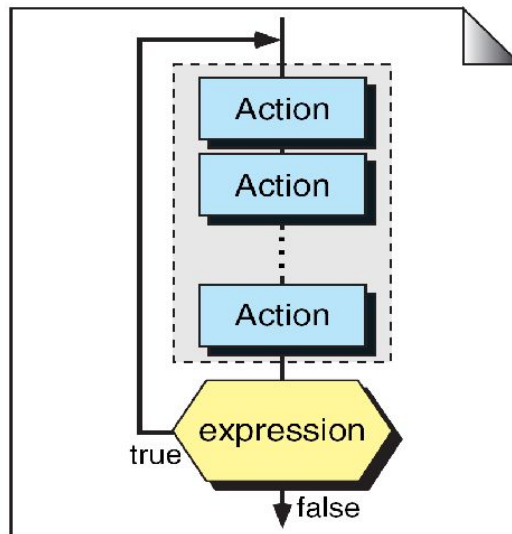
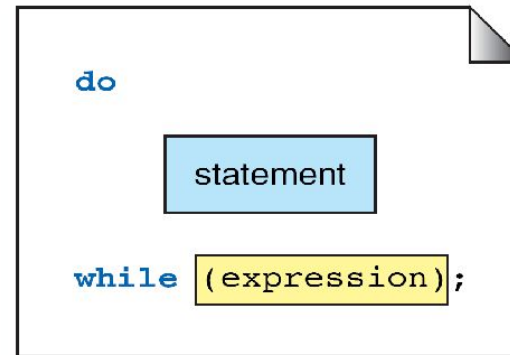
do while

- ❑ The “do while” loop is a repetition statement that allows an action to be done at least once and then condition is tested.
- ❑ On reaching **do** statement, the program proceeds to evaluate the body of the loop first.
- ❑ At the end of the loop, **condition** statement is **evaluated**.
- ❑ If the condition is **true**, it evaluates the body of the loop once **again**.
- ❑ This process **continues** up to the condition becomes false.

Flowchart



Sample Code



do...while Statement

Example 3: To print fibonacci sequence for the given number.

```
#include<stdio.h>
main()
{
    int a=0,b=1,c,i;
    i=1;
    printf("%d%d",a,b);
    do
    {
        c=a+b;
        i++;
        printf("%3d",c);
        a=b;
        b=c;
    }while(i<=10);
}
```


Example 4: To print multiplication table for 5.

```
#include <stdio.h>
void main()
{
    int i = 1, n=5;
    do
    {
        printf(“ %d * %d = %d “, n, i, n*i);
        i = i + 1;
    } while ( i<= 5);
}
```

//Program to print 5 4 3 2 1 using while and do...while


```
1  /* Demonstrate while and do...while loops.
2      Written by:
3      Date:
4  */
5  #include <stdio.h>
6
7  int main (void)
8  {
9      // Local Declarations
10     int loopCount;
11
12     // Statements
13
14     loopCount = 5;
15     printf("while loop      : ");
16     while (loopCount > 0)
17         printf ("%3d", loopCount--);
18     printf ("\n\n");
```

```
19     loopCount = 5;
20     printf("do...while loop: ");
21     do
22         printf ("%3d", loopCount--);
23     while (loopCount > 0);
24     printf("\n");
25     return 0;
26 } // main
```

Results


```
while loop      :    5    4    3    2    1
```

```
do...while loop:    5    4    3    2    1
```



```
while (false)
{
    printf("Hello World");
} // while
```

```
do
{
    printf("Hello World");
} while (false);
```



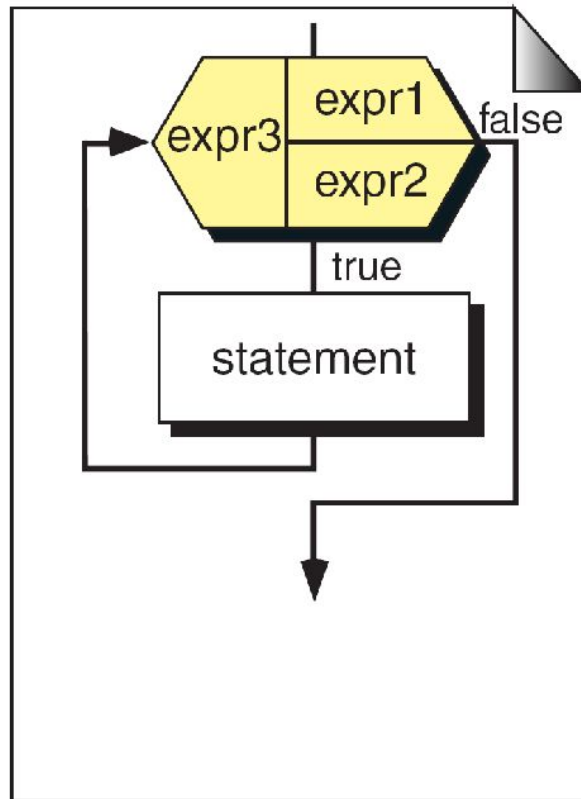
Pre- and Post-test Loops

for

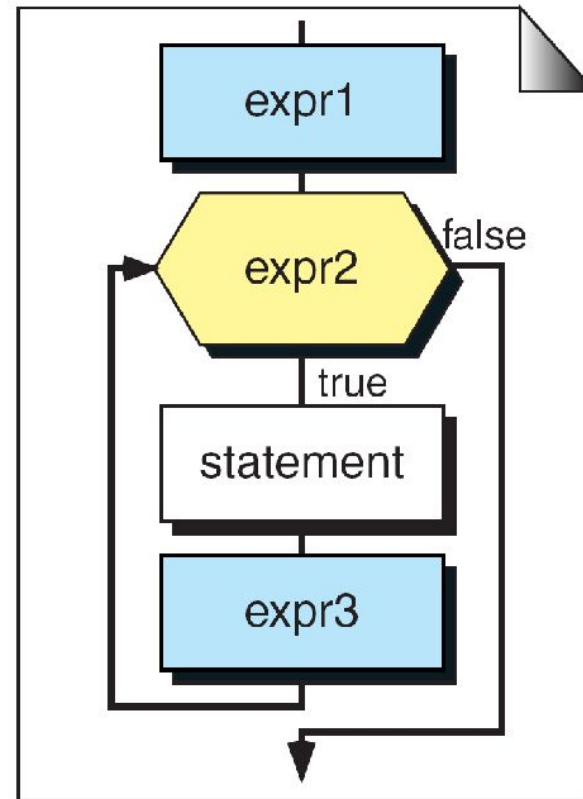
- A for loop is used when a loop is to be executed a **known number of times**.
- We can do the same thing with a while loop, but the for loop is easier to read and more natural for counting loops.

General form of the for is:

```
for( initialization; test-condition; updation)
{
    Body of the loop
}
```



(a) Flowchart

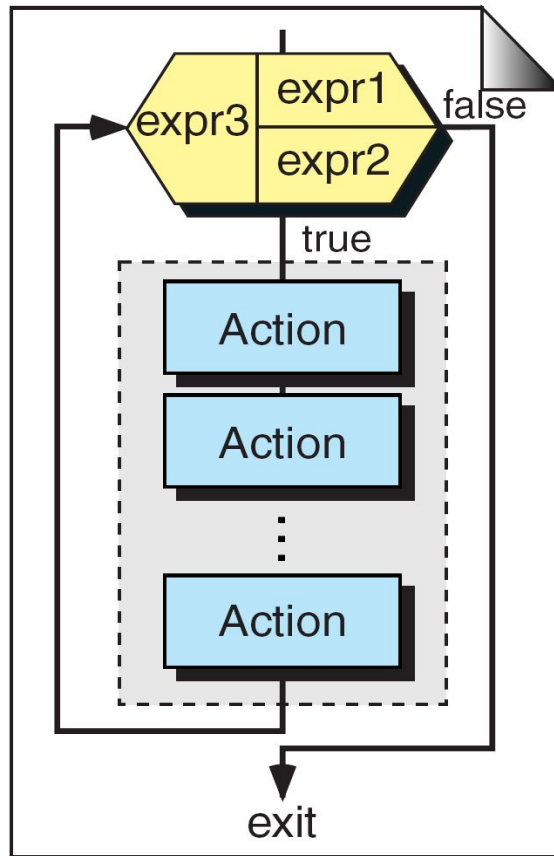


(b) Expanded Flowchart

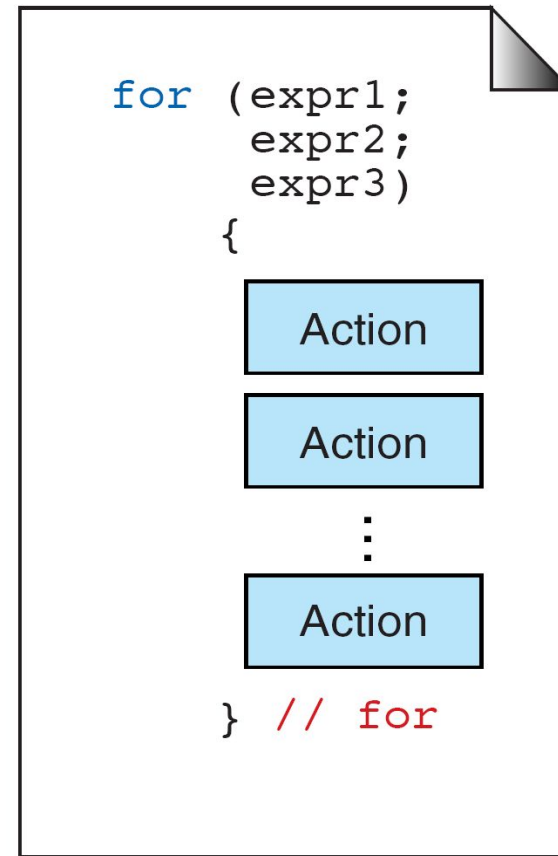
```

for (expr1; expr2; expr3)
    statement
  
```

***for* Statement**

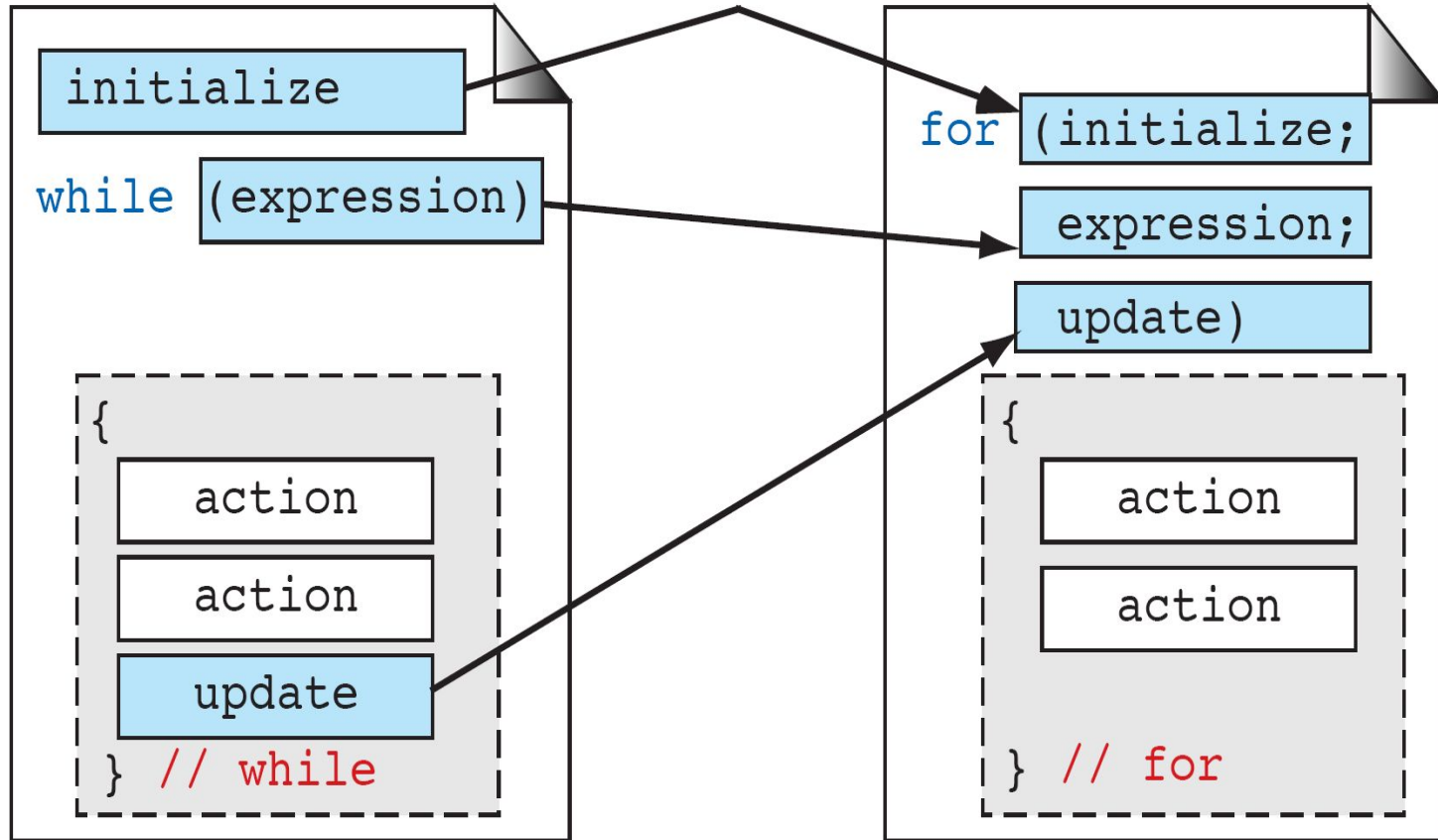


(a) Flowchart



(b) C Language

Compound *for* Statement



Comparing *for* and *while* Loops


```
1  /* Print number series from 1 to user-specified limit.
2      Written by:
3      Date:
4  */
5  #include <stdio.h>
6  int main (void)
7  {
8  // Local Declarations
9      int limit;
10
11 // Statements
12     printf ("\nPlease enter the limit: ");
13     scanf ("%d", &limit);
14     for (int i = 1; i <= limit; i++)
15         printf("\t%d\n", i);
16     return 0;
17 } // main
```

Results:

Please enter the limit: 3

1
2
3

A Simple Nested *for* Loop

```
1  /* Print numbers on a line.  
2      Written by:  
3      Date:  
4  */  
5  #include <stdio.h>  
6  
7  int main (void)  
8  {  
9      // Statements  
10     for (int i = 1; i <= 3; i++)  
11     {  
12         printf("Row %d: ", i);  
13         for (int j = 1; j<= 5; j++)  
14             printf("%3d", j);  
15         printf("\n");  
16     } // for i  
17     return 0;  
18 } // main
```

Results:

Row 1: 1 2 3 4 5

Row 2: 1 2 3 4 5

Row 3: 1 2 3 4 5

We can write the for loop in the following ways:

Option 1:

```
for (k= 1; k<= 10 ;)  
{  
    printf("%d", k);  
    k = k + 1;  
}
```

Here the increment is done within the body of the for loop and not in the for statement. Note that the semicolon after the condition is necessary.

Option 2:

```
int k = 1;  
for (; k<= 10; k++);  
{  
    printf(", k);  
}
```

Here the initialization is done in the declaration statement itself, but still the semicolon before the condition is necessary.

Option 3: The infinite loop

One of the most interesting uses of the for loop is the creation of the infinite loop. Since none of the three expressions that form the for loop are required, it is possible to make an endless loop by leaving the conditional expression empty.

For example: **for (; ;)**

```
printf("The loop will run forever\n");
```

Actually the for (; ;) construct does not necessarily create an infinite loop because C's break statement, when encountered anywhere inside the body of a loop, causes immediate termination of the loop.

Program control then picks up the code following the loop, as shown here:

```
for (; ;)
```

```
{
```

```
    ch = getchar( );          /* get a character */
```

```
    if (ch == 'A')
```

```
        break ;
```

```
}
```

```
printf ("you typed an A");
```

This loop will run until A is typed at the keyboard.

Option 3: For loop with no body

A statement, as defined by the C syntax, may be empty.

This means that the body of the for may also be empty.

This fact can be used to improve the efficiency of certain algorithms as well as to create time delay loops.

The following statement shows how to create a time delay loop using a for loop:

```
for (t = 0; t < SOME_VALUE; t++);
```

-
- The operator comma , is used to separate the more than one expressions.
 - A pair of expressions separated by a comma is evaluated left to right, and the type and value of the result are the type and value of the right operand.
 - Thus, in a for statement, it is possible to place multiple expressions in the various parts.



Nested Comma Expression

Comparison of *while* and *do...while*

```
1  /* Demonstrate while and do...while loops.
2      Written by:
3      Date:
4  */
5  #include <stdio.h>
6
7  int main (void)
8  {
9      // Local Declarations
10     int loopCount;
11     int testCount;
12
13     // Statements
14     loopCount = 1;
15     testCount = 0;
16     printf("while loop:      ");
17     while (testCount++, loopCount <= 10)
18         printf("%3d", loopCount++);
19     printf("Loop Count:      %3d\n", loopCount);
```


break

- When a break statement is enclosed inside a block or loop, the loop is immediately exited and program continues with the next statement immediately following the loop.
- When loop are nested break only exit from the inner loop containing it.

The format of the break statement is:

```
while (expr)
```

```
{  
  ...
```

```
  break;
```

```
  ...
```

```
} // while
```

```
do
```

```
{  
  ...
```

```
  break;
```

```
  ...
```

```
} while (expr);
```

```
for (expr1; expr2; expr3)
```

```
{  
  ...
```

```
  break;
```

```
  ...
```

```
} // for
```


Example 6: Program to demonstrate **break** statement.

```
#include<stdio.h>
main()
{
    int i;
    i=1;
    while(i<=10)
    {
        if(i==8)
            break;
        printf(“%d\t”,i);
        i=i+1;
    }
    printf(“\n Thanking You”);
}
```

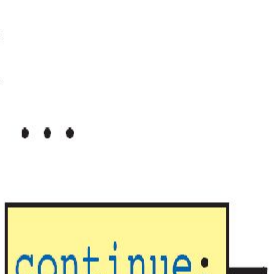
continue

- When a continue statement is enclosed inside a block or loop, the loop is to be continued with the next iteration.
- The continue statement tells the compiler, skip the following statements and continue with the next iteration.
- **The format of the continue statement is:**

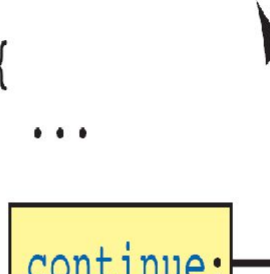
```
while (expr)
{
    ...
    continue;
    ...
} // while
```



```
do
{
    ...
    continue;
    ...
} while (expr);
```



```
for (expr1; expr2; expr3)
{
    ...
    continue;
    ...
} // for
```



Example 5: Program to demonstrate **continue** statement.

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

```
main()
```

```
{
```

```
    int i;
```

```
    for(i=1;i<=5;i++)
```

```
    {
```

```
        if(i == 3)
```

```
            continue;
```

```
        printf(" %d",i);
```

```
    }
```

```
}
```