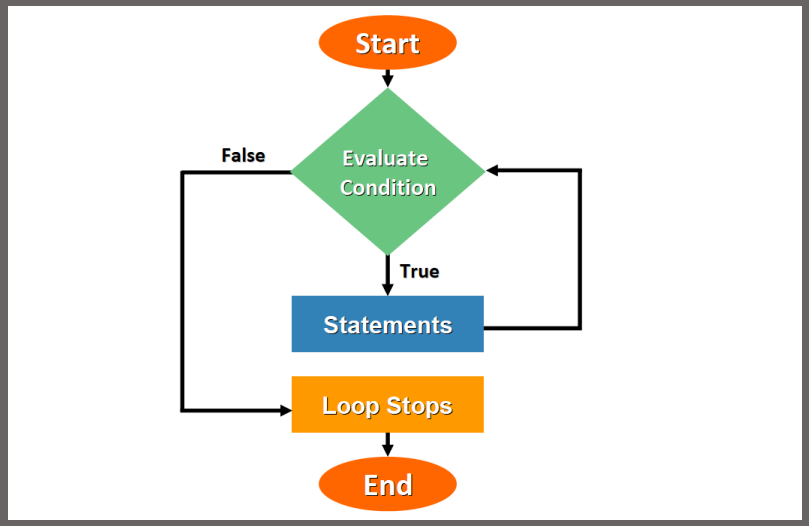
Repetition Control Structures

Lecture 06 – ICT1132



Piyumi Wijerathna
Department of ICT
Faculty of Technology

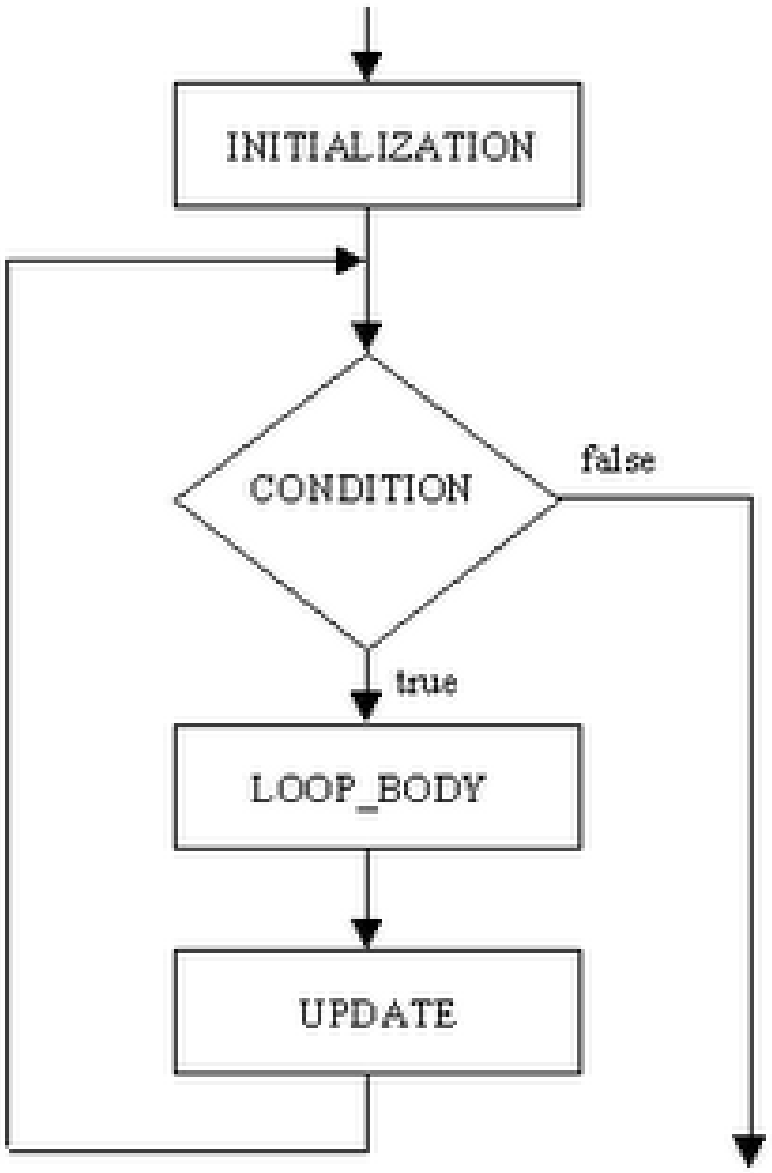
Overview

Repetition Structure

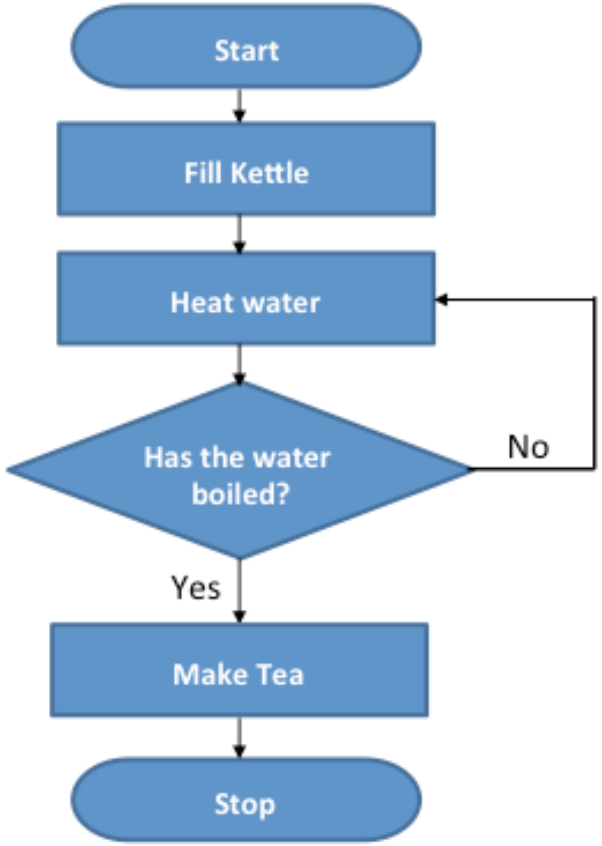
- ✓ while loop
- ✓ for loop
- ✓ do-while loop
- ✓ nested repetition statements

What is a Repetition Structure?

- Block of code needs to be **executed several number of times**.
  - Ex: Display numbers from 1 to 1000
- A repetition structure executes a statement or group of statements multiple times while some condition remains true.
- Also known as Iteration / Loop Structure.


INITIALIZATION
CONDITION
false
true
LOOP_BODY
UPDATE

Example (Making the Tea)



teachwithict.weebly.com

Pre-test & Post-test Loops

- **Pre-test** loops are entrance controlled loops.
  - You **execute the loop body after evaluating the test**.
  - Loop body can be executed zero or more times.

- **Post-test** loops are exit controlled loops.
  - You **test the loop after executing the entire loop** body.
  - Loop body can be executed one or more times.

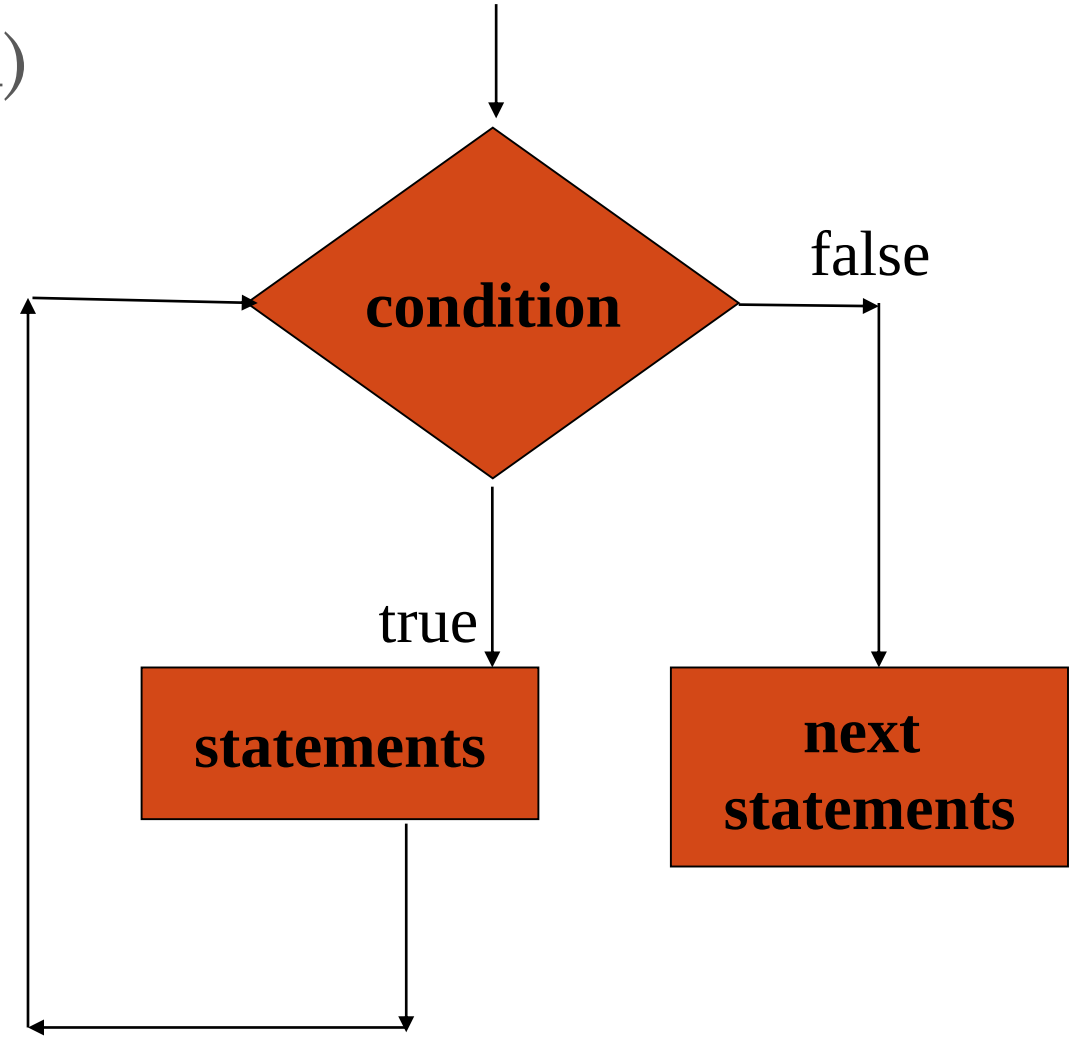
While Loop

The While Loop

- A while loop in C programming **repeatedly executes the body** of the loop as long as a **given condition is true.**
- When condition becomes false, control passes to the next line of code immediately after the loop.
- It **tests the condition before executing** the loop body.

Syntax of While Loop

```
while (condition)
{
    statements
}
```



While Loop Example

```
# include <stdio.h>
void main()
{
        int a=0, b=30;
        while (a < 10)
        {
                printf("%d  ",  a);
                a++;
        }
        printf("\n%d ",  b);
}
```

Output

0 1 2 3 4 5 6 7 8 9
30

More about While Loop

- The while statement body may contain single or a compound statement.
- The curly braces are unnecessary if only a single statement is being repeated.

```
int count=5;
while(count>0)
        count--;
printf("The value of count is: %d", count);
```

Output

The value of count is:0

Counter Controlled Repetition

- Number of **repetitions is known before begins the loop** execution.
- A **control variable** is used to count the number of repetitions.
- The **control variable is incremented/decremented each time** the loop body is performed (usually by 1).
- The **repetition/loop terminates** when the counter exceeds its number.

Counter Controlled Repetition

- Counter-controlled repetition requires:
  1. The **name** of a control variable.
  2. The **initial value** of the control variable.
  3. The **control variable needs to be modified** each time within the loop, by incrementing or decrementing.
  4. The **condition** that tests for the final value of the control variable.

Counter Controlled Repetition with while loop

What is the loop control variable here?

```
# include <stdio.h>
void main()
{
        int a=0, b=30;                  // initialization

        while (a < 10)                  //repetition condition
        {
                printf("%d ",  a);      // display a
                a++;                    // increment a
        }                               // end while
        printf("%d ",  b);              // display b
}                                       //end function main
```

Output

0 1 2 3 4 5 6 7 8 9 **30**

Important about while loop

- All variables in the boolean expression (condition) must be initialized prior to the loop.
- At least one variable in the expression must be the control variable.
- The boolean expression is tested prior to entering the loop and before each repetition of the loop body.
- The entire loop body is executed if the boolean expression is true.
- Loop body will not be executed, when the boolean expression is initially false.

Exercise - What is the output?

```
# include <stdio.h>
int main()
{
        int x = 1, y = 0, z;
        while ( x  <=  5)  {
                ++ x;
                y++;
                z = x * y;
                printf("%d ",  z);
        } // end while
return 0;
} //end function main
```

Any Difference between pre & post increments here?

Output

2 6 12 20 30

Can you display the answer into a single column?

Output

2
6
12
20
30

Sentinel Controlled Repetition

- When **no indication** is given of **how many times the loop should execute** (No counter-controller), a sentinel value is used to terminate the loop.

  Ex : type -99 to terminate entering of marks.

- A loop should have a statement to obtain this value in each loop iteration.

Example – Sentinel Control

- Input a list of numbers from the keyboard and find the average. The list is terminated when the value -99 is entered.

```
#include <stdio.h>
int main()
{
   int number, sum=0, count=0;      //Initialization

   printf("Enter a list of integers terminated by -99");
   scanf("%d",&number);                //take the first number

   while (number != -99)     //Loop to add and count values
   {
        sum = sum + number;
        ++count;
        scanf ("%d",&number);          //Read the next number
   }

   //Calculate and print average
   printf("The average is: %.2f", sum/(float)count);
   return 0;
}
```

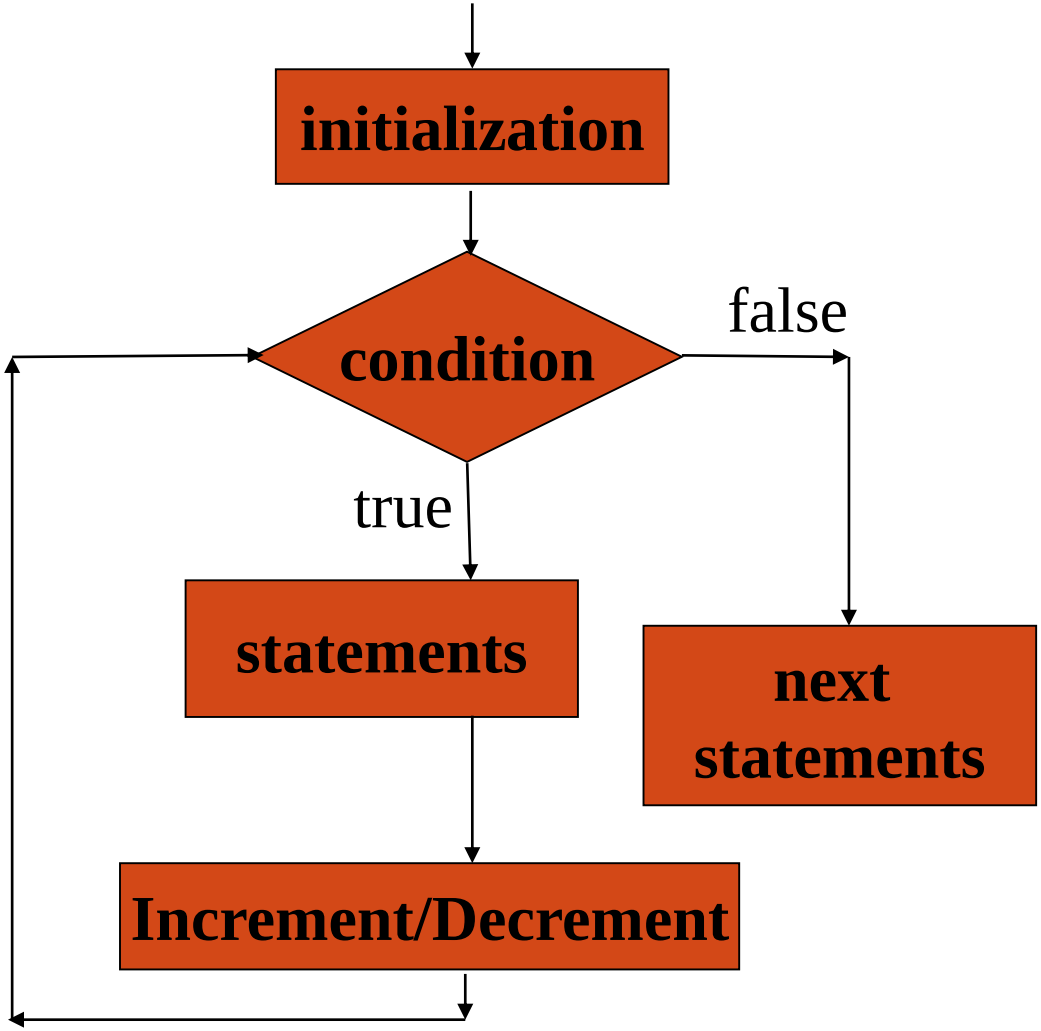
Output

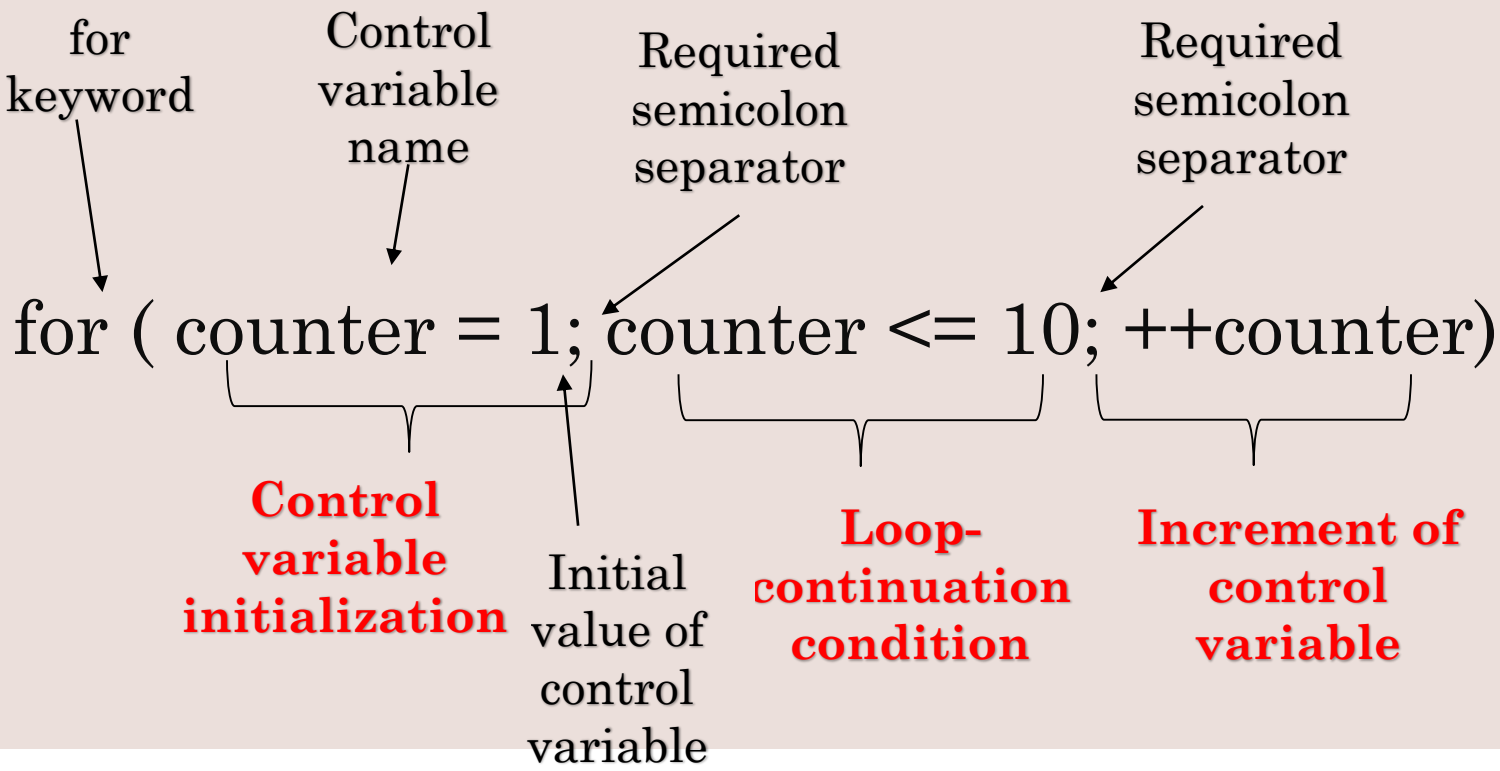
```
Enter  value: 75
Enter  value: 23
Enter  value: 76
Enter  value: 48
Enter  value: 65
Enter  value: 29
Enter  value: 92
Enter  value: 71
Enter  value: 19
Enter  value: 2
Enter  value: -99
The Average is: 50.00
```

For Loop

Syntax of for Loop

```
for (initialization; condition; update statement)
{
    statements
}
```




for keyword
Control variable name
Required semicolon separator
Required semicolon separator
for (counter = 1; counter <= 10; ++counter)
Control variable initialization
Initial value of control variable
Loop-continuation condition
Increment of control variable

```
for ( counter = 1; counter <= 10; ++counter){
    //statements
}
```

1.The ***initialization*** step is **executed first**, and **only once**. This step allows you to declare and initialize any loop control variables. This statement can be kept blank with semicolon.

2. 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.

3. After the body of the 'for' loop executes, the flow of control jumps back up to the ***increment/decrement*** statement. This is usually an expression that **increments or decrements the loop control variable**. This statement can be kept blank, as long as a semicolon appears after the condition.

4. The ***condition*** is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment/decrement step, and then again condition). After the condition become false, the 'for' loop terminates.

```
#include<stdio.h>
int main(){
        int i=0;
        for(;i<5;){
            printf("%d ",i);
            i++;
        }
    return 0;
}
```

Counter Controlled Repetition with for loops

```
# include <stdio.h>

void main()
{
        int x; // define counter variable

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

Output

1 2 3 4 5 6 7 8 9 10

Comparison of for and while Loops

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

```
int i=1;
while (i <= 10)
{
        printf("%d \n",i );
        i++;
}
```

Example

- Print the numbers from 100 to 10 as follows;
  100 90 80 70 --------- 10

```
for ( int i=100 ; i>=10 ; i = i-10 )
{
    printf("%d  ",i );
}
```

i -=10

More on for Loops

- All three expressions that are part of the for loop are optional. The semicolons are compulsory → for(; ;)
- Without **iteration**(increment/decrement) or **condition** you will have an infinite loop.

- Both declaration and Initialization of control variable can be done within the header of the for loop.

```
#include<stdio.h>
int main(){
        int i;
        for(i=0;i<5;){
            printf("%d ",i);
            //i++;
        }
    return 0;
}
```

- It is easy to use for loop when you know exactly how many times the loop body is to be executed.

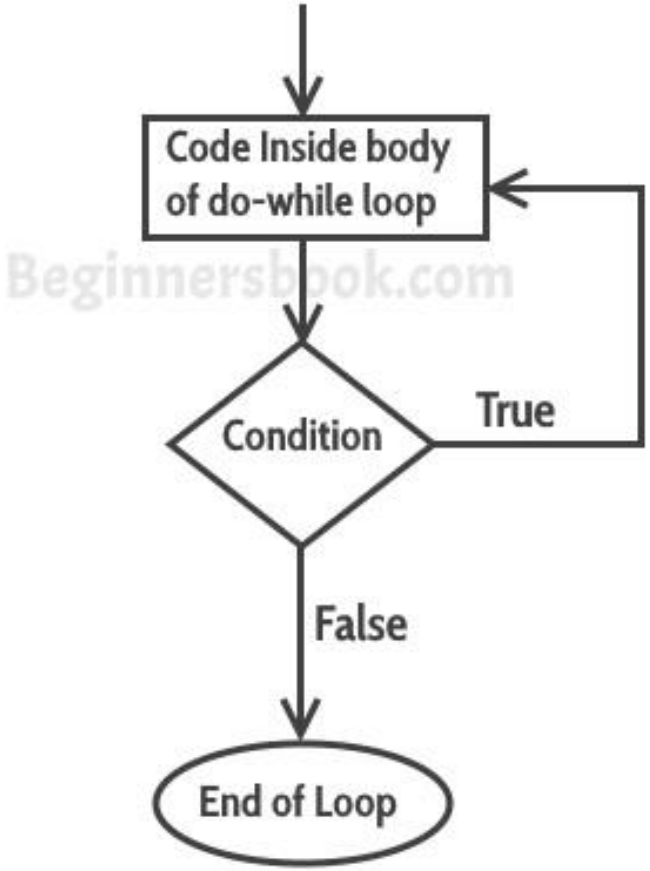
do while Loop

Do While Loop

- It is more like a while statement, except that the **test condition is performed after the loop body**.
- This is a post test loop.
- Hence the loop body will be executed at least once.

Syntax of do while loop

```
do{
   statement;
}while (condition);
```



<u>Ex:</u>

```
do
{
    statement 1;
    statement 2;
}while (condition);
```

Counter Controlled Repetition with do-while loops

```
# include <stdio.h>

void main()
{
        int x =1; // initialize the counter variable

        do{
                printf("%d ", x);
                x++;
        } while (x <= 6);
}
```

Do not forget semicolon

Output

1 2 3 4 5 6

Sentinel Control Repetition with do-while loops

```
// Program to add numbers until user enters zero

#include <stdio.h>
int main()
{
    double number, sum = 0;

    do                  // loop body is executed at least once
    {
        printf("Enter a number: ");
        scanf("%lf", &number);
        sum += number;
    } while(number != 0.0);

    printf("Sum = %.2f",sum);
return 0;
}
```

```
Enter a number: 12
Enter a number: 45
Enter a number: 10
Enter a number: 63
Enter a number: 0
Sum = 130.00
```

Jump Statements (break & continue)

“break” Statement with Loops

- The break statement causes the immediate termination of the execution of the loop body completely **(exit the whole loop).**
- It will continue the execution of the **first statement after the loop**.
- Common uses of the break statement are to escape early from a loop.

```
#include <stdio.h>

int main () {

   int a = 10;
   while( a < 20 ) {

      if( a == 15) {
          break; //terminate the  loop
      }

      printf("value of a: %d\n", a);
      a++;

   } //end while
    return 0;
}
```

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
```

“continue” Statement with Loops

- The continue statement causes the immediate **termination of the current iteration** of the loop body (skip remaining statements of the body of the current iteration).
- It will continue the execution of the loop from the next iteration.

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

    int a = 10;

    while( a < 20 )
    {
        if( a == 15) {
            a++;
            continue; //skip the iteration
        }

        printf("value of a: %d\n", a);
        a++;

    } //end while
}
```

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Nested Loops

Nesting of Loops

- C programming allows to use one loop inside another loop

Outer Loop

Inner Loop

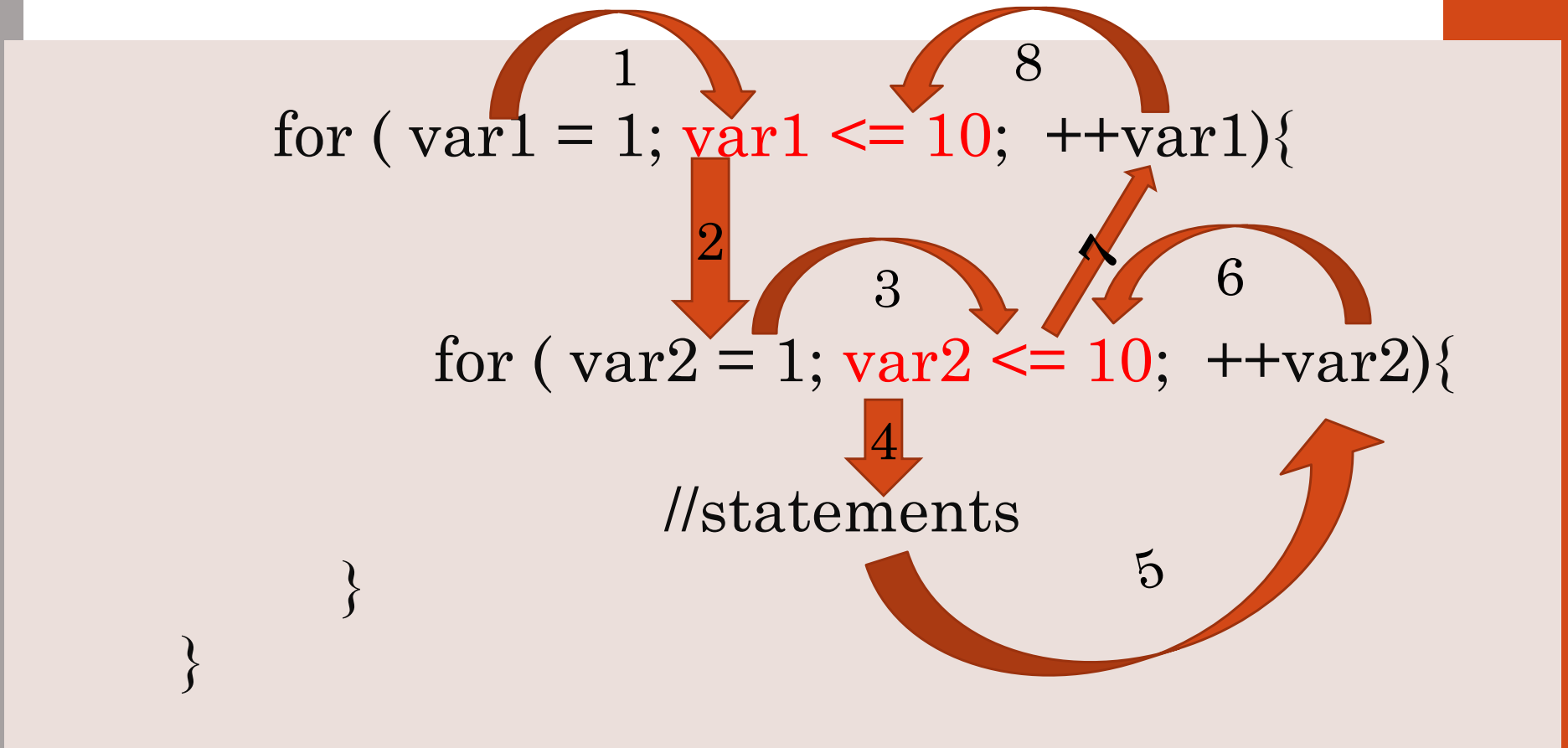
```
for ( init; condition; increment )
{
        for ( init; condition; increment)
        {
           statement(s);
        }

 statement(s);
}
```

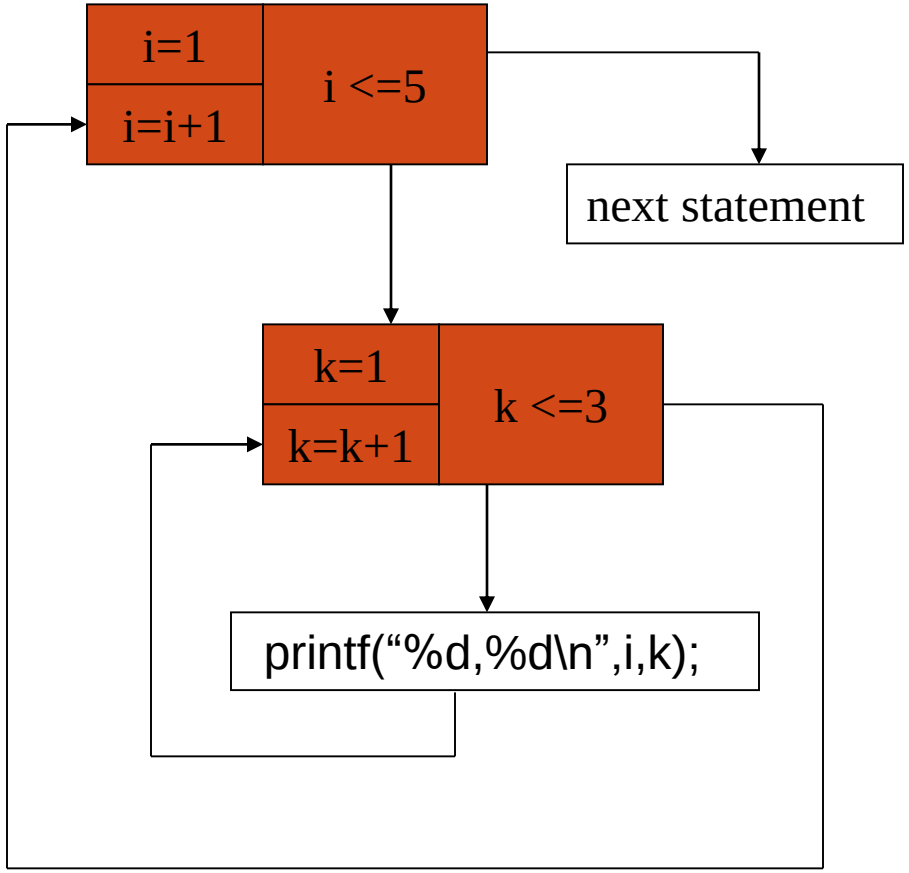
```
while(condition)
{
    while(condition)
    {
        statement(s);
    }
 statement(s);
}
```

Nesting of Loops

- To avoid conflictions it is better to use distinct names for control variables of nested loops.
- When inner loop finish all the iterations, then goes to next iteration of outer loop (4,5,6 iterates, then 7).



```
for (i=1; i<=5; i++){
  for(k=1; k<=3; k++){
    printf("%d,%d\n",i,k);
  }
Next statements
}
```



OUTPUT:

1,1

1,2

1,3

2,1

2,2

2,3

3,1

….

5,2

5,3

Patterns with loops

*****

*****

*****

decides number of Rows

```
for (int i=1; i<=3; i ++)
        {
                for (int j = 1 ; j<=5; j ++)
                {
                        printf(“*”);
                }
                printf("\n“);
        }
```

decides number of Columns

Exercise

Get the following pattern using nested for loops.

```
*
**
***
****
*****
******
```

```
for (int i= 1; i<=6; i++)
{
        for (int j = 1 ; j<=i; j++){
                printf(“*“);
        }
        printf("\n“);
}
```

Determining which Loop to use

- If the loop body may not be executed when the condition is false, use a while/for loop.
- If the body of the loop must be executed at least once any loop may be used but the do-while loop is preferable.

Infinite Loops

- An infinite loop is one in which the condition is initially satisfied, so the loop is entered, but **the condition for exiting the loop is never met**.

- Generally an infinite loop is **caused by failing to modify the control variable** within the loop body.

- To break out of a malfunctioning program press **ctrl+c** on Linux or ctrl+break, on an DOS or Windows machine.

Exercises

1. Write a program that print all the even integers from 0 to 100.

2. Write a C program to print A to Z in English alphabet using a loop.

A B C D E F G .....Z


Any
Question

THANK YOU… !