Dr.SNS RAJALAKSHMI COLLEGE OF ARTS AND SCIENCE

(AUTONOMOUS)
COIMBATORE-641049
Accredited by NAAC(Cycle III) with "A+" Grade
Recognised by UGC, Approved by AICTE, New Delhi and
Affiliated to Bharathiar University, Coimbatore.

DEPARTMENT OF COMPUTER SCIENCE

21UCU401: Programming in C
I YEAR - I SEM

UNIT 2 – The Loop Control Structure &

The Case Control Structure

TOPIC – The Loop Control Structure

Outline

- The *while* Loop
- The *for* Loop
- The Odd Loop
- The *break* Statement
- The *continue* Statement
- The *do-while* Loop.

Outline

- Decisions Using *switch*
- *switch* Versus *if-else* Ladder
- The *goto* Keyword.

The while Loop

- In programming, a loop is used to repeat a block of code until the specified condition is met.
- C programming has three types of loops:
- for loop
- while loop
- do...while loop

The while Loop

while loop

The syntax of the while loop is:

```
while (testExpression)
{
 // the body of the loop
 }
```

The while Loop

How while loop works?

•The while loop evaluates the testExpression inside the parentheses ().

•If testExpression is **true**, statements inside the body of while loop are executed.

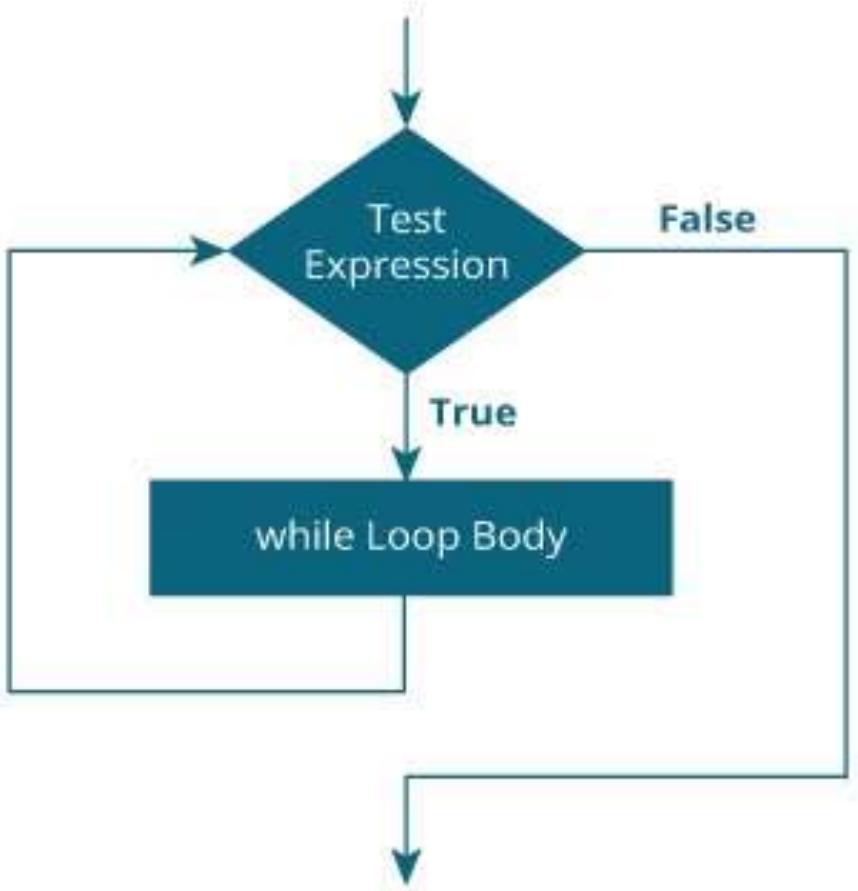
Then, testExpression is evaluated again.

•The process goes on until testExpression is evaluated to **false**.

•If testExpression is **false**, the loop terminates (ends).

The while Loop

Flowchart of while loop



The while Loop

Example 1: while loop

```
// Print numbers from 1 to 5
#include <stdio.h>
 int main()
{
int i = 1;
while (i <= 5)
                {
                printf("%d\n", i);
                ++i;
                }
 return 0;
 }
```

The while Loop

Example 1: while loop

Output

1 2 3 4 5

The while Loop

Here, we have initialized i to 1.

1.When i = 1, the test expression i <= 5 is **true**. Hence, the body of the while loop is executed.

This prints 1 on the screen and the value of i is increased to 2.

2.Now, i = 2, the test expression i <= 5 is again **true**. The body of the while loop is executed again.

This prints 2 on the screen and the value of i is increased to 3.

3.This process goes on until i becomes 6. Then, the test expression i <= 5 will be **false** and the loop terminates.

For Loop

for Loop

The syntax of the for loop is:

```
for (initializationStatement; testExpression; updateStatement)
{
 // statements inside the body of loop
 }
```

For Loop

How for loop works?

•The initialization statement is executed only once.

•Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.

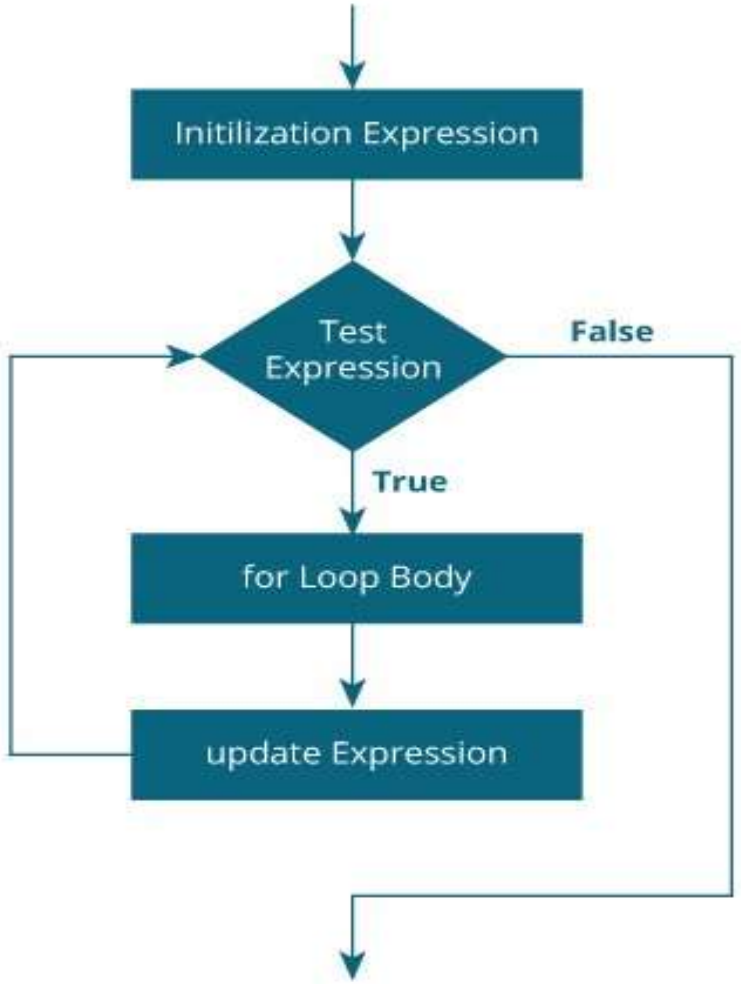
•However, if the test expression is evaluated to true, statements inside the body of the for loop are executed, and the update expression is updated.

•Again the test expression is evaluated.

This process goes on until the test expression is false. When the test expression is false, the loop terminates.

For Loop

For loop Flowchart



For Loop

Example 1: for loop

```
// Print numbers from 1 to 10
#include <stdio.h>
 int main()
{
int i;
 for (i = 1; i < 11; ++i)
 {
printf("%d ", i);
 }
return 0;
 }
```

For Loop

Example 1: for loop

Output

1 2 3 4 5 6 7 8 9 10

For Loop

1.i is initialized to 1.

2.The test expression i < 11 is evaluated. Since 1 less than 11 is true, the body of for loop is executed. This will print the **1** (value of i) on the screen.

3.The update statement ++i is executed. Now, the value of i will be 2. Again, the test expression is evaluated to true, and the body of for loop is executed. This will print **2** (value of i) on the screen.

4.Again, the update statement ++i is executed and the test expression i < 11 is evaluated. This process goes on until i becomes 11.

5.When i becomes 11, i < 11 will be false, and the for loop terminates.


THANK YOU

The Odd Loop

Odd loops

Sometimes a user may not know about how many times a loop is to be executed. If we want to execute a loop for unknown number of times, then the concept of odd loops should be implemented. This can be done using for-loop, while-loop or do-while-loops.

The Odd Loop

Example C Program-The Odd Loop Demo

```
#include<stdio.h>
#include<conio.h>
void main()
{
char another='y';
int num,sq;
while(another=='y')
{
 printf("Enter a number:");
 scanf("%d",&num);
 sq=num*num;
 printf("Square of %d is %d",num,sq);
 printf("\nWant to enter another number y/n:");
scanf("%c",&another);
 another=getchar();
 }
getch();
```

The Odd Loop

Output:

Enter a number: 5

Square of 5 is 25

Want to enter another number y/n:y

Enter a number: 17

Square of 17 is 289

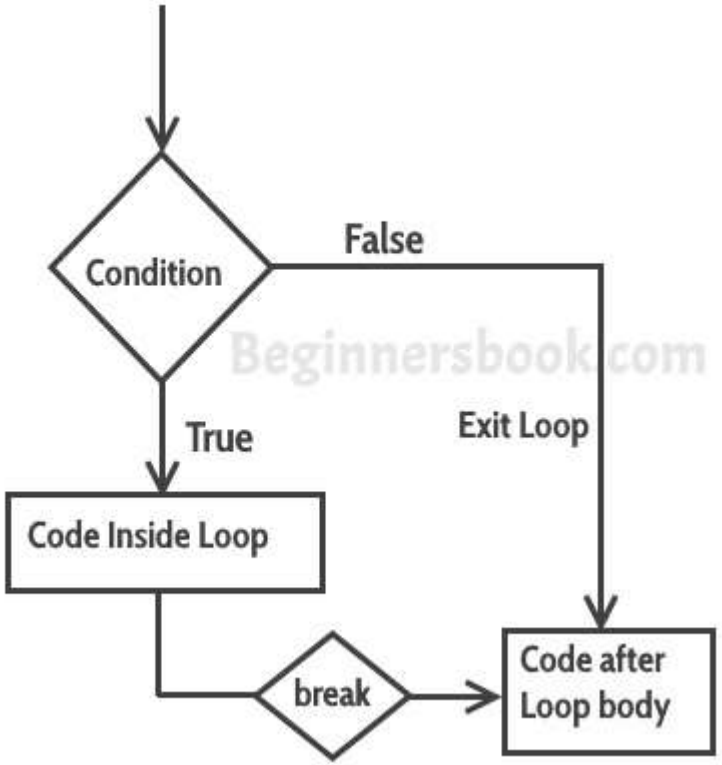
Want to enter another number y/n:n

The break Statement

- We often come across situations where we want to jump out of a loop instantly, without waiting to get back to the conditional test.
- The keyword break allows us to do this.
- When break is encountered inside any loop, control automatically passes to the first statement after the loop. A break is usually associated with an if.

The break Statement

Flow diagram of break statement



The break Statement

Syntax:

break;

Example C Program - A number is prime or not.

```
#include <stdio.h>
void main( )
{
int num, i ;
printf ( "Enter a number " ) ;
scanf ( "%d", &num ) ;
i = 2 ;
```

The break Statement

```
while ( i <= num - 1 )
{
if ( num % i == 0 )
{
printf ( "Not a prime number" ) ;
break ;
}
i++ ;
}
if ( i == num )
printf ( "Prime number" ) ;
}
```

The break Statement

Output:

Enter a number 19

Prime number

Enter a number 15

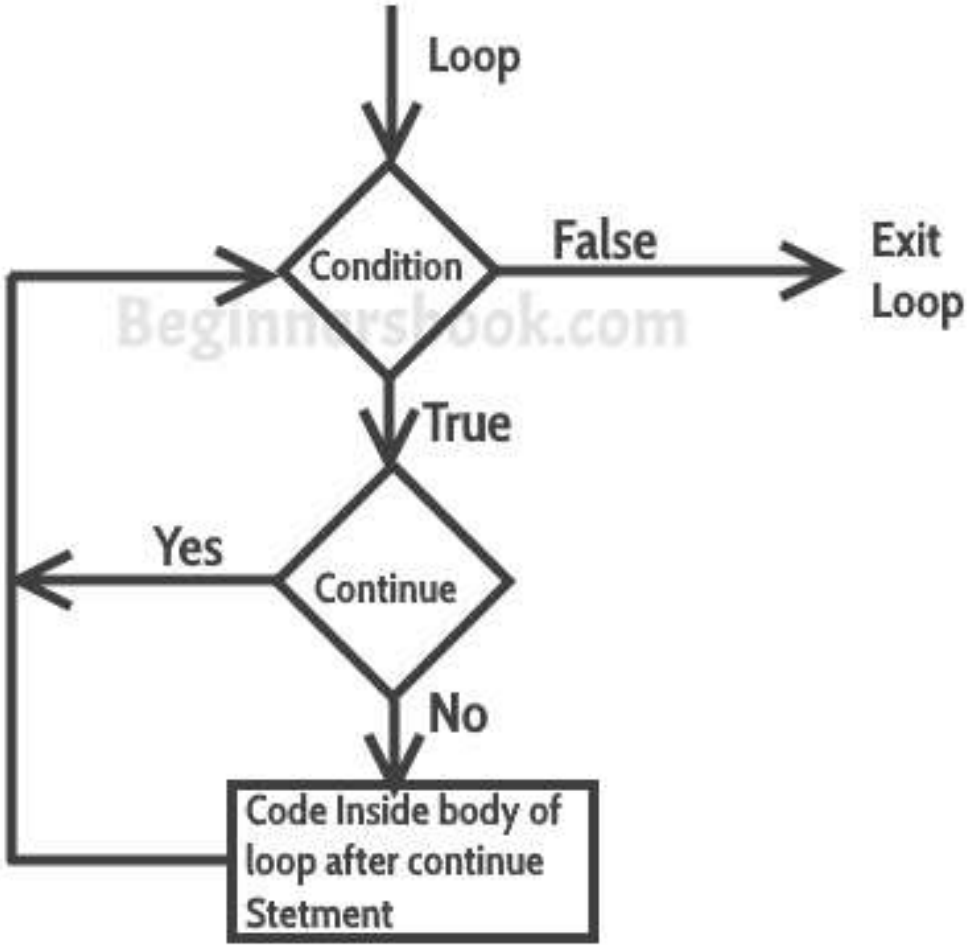
Not a prime number


THANK YOU

The continue Statement

- In some programming situations we want to take the control to the beginning of the loop, bypassing the statements inside the loop, which have not yet been executed.
- The keyword continue allows us to do this.
- When continue is encountered inside any loop, control automatically passes to the beginning of the loop.

The continue Statement



The break Statement

Syntax:

continue;

Example C Program - The print statement is skipped when the counter value was 7.

```
#include <stdio.h>
int main ()
{
   int counter = 10;
   while (counter >= 0)
   {
     if (counter == 7)
  {
     counter--;
     continue;
  }
```

The break Statement

Syntax:

continue;

Example C Program - The print statement is skipped when the counter value was 7.

```
printf ("%d ", counter);
      counter--;
    }
    return 0;
}
```

The break Statement

Output:

10 9 8 6 5 4 3 2 1 0

if we do not place the counter– statement in the body of “if” then the value of counter would remain 7 indefinitely. The print statement is skipped when the counter value was 7.

Note: When the continue statement is executed within the loop body then control will pass back to the condition without executing the remaining statement.

Difference Between the break and continue statement

Break Statement

- ✓ The Break statement is used to exit from the loop constructs.
- ✓ The break statement is usually used with the switch statement, and it can also use it within the while loop, do-while loop, or the for-loop.
- ✓ When a break statement is encountered then the control is exited from the loop construct immediately.
- ✓ **Syntax:**
  break;

Continue Statement

- ✓ The continue statement is not used to exit from the loop constructs.
- ✓ The continue statement is not used with the switch statement, but it can be used within the while loop, do-while loop, or for-loop.
- ✓ When the continue statement is encountered then the control automatically passed from the beginning of the loop statement.
- ✓ **Syntax:**
  continue;

The do while Loop

- Unlike **for** and **while** loops, which test the loop condition at the top of the loop, the **do...while** loop in C programming checks its condition at the bottom of the loop.

- A **do...while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Syntax

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

The do while Loop

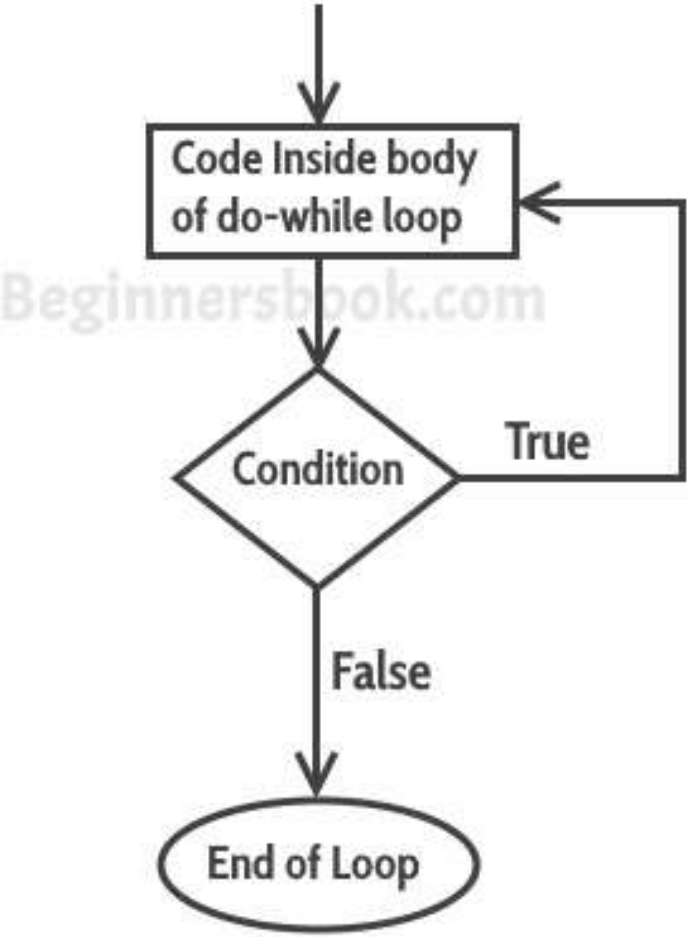
Syntax

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

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

The do while Loop



The do while Loop

```
#include <stdio.h>
 int main ()
 { /* local variable definition */
 int a = 10;
 /* do loop execution */
 do
 {
 printf("value of a: %d\n", a);
a = a + 1;
}while( a < 20 );
 return 0;
 }
```

The do while Loop

Output:

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

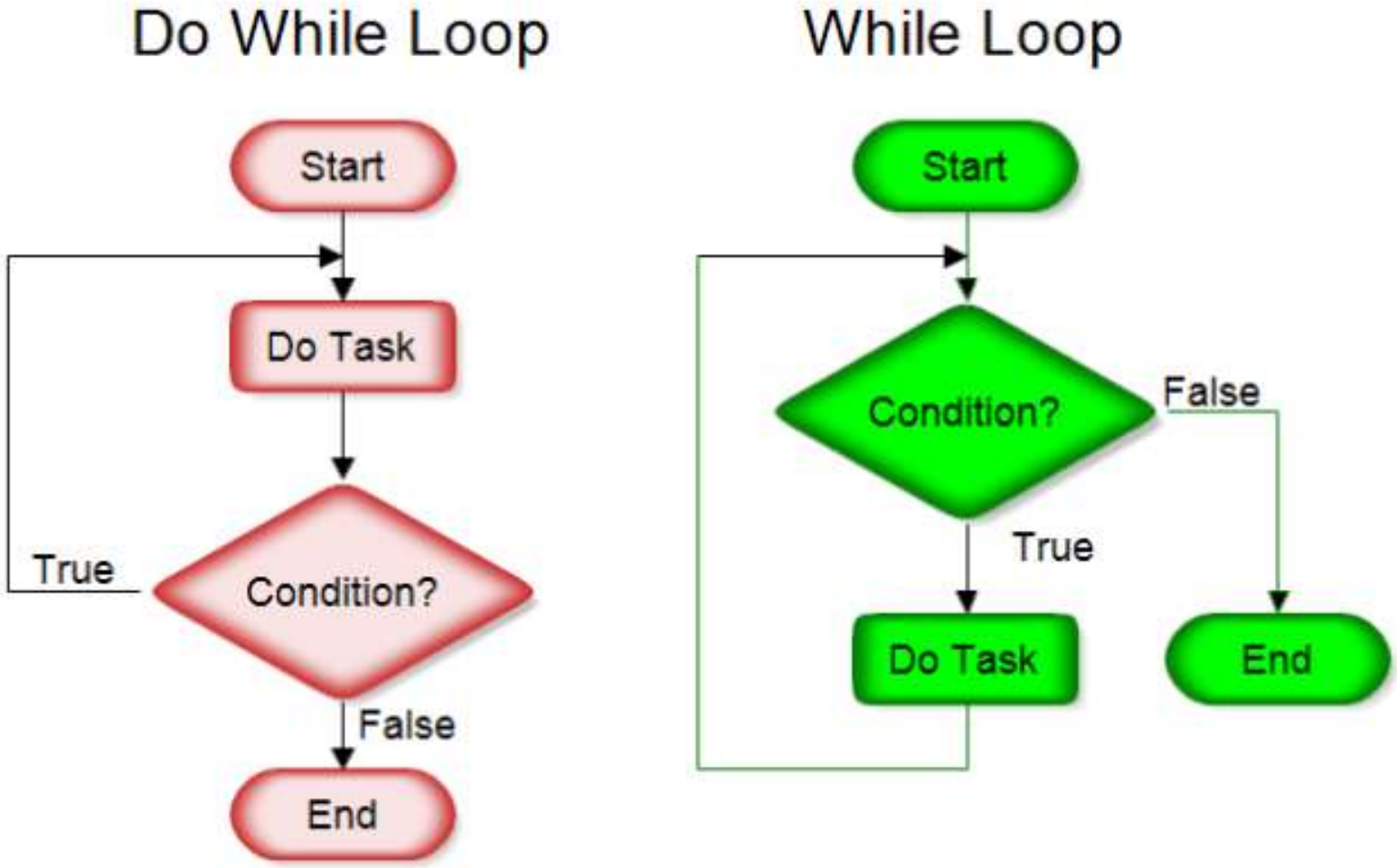
value of a: 16

value of a: 17

value of a: 18

value of a: 19

Difference between Do While Loop and While Loop



Difference between Do While Loop and While Loop

while	do-while
✓ Condition is checked first then statement(s) is executed.	✓ Statement(s) is executed atleast once, thereafter condition is checked.
✓ It might occur statement(s) is executed zero times, If condition is false.	✓ At least once the statement(s) is executed.
✓ No semicolon at the end of while. while(condition)	✓ Semicolon at the end of while. while(condition);
✓ If there is a single statement, brackets are not required.	✓ Brackets are always required.
✓ Variable in condition is initialized before the execution of loop.	✓ variable may be initialized before or within the loop.
✓ while loop is entry controlled loop.	✓ do-while loop is exit controlled loop.
✓ while(condition) { statement(s); }	✓ do { statement(s); } while(condition);


THANK YOU