

MODULE 2

BRANCHING AND LOOPING

1 Two-Way Selection

A two way selection statement checks for a condition and selects from the two statement blocks which one to execute depending upon whether the condition is true or false.

Control statements are used to control the order of execution of the statements. Decision statements controls the execution/not execution of certain statements based on the condition. The main decision statement in C is if statement. There are four variants:

- if
- if-else
- nested if-else
- cascaded if-else or else-if ladder

1.1 If Statement

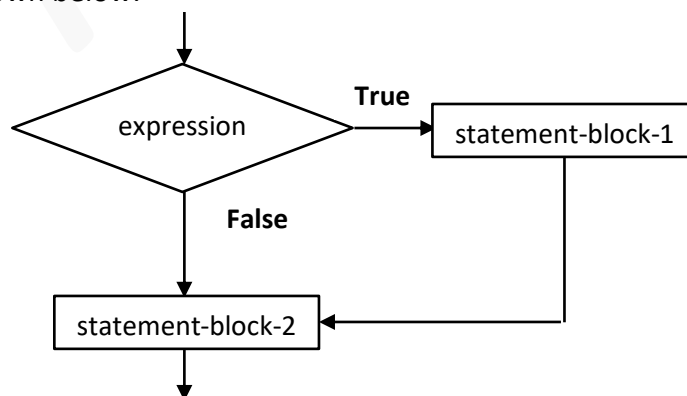
➤ It is one-way selection statement. It is used only when there is one alternative.

➤ The syntax of if statement is:

```
if ( expression )
{
    statement-block-1;
}
statement-block-2;
```

➤ The expression is evaluated to true or false.

- If the expression is evaluated to true (Note: Any non-zero value is evaluated to true), then statement-block-1 is executed and the control comes outside of if statement and the execution of further statements (statement-block-2) continues if any.
- If the expression is evaluated to be false, then statement-block-1 is skipped. The flowchart is shown below:



- **Example:** Program to check whether a given number is positive.

```
#include<stdio.h>

int main()
{
    int n;

    /*Input the number.*/
    printf ( "\nEnter the number: " );
    scanf ( "%d", &n );

    /*Check if the number is greater than 0.*/
    if ( n > 0 )
    {
        printf ( "\nThe number is positive" );
    }

    printf ( "\nBye!!!" );

    return ( 0 );
}
```

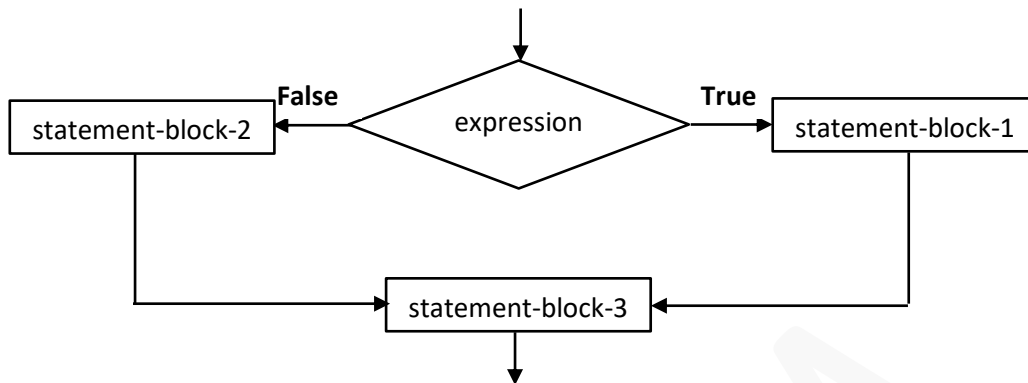
Output:

1. Enter the number: 10
The number is positive
Bye!!!
2. Enter the number: -10
Bye!!!

1.2 If-else Statement

- It is two-way selection statement. It is used when we have to choose between two alternatives.
- The syntax of if-else statement is:
- ```
if (expression)
{
 statement-block-1;
}
else
{
 statement-block-2;
}
statement-block-3;
```
- The expression is evaluated to true or false.
- If the expression is evaluated to true, then statement-block-1 is executed and the control comes outside of if-else and statement-block-3 is executed.

- If the expression is evaluated to false, then statement-block-2 is executed and the control comes outside of if-else and statement-block-3 is executed. The flowchart is shown below:



- **Example:** Program to check whether a given number is positive or negative.

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

```
int main()
{
 int n;

 /*Input the number.*/
 printf ("\nEnter the number: ");
 scanf ("%d", &n);

 /*Check if the number is greater than 0.*/
 if (n > 0)
 {
 printf ("\nThe number is positive");
 }
 else
 {
 printf ("\nThe number is negative");
 }

 printf ("\nBye!!!");

 return (0);
}
```

Output:

1. Enter the number: 10  
The number is positive  
Bye!!!
2. Enter the number: -10  
The number is negative  
Bye!!!

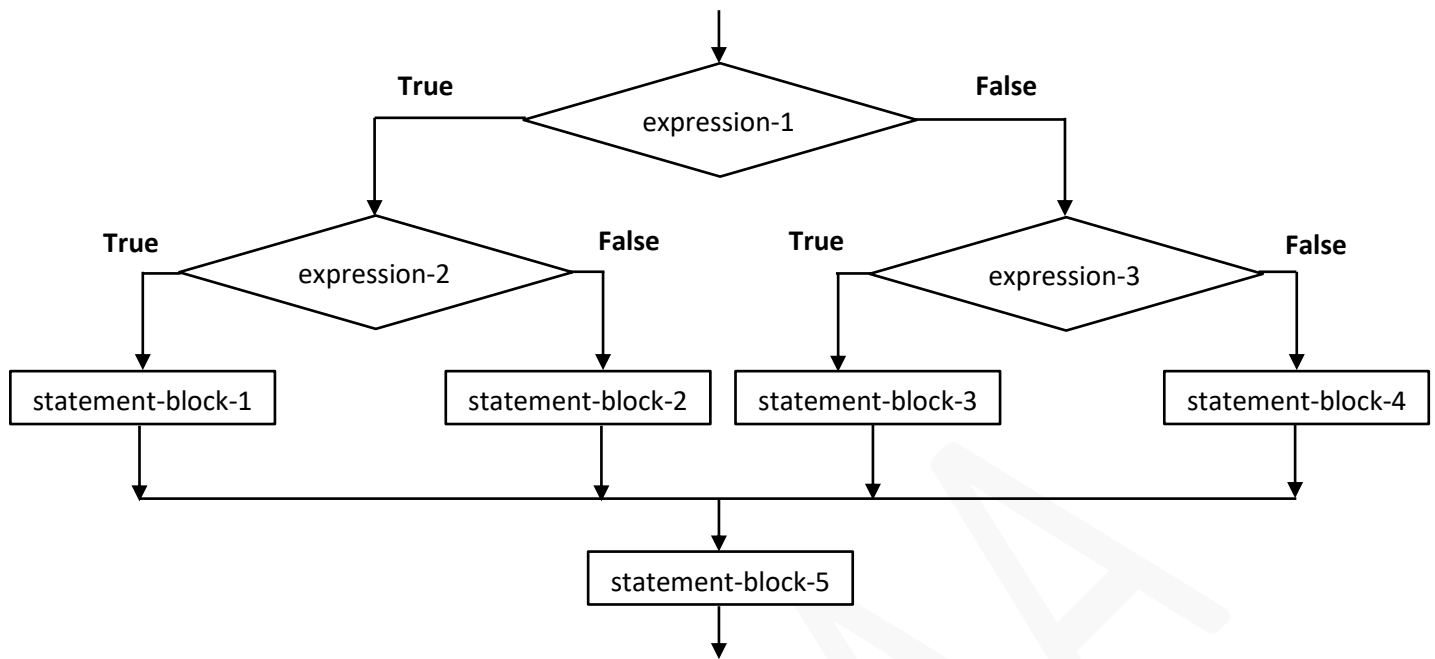
### 1.3 Nested if-else Statement

- It is multi-way selection statement. It is used when an action has to be performed based on many decisions.
- An if-else statement within another if-else statement is called nested if-else statement.
- The syntax of nested if-else statement is:

```
if (expression-1)
{
 if (expression-2)
 statement-block-1;
 else
 statement-block-2;
}
else
{
 if (expression-3)
 statement-block-3;
 else
 statement-block-4;
}
statement-block-5;
```

- The expression-1 is evaluated to true or false.
  - If expression-1 is evaluated to true, then expression-2 is evaluated to true or false. If expression-2 is evaluated to true, then statement-block-1 is executed. If expression-2 is evaluated to false, then statement-block-2 is executed.
  - If expression-1 is evaluated to false, then expression-3 is evaluated to true or false. If expression-3 is evaluated to true, then statement-block-3 is executed. If expression-3 is evaluated to false, then statement-block-4 is executed.

In either of the cases, after execution of a particular statement-block the control comes outside the nested if-else statement and continues execution from statement-block-5. The flowchart is shown below:



➤ **Example:** Program to find the largest of three numbers.

```

#include<stdio.h>

int main()
{
 int a, b, c;

 printf ("\nEnter three numbers: ");
 scanf ("%d%d%d", &a, &b, &c);

 if (a > b)
 {
 if (a > c)
 printf ("\n%d is largest", a);
 else
 printf ("\n%d is largest", c);
 }
 else
 {
 if (b > c)
 printf ("\n%d is largest", b);
 else
 printf ("\n%d is largest", c);
 }

 printf ("\nBye!!!");

 return (0);
}

```

```
}
```

Output:

1. Enter three numbers: 10 5 0  
10 is largest  
Bye!!!
2. Enter three numbers: 23 -15 46  
46 is largest  
Bye!!!

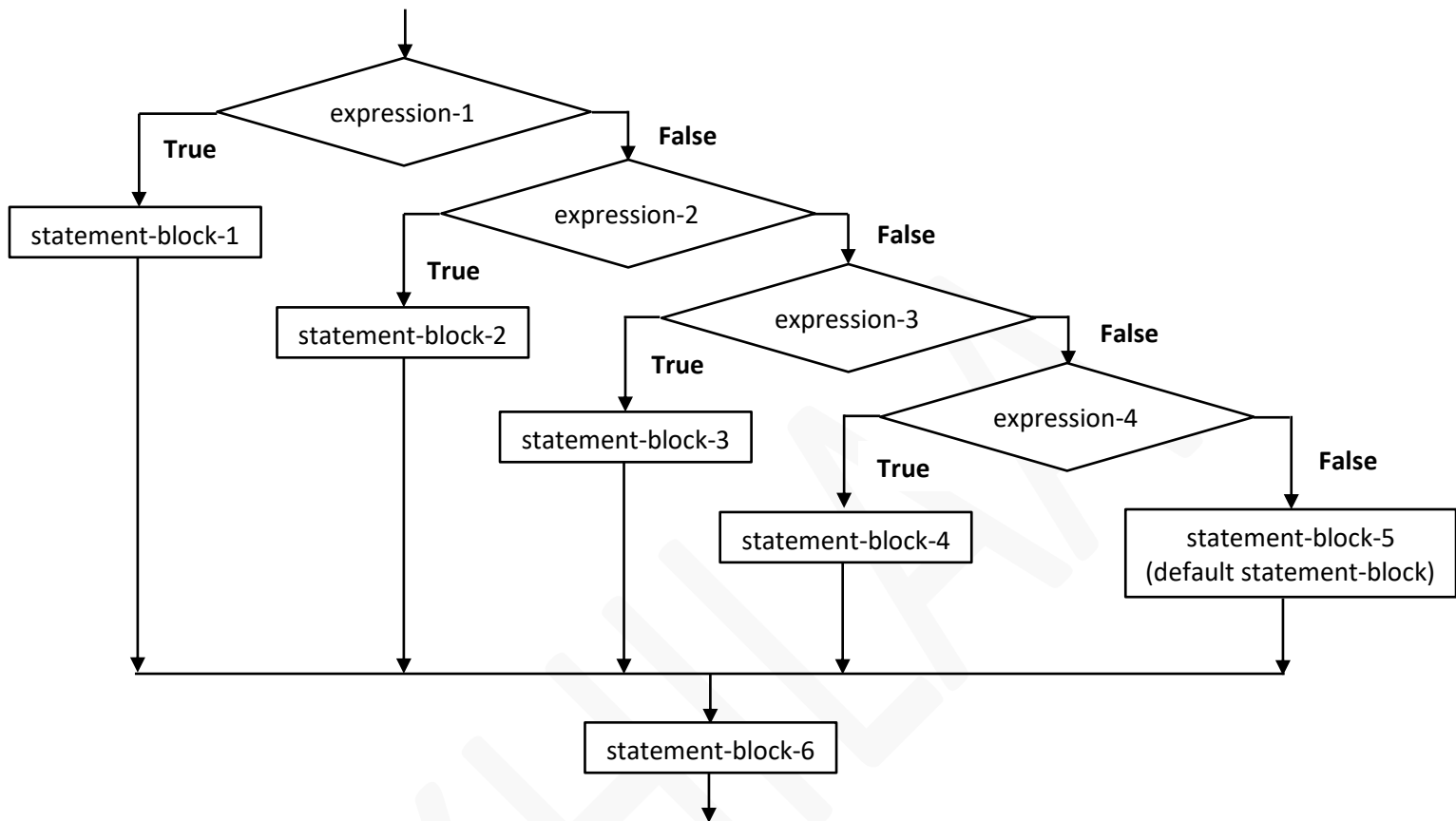
#### 1.4 Cascaded if-else or else if ladder Statement

- It is multi-way selection statement. It is used when we must choose among many alternatives.
- The syntax of cascade if-else or else if ladder statement is:

```
if (expression-1)
{
 statement-block-1;
}
else if (expression-2)
{
 statement-block-2;
}
else if (expression-3)
{
 statement-block-3;
}
else if (expression-4)
{
 statement-block-4;
}
else
{
 statement-block-5;
}
statement-block-6;
```

- The expression is evaluated in top to bottom order. If an expression is evaluated to true, then the statement-block associated with that expression is executed and the control comes out of the entire else if ladder and continues execution from statement-block-6 if any.
- For example:
  - If expression-1 is evaluated to true, then statement-block-1 is executed and the control comes out of the entire else if ladder and continues execution from statement-block-6 if any.

- If all the expressions are evaluated to false, then the last statement-block-5 (default) is executed and the control comes out of the entire else if ladder and continues execution from statement-block-6 if any. The flowchart is shown below:



➤ **Example:** Program to check whether a given number is zero, positive or negative.

```

#include<stdio.h>

int main()
{
 int n;

 /*Input the number.*/
 printf ("\nEnter the number: ");
 scanf ("%d", &n);

 /*Check if the number is greater than 0.*/
 if (n > 0)
 {
 printf ("\nThe number is positive");
 }
 /*Check if the number is lesser than 0.*/
 else if (n < 0)
 {
 printf ("\nThe number is negative");
 }
}

```

```

 }
 /*The number is zero.*/
 else
 {
 printf ("\nThe number is zero");
 }

 printf ("\nBye!!!");

 return (0);
}

```

Output:

1. Enter the number: 10  
The number is positive  
Bye!!!
2. Enter the number: 0  
The number is zero  
Bye!!!
3. Enter the number: -10  
The number is negative  
Bye!!!

## 2 Switch Statement

- It is multi-way selection statement. It is used when we must choose among many alternatives.
- The switch statement tests the value of a given expression against a list of case values and when a match is found, a block of statement associated with that case is executed.
- The syntax of switch statement is:

```

switch (expression)
{
 case value-1: statement-block-1
 break;
 case value-2: statement-block-2
 break;

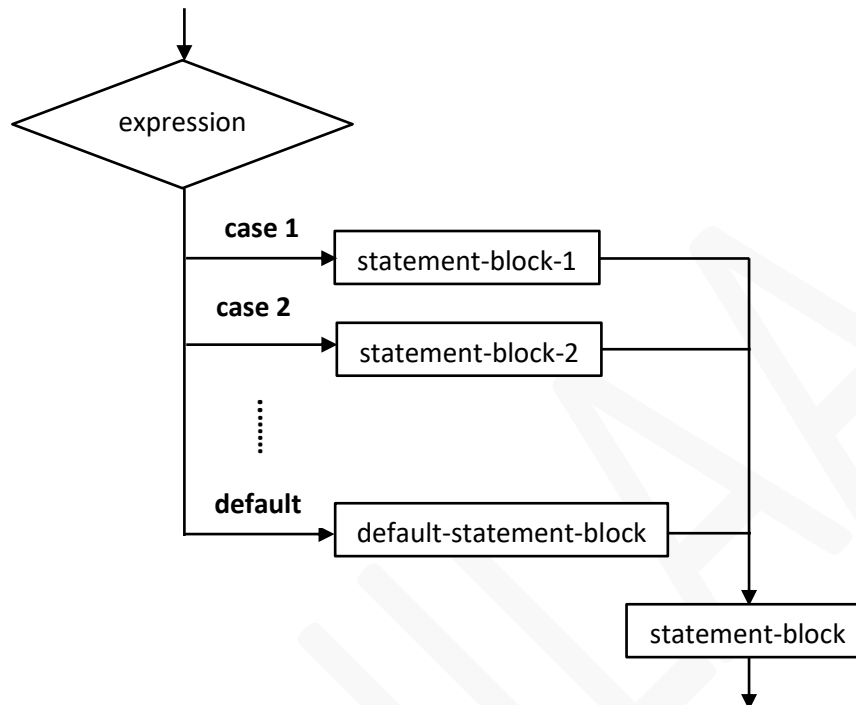
 default : default-statement-block
}
statement-block

```

- Here, expression is an integer expression i.e., it should evaluate to an integer. It can also contain any character.
- value-1, value-2,... are constants. Each of which must be unique.
- Based on the value of the expression (integer or character), the control is transferred to a matching case-value where corresponding statement-block is executed and the control comes out of the



switch block using break statement. If the value of the expression doesn't match with any of the case-values (i.e. value-1, value-2,...), then the control goes to default-case (if it exists, since default is optional). The flowchart is shown below:



- **Example:** Program to illustrate the use of switch statement.

```

#include<stdio.h>

int main()
{
 char grade;

 printf ("\nEnter the grade (A-E): ");
 scanf ("%c", &grade);

 switch (grade)
 {
 case 'A': printf ("\nOutstanding!!!");
 break;
 case 'B': printf ("\nGood!!!");
 break;
 case 'C': printf ("\nAverage!!!");
 break;
 case 'D': printf ("\nPoor!!!");
 break;
 case 'E': printf ("\nFail!!!");
 }
}

```

```

 break;
 default: printf ("\nInvalid Grade!!!");
}

printf ("\nYour grade is: %c", grade);

return (0);
}

```

Output:

1. Enter the grade (A-E): A  
Outstanding!!!  
Your grade is: A
2. Enter the grade (A-E): F  
Invalid Grade!!!  
Your grade is: F
3. Enter the grade (A-E): B  
Good!!!  
Your grade is: B
4. Enter the grade (A-E): E  
Fail!!!  
Your grade is: A

### 3 Ternary Operator (?:)

- It is useful for making two-way selections.
- The syntax of ternary operator is:  
expression-1 ? expression-2 : expression-3
- The expression-1 is evaluated to true or false.
  - If the expression-1 is evaluated to true, then expression-2 is executed.
  - If the expression-1 is evaluated to false, then expression-3 is executed.
- **Example-2:** Program to check whether a given number is positive or negative.

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

```
int main()
{
```

```
 int n;
```

```
 printf("\nEnter a number: ");
 scanf("%d", &n);
```

```
 (n > 0) ? printf("POSITIVE!!!\n") : printf("NEGATIVE!!!\n");
```

```
 return 0;
```

}

Output:

1. Enter a number: 10  
POSITIVE!!!
2. Enter a number: -57  
NEGATIVE!!!

➤ **Example-1:** Program to check whether a number is even or odd.

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

```
int main()
{
 int n, res;

 printf ("\nEnter the number: ");
 scanf ("%d", &n);

 res = (n % 2 == 0) ? 0 : 1;

 if (res == 0)
 printf ("\nThe number is Even!!!");
 else
 printf ("\nThe number is Odd!!!");

 return (0);
}
```

**OR**

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

```
int main()
{
 int n;

 printf ("\nEnter the number: ");
 scanf ("%d", &n);

 (n % 2 == 0) ? printf("\nThe number is Even!!!") : printf("\nThe number is Odd!!!");

 return (0);
}
```

Output:

1. Enter the number: 10  
The number is Even!!!
2. Enter the number: 5  
The number is Odd!!!

- **Example-3:** Program to find the largest of three numbers.

```
#include <stdio.h>

int main ()
{
 int a, b, c, large;

 printf ("\nEnter the three numbers: ");
 scanf ("%d%d%d", &a, &b, &c);

 large = (a > b) ? ((a > c) ? a : c) : ((b > c) ? b : c);

 printf ("\nThe largest is: %d", large);

 return (0);
}
```

Output:

1. Enter the three numbers: 10 5 0  
The largest is 10
2. Enter three numbers: 23 -15 46  
The largest is 46

#### 4 **Goto and Labels**

- The goto statement is used to alter the normal sequence of program execution by transferring the control to some other part of the program unconditionally.
- The syntax of goto statement is:

|                                                                |    |                                                                |
|----------------------------------------------------------------|----|----------------------------------------------------------------|
| <pre>goto label; ... ... ... label:     statement-block;</pre> | OR | <pre>label:     statement-block; ... ... ... goto label;</pre> |
|----------------------------------------------------------------|----|----------------------------------------------------------------|

- During execution when a goto statement is encountered, the control will jump to the statement immediately following the label. The label can be anywhere in the program either before or after goto.
- **Example:** Program to illustrate the use of goto statement.

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

```

int main()
{
 int a = 1;

 LOOP:
 if (a != 6)
 {
 printf("\nThe value of a is: %d", a);
 a++;
 goto LOOP;
 }

 printf ("\nBye!!!");

 return (0);
}

```

Output:

```

The value of a is: 1
The value of a is: 2
The value of a is: 3
The value of a is: 4
The value of a is: 5
Bye!!!

```

## 5 Loops

A set of statements that are executed repeatedly until a condition is satisfied is called a loop. When a condition is not satisfied then the loop is terminated. It is useful in performing repetitive tasks. There are 3 types of loops:

- while
- do-while
- for

### 5.1 While Loop

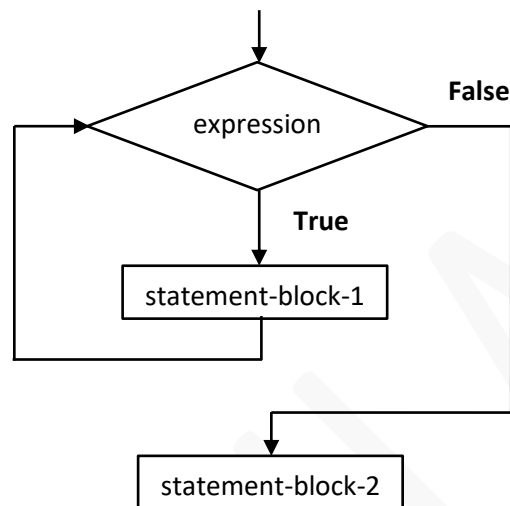
- A while loop is used to execute a set of statements repeatedly as long as the given condition is satisfied.
- The syntax of while loop is:
 

```

while (expression)
{
 statement-block-1;
}
statement-block-2;

```
- The expression is evaluated to true or false.

- If the expression is evaluated to true (1) (Note: Any non zero value is evaluated to true), then the body of the loop (statement-block-1) is executed. After executing the body of the loop, the control goes back to the beginning of the while statement and the expression is evaluated to true or false. This cycle is continued until the expression becomes false.
- If the expression is evaluated to false (0), then the control comes out of the loop, without executing the body of the loop and starts executing from statement-block-2. The flowchart is shown below:



➤ **Example:** Program to print numbers from 1-5.

```

#include<stdio.h>

int main()
{
 int i;

 /*Initialization.*/
 i = 1;

 /*Beginning of Loop.*/
 while (i <= 5)
 {
 printf ("%d\n", i);
 i = i + 1; /*Increment.*/
 }
 /*End of Loop.*/

 printf ("\nFinish!!!");

 return (0);
}

```

Output:

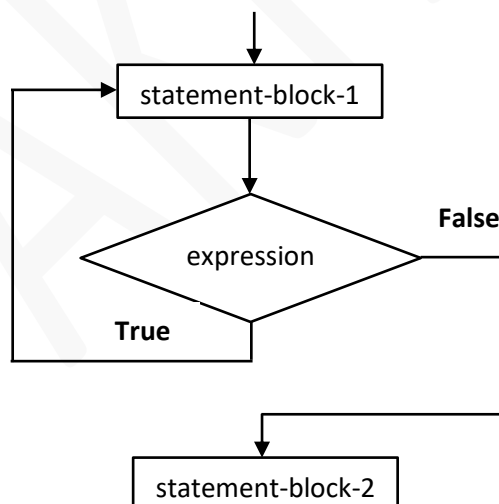
```
1
2
3
4
5
Finish!!!
```

## 5.2 Do-while Loop

- A do-while loop works similar to a while loop, except that it is guaranteed to execute the body of the loop at least one time.
- The syntax of do-while loop is:

```
do
{
 Statement-block-1;
}
while (expression);
statement-block-2;
```

- The body of the loop (statement-block-1) is executed at least once. Then the expression is evaluated to true or false.
  - If the expression is evaluated to true, then the body of the loop (statement-block-1) is executed. After executing the body of the loop the expression is evaluated again to true or false. This cycle continues until the expression becomes false.
  - If the expression is evaluated to false, then the control comes out of the loop and starts executing from statement-block-2. The flowchart is shown below:



- **Example:** Program to print numbers from 1-5.

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

```

int main()
{
 int i;

 /*Initialization.*/
 i = 1;

 /*Beginning of Loop.*/
 do
 {
 printf ("%d\n", i);
 i = i + 1; /*Increment.*/
 } while (i <= 5);
 /*End of Loop.*/

 printf ("\nFinish!!!");

 return (0);
}

```

Output:

```

1
2
3
4
5
Finish!!!

```

### 5.3 For Loop

- A for loop is used to execute a set of statements repeatedly as long as the given condition is true.
- The syntax of while loop is:
 

```

for (expression-1; expression-2; expression-3)
{
 statement-block-1;
}
statement-block-2;

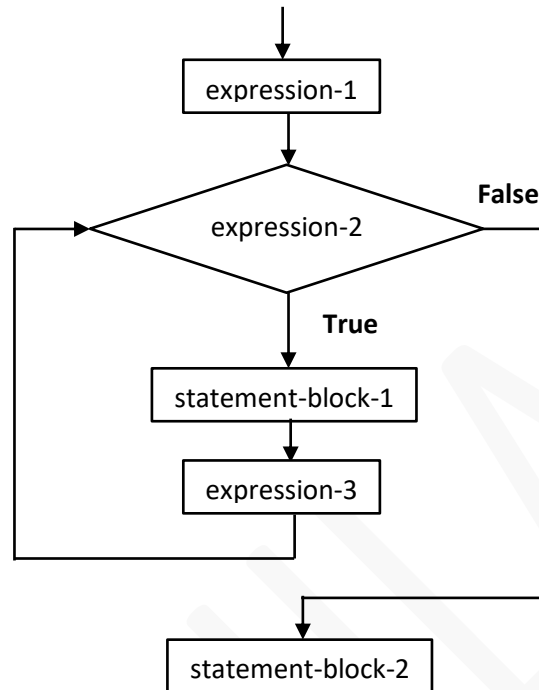
```

  - Here, expression-1 contains initialization statement.
  - expression-2 contains limit test expression.
  - expression-1 contains updating statement.
- Expression-1 is evaluated first. It is executed only once i.e. when for loop is entered for the first time.
- Then expression-2 is evaluated to true or false.
  - If the expression-2 is evaluated to true, then the body of the loop (statement-block-1) is executed once. After executing the body of the loop, expression-3 is evaluated. Then the



expression-2 is evaluated to true or false. If true the cycle continues until expression-2 becomes false.

- If the expression-2 is evaluated to false, then the control comes out of the loop, without executing the body of the loop and the starts executing from statement-block-2. The flowchart is shown below:



- **Example:** Program to print numbers from 1-5.

```

#include<stdio.h>

int main()
{
 int i;

 /*Beginning of Loop.*/
 for (i = 1 ; i <= 5; i++)
 {
 printf ("%d\n", i);
 }
 /*End of Loop.*/

 printf ("\nFinish!!!");

 return (0);
}

```

Output:

1

2  
3  
4  
5  
Finish!!!

## 6 Break and Continue

### 6.1 Break Statement

- The break statement in C has two uses:
  - When a break statement is encountered inside a loop, the loop is terminated immediately and the control is transferred to the next statement following the loop.
  - It can be used to terminate a case in the switch statement.
- The syntax of break statement is:  
**break;**
- **Example:** Program to illustrate the use of break statement.

```
#include<stdio.h>

int main()
{
 int a;

 for(a = 1 ; a <= 10 ; a++)
 {
 if(a == 5)
 {
 break;
 }

 printf("The value of a is %d\n", a);
 }

 return 0;
}
```

Output:

The value of a is 1  
The value of a is 2  
The value of a is 3  
The value of a is 4

### 6.2 Continue Statement

- The continue statement forces the next iteration of the loop to take place, skipping any code in between.

- The syntax of continue statement is:

**continue;**

- **Example:** Program to illustrate the use of continue statement.

```
#include<stdio.h>

int main()
{
 int a;

 for(a = 1 ; a <= 10 ; a++)
 {
 if(a == 5)
 {
 continue;
 }

 printf("The value of a is %d\n", a);
 }

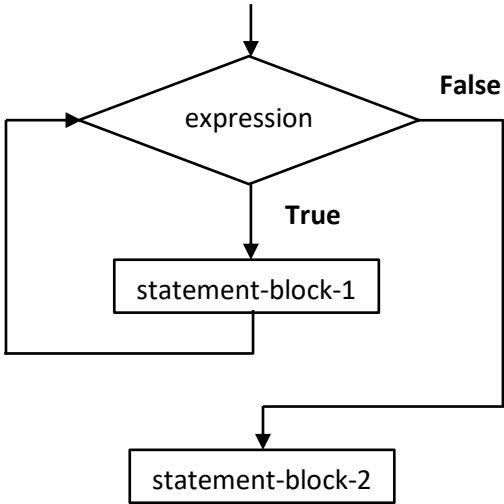
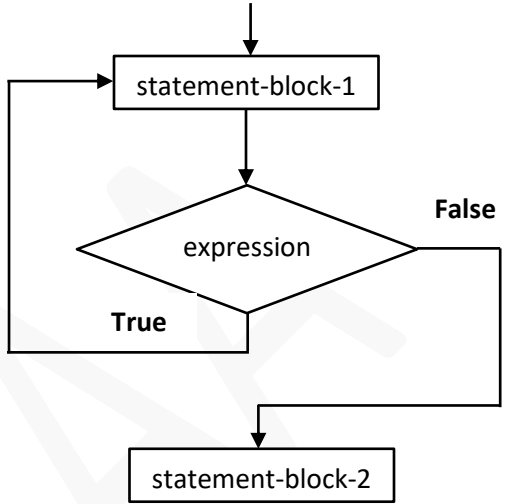
 return 0;
}
```

Output:

```
The value of a is 1
The value of a is 2
The value of a is 3
The value of a is 4
The value of a is 6
The value of a is 7
The value of a is 8
The value of a is 9
The value of a is 10
```

## 7 Difference between while and do-while statement

| while Statement                                                        | do-while Statement                                                                         |
|------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| Entry-controlled loop                                                  | Exit-Controlled loop                                                                       |
| The body of the loop will be executed only when the condition is true. | The body of the loop will be executed atleast once whether the condition is true or false. |
| The syntax of while loop is:<br><b>while</b> ( expression )<br>{       | The syntax of while loop is:<br><b>do</b><br>{                                             |

|                                                                                                                                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> statement-block-1; } statement-block-2; </pre>                                                                                                                                                                                                                                                                                                                          | <pre> Statement-block-1; } while ( expression ); statement-block-2; </pre>                                                                                                                                                                                                                                                                                                        |
| <p>The flowchart of while statement is:</p>  <pre> graph TD     Entry(( )) --&gt; Expression{expression}     Expression -- True --&gt; SB1[statement-block-1]     SB1 --&gt; Entry     Expression -- False --&gt; SB2[statement-block-2] </pre>                                              | <p>The flowchart of do-while statement is:</p>  <pre> graph TD     Entry(( )) --&gt; SB1[statement-block-1]     SB1 --&gt; Expression{expression}     Expression -- True --&gt; SB1     Expression -- False --&gt; SB2[statement-block-2] </pre>                                                |
| <p>Example: Program to print numbers from 1-5.</p> <pre> #include&lt;stdio.h&gt;  int main() {     int i;      /*Initialization.*/     i = 1;      /*Beginning of Loop.*/     while ( i &lt;= 5 )     {         printf ( "%d\n", i );         i = i + 1;    /*Increment.*/     }     /*End of Loop.*/      printf ( "\nFinish!!!" );      return ( 0 ); }  Output: 1 2 </pre> | <p>Example: Program to print numbers from 1-5.</p> <pre> #include&lt;stdio.h&gt;  int main() {     int i;      /*Initialization.*/     i = 1;      /*Beginning of Loop.*/     do     {         printf ( "%d\n", i );         i = i + 1;    /*Increment.*/     } while ( i &lt;= 5 );     /*End of Loop.*/      printf ( "\nFinish!!!" );      return ( 0 ); }  Output: 1 2 </pre> |

|           |           |
|-----------|-----------|
| 3         | 3         |
| 4         | 4         |
| 5         | 5         |
| Finish!!! | Finish!!! |

## 8 Exercises

1. Rewrite your pay computation to give the employee 1.5 times the hourly rate for hours worked above 40 hours.

Enter Hours: 45

Enter Rate: 10

Pay: 475.0

2. Rewrite your pay program using try and except so that your program handles non-numeric input gracefully by printing a message and exiting the program. The following shows two executions of the program:

Enter Hours: 20

Enter Rate: nine

Error, please enter numeric input

Enter Hours: forty

Error, please enter numeric input

3. Write a program to prompt for a score between 0.0 and 1.0. If the score is out of range, print an error message. If the score is between 0.0 and 1.0, print a grade using the following table:

Score Grade

>= 0.9 A

>= 0.8 B

>= 0.7 C

>= 0.6 D

< 0.6 F

---

Enter score: 0.95 A-

Enter score: perfect

Bad score

Enter score: 10.0

Bad score

Enter score: 0.75

C

Enter score: 0.5

F

Run the program repeatedly as shown above to test the various different values for input.

4. Write a C program to check if a number is even or odd.
5. Write a C program to find maximum two numbers.
6. Write a C program to find maximum of three numbers.
7. Write a C program to check whether the entered character is a vowel or consonant.
8. Write a C program to check whether the entered character is 'F' (Female) or 'M' (Male).
9. Write a C program to check if a year is leap year, century leap year or not a leap year.
10. Write a C program to input week number and print week day.
11. Write a C program to input month number and print the number of days in that month.
12. Write a C program to check whether the triangle, isosceles or scalene triangle.
13. Write a C program to compute roots of a quadratic equation.
14. Write a C program to calculate profit or loss.
15. Write a C program to input the basic salary of an employee and calculate its gross salary according to the following:
  - Basic Salary  $\leq$  10000 : HRA = 20%, DA = 80%
  - Basic Salary  $\leq$  20000 : HRA = 25%, DA = 90%
  - Basic Salary  $>$  20000 : HRA = 30%, DA = 95%
16. Write a C program to input electricity unit charges and calculate total electricity bill according to the given condition:
  - For first 50 units Rs. 0.50/unit
  - For next 100 units Rs. 0.75/unit
  - For next 100 units Rs. 1.20/unit
  - For unit above 250 Rs. 1.50/unit
  - An additional surcharge of 20% is added to the bill
17. Write a C program to input marks of five subjects Physics, Chemistry, Biology, Mathematics and Computer. Calculate percentage and grade according to the following:
  - Percentage  $\geq$  90% : Grade A
  - Percentage  $\geq$  80% : Grade B
  - Percentage  $\geq$  70% : Grade C
  - Percentage  $\geq$  60% : Grade D
  - Percentage  $\geq$  40% : Grade E
  - Percentage  $<$  40% : Grade F
18. Write a C program to generator a calculator using switch.
19. Write a C program to input week number and print week day using switch.
20. Write a C program to find roots of a quadratic equation using switch.
21. Write a C program to check if the entered character is vowel or consonant using switch.
22. Write a C program to print all the ASCII Values.

23. Write a C program to print natural numbers from 1 to n. ( $1+2+3+4...+n$ )
24. Write a C program to find sum of squares of natural numbers from 1 to n. ( $1^2+2^2+3^2+4^2...+n^2$ )
25. Write a C program to find the sum of  $x+x^2/2+x^3/3+.....+x^n/n$ .
26. Write a C program to print natural numbers from n to 1.
27. Write a C program to print odd numbers from 1 to n.
28. Write a C program to print even numbers from 1 to n.
29. Write a C program to check if a number is prime or not.
30. Write a C program to print all prime numbers between 1 to n.
31. Write a C program to find sum of first n natural numbers.
32. Write a C program to find sum of all odd numbers from 1 to n.
33. Write a C program to find sum of all even numbers from 1 to n.
34. Write a C program to find factorial of a number.
35. Write a C program to find fibonacci series.
36. Write a C program to generate multiplication table.
37. Write a C program to find HCF of two numbers.
38. Write a C program to find LCM of two numbers.
39. Write a C program to find the number of digits in an integer number.
40. Write a C program to calculate the sum of digits of a number.
41. Write a C program to reverse an integer number.
42. Write a C program to calculate power of a number.
43. Write a C program to check whether a given number is palindrome or not.
44. Write a C program to check whether a given number is Armstrong number or not. (Eg: 153, 371, 1634...)
45. Write a C program to print Pascal Triangle.
46. Write a C program to print the following patterns:

a. 1 1 1 1 1  
 1 1 1 1 1  
 1 1 1 1 1  
 1 1 1 1 1  
 1 1 1 1 1

```
/* Iterate through rows */
for(i=1; i<=rows; i++)
{
 /* Iterate through columns */
 for(j=1; j<=cols; j++)
 {
 printf("1");
 }

 printf("\n");
}
```

b. \*  
 \* \*

```

* * *
* * * *
* * * * *

```

```

for(i=1; i<=n; i++)
{
 /* Print i number of stars */
 for(j=1; j<=i; j++)
 {
 printf("*");
 }

 /* Move to next line */
 printf("\n");
}

```

c. 

```

* * * * *
* * * *
* * *
* *
*

```

```

/* Iterate through rows */
for(i=1; i<=rows; i++)
{
 /* Iterate through columns */
 for(j=i; j<=rows; j++)
 {
 printf("*");
 }

 /* Move to the next line */
 printf("\n");
}

```

d. 

```

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

```

```

for(i=1; i<=N; i++)
{
 // Logic to print numbers
 for(j=1; j<=i; j++)
 {
 printf("%d", j);
 }
}

```



```
 }

 printf("\n");
}
```

e. 1  
2 1  
3 2 1  
4 3 2 1  
5 4 3 2 1

```
for(i=1; i<=N; i++)
{
 // Logic to print numbers
 for(j=i; j>=1; j--)
 {
 printf("%d", j);
 }

 printf("\n");
}
```

f. 5 4 3 2 1  
4 3 2 1  
3 2 1  
2 1  
1

```
for(i=N; i>=1; i--)
{
 // Logic to print numbers
 for(j=i; j>=1; j--)
 {
 printf("%d", j);
 }

 printf("\n");
}
```