Unit 5

Control Structure

Statement

- Statements is an executable part of the program it will do some action.
- A statement is a complete direction instructing the computer to carry out some task.
- In general all arithmetic actions and logical actions are falls under statements Categories
- There are few categories of statement
  - Null Statement
  - Expression Statement
  - Compound Statement
  - Selection Statement
  - Iterative Statement
  - Jump Statement

Null Statement

- Writing only a semicolon indicates a null statement.
- Thus ; is a null or empty statement

```
  ;     // null statement
```

- This statement is generally used in for and while looping statement

Expression Statement

- It consist of an expression followed by a semicolon
- The execution statement causes the expression to be evaluated
- Few Examples for expression Statements
  - X = Y + 10 ;
  - 20 > 90;
  - a ? b : c ;
  - a = 10 + 20 * 30;

Compound Statement

- Compound statement is combination of several expression statements. Compound Statement is Enclosed within the Braces { }.
- Compound statement is also called as Block Statement.
- There is no need of any semicolon at the end of Compound Statement.
- Example for Compound Statement

-
  ```
  {
          int a=10,b=20,c;
          c = a + b;
          printf(“value of C is : %d n”,c);
  }
  ```

Selection Statement

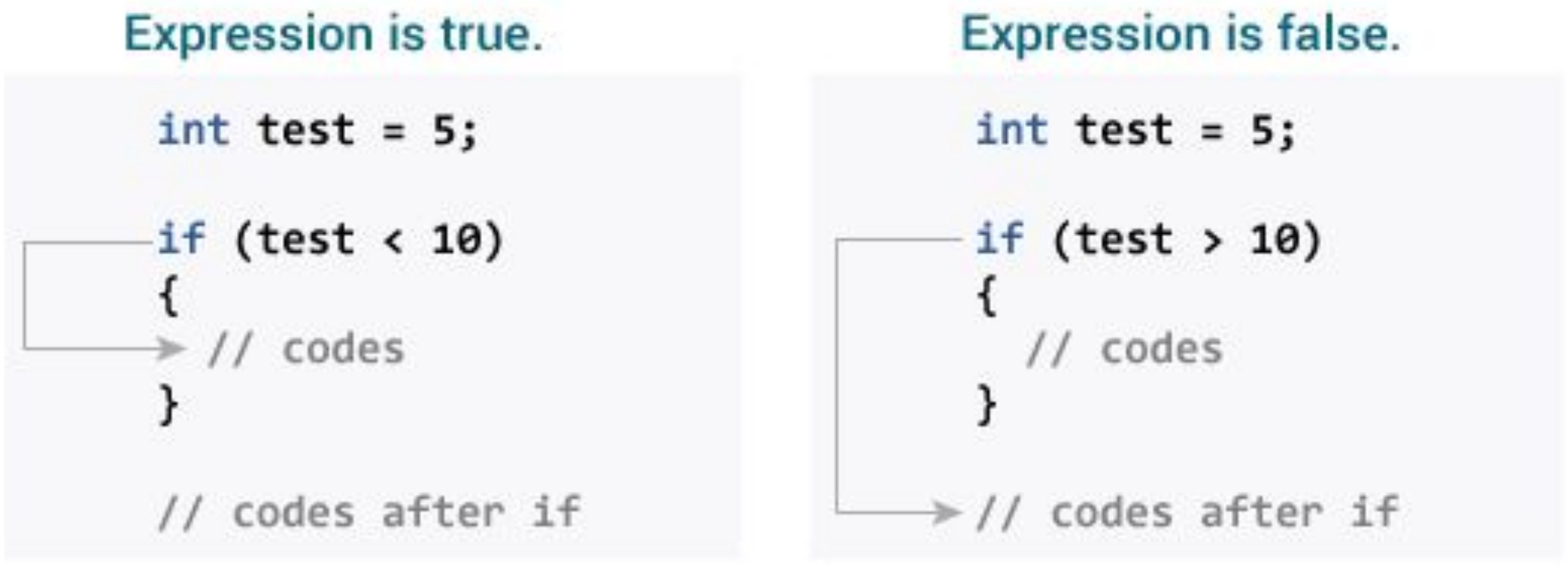
- A selection statement is a one in which one group of statement is selected from several available groups, depending on the outcome of a logical test
- Selection Statements are used in decisions making situations
  - **if**
  - **if…else**
  - **switch**

The if Statement

- Syntax

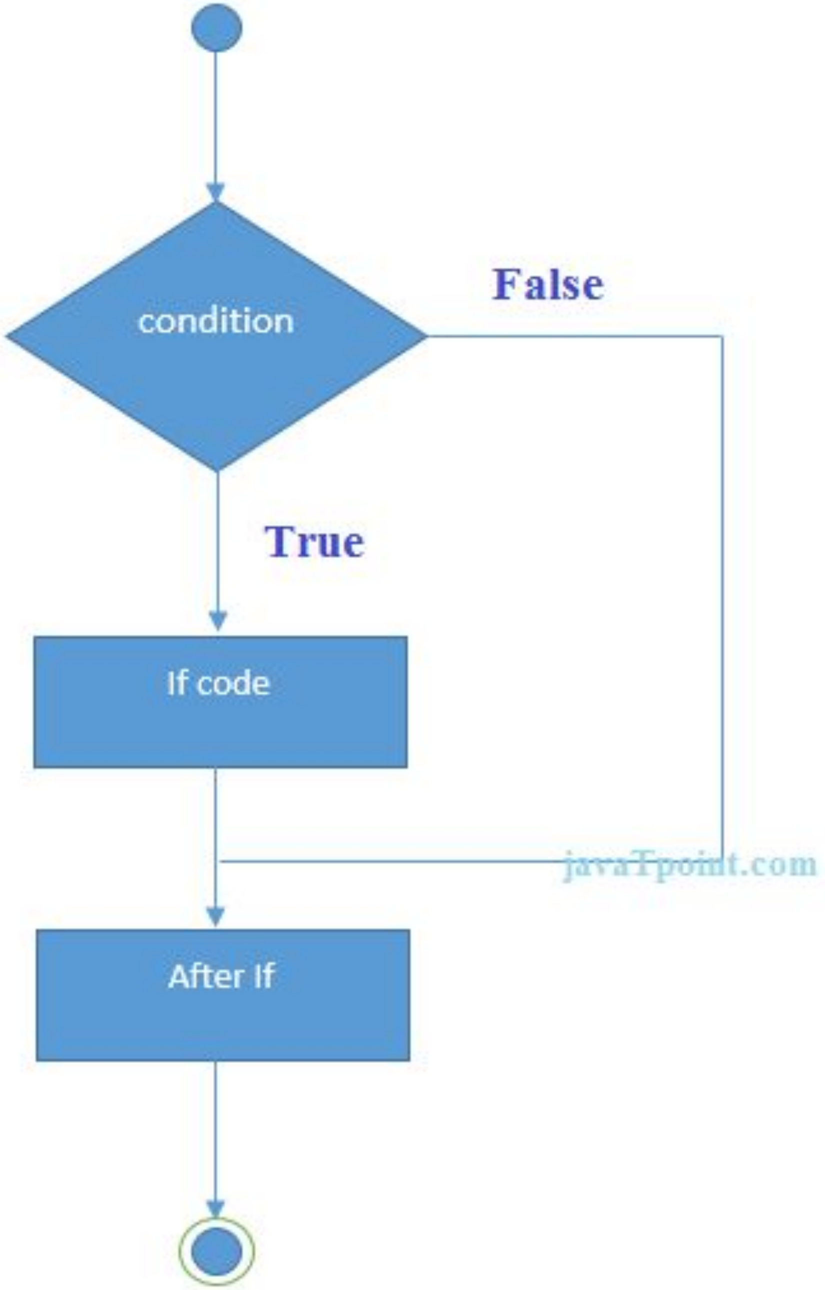
```
if (testExpression)
{
      // statement(s)
}
```

- **How if statement works?**
- The if statement evaluates the test expression inside the parenthesis.
- If the test expression is evaluated to true (nonzero), statement(s) inside the body of if is executed.
- If the test expression is evaluated to false (0), statement(s) inside the body of if is skipped from execution.



```
int test = 5;

if (test < 10)
{
  // codes
}

// codes after if
```

```
int test = 5;

if (test > 10)
{
  // codes
}

// codes after if
```


condition
False
True
If code
After If

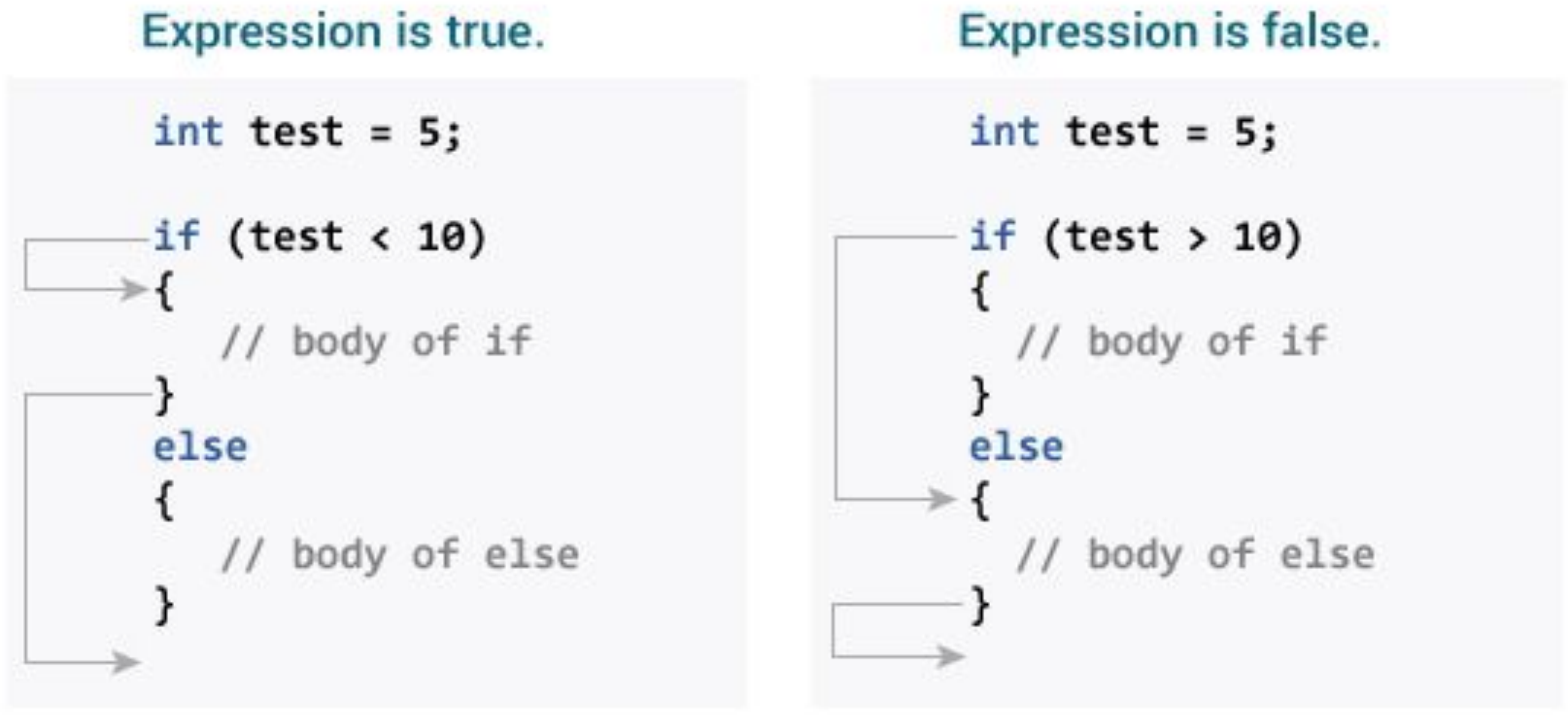
Write a program that reads a number from user and display **“Hello!!”** if the number inputted is greater than 10.

C if...else statement

- The if statement may have an optional else block. The syntax of if..else statement is:

```
if (testExpression)
{
  // statement(s) inside the body of if
}
 else
{
 // statement(s) inside the body of else
}
```

- **How if...else statement works?**
- If test expression is evaluated to true,
- statement(s) inside the body of if statement is executed
- statement(s) inside the body of else statement is skipped from execution.
- If test expression is evaluated to false,
- statement(s) inside the body of else statement is executed
- statement(s) inside the body of if statement is skipped.

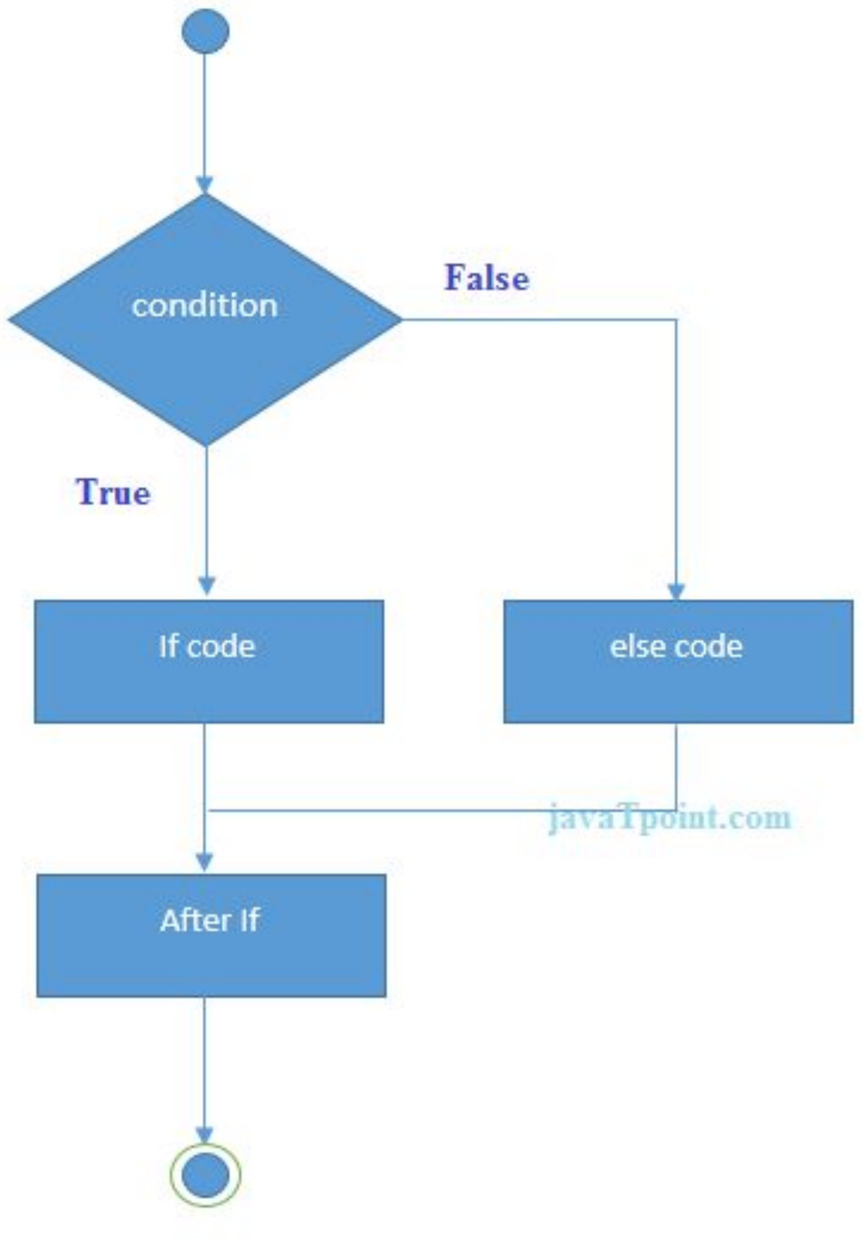


Expression is true.

```
int test = 5;

if (test < 10)
{
  // body of if
}
else
{
  // body of else
}
```

Expression is false.

```
int test = 5;

if (test > 10)
{
  // body of if
}
else
{
  // body of else
}
```


condition
False
True
If code
else code
javaTpoint.com
After If

Example

```
#include<stdio.h>
void main()
{
int number;
printf("enter a number:");
scanf("%d",&number);
if(number%2==0)
{
	printf("%d is even number",number);
}
else
{
	printf("%d is odd number",number);
}
}
```

Write a program that reads a student score in C programming in the exam. If score is greater than 20, display “**congratulation!! You have pass the exam**” otherwise display “**Sorry!!! You have fail the exam**”.

if...else Ladder (if...else if....else Statement)

- The if...else statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.
- The if...else ladder allows you to check for multiple test expressions and execute different statement(s).

- **Syntax of nested if...else statement.**

```
if (testExpression1) {
        // statement(s)
        }
else if(testExpression2) {
        // statement(s)
        }
else if (testExpression 3) {
        // statement(s)
        }
else {
        // statement(s)
}
```

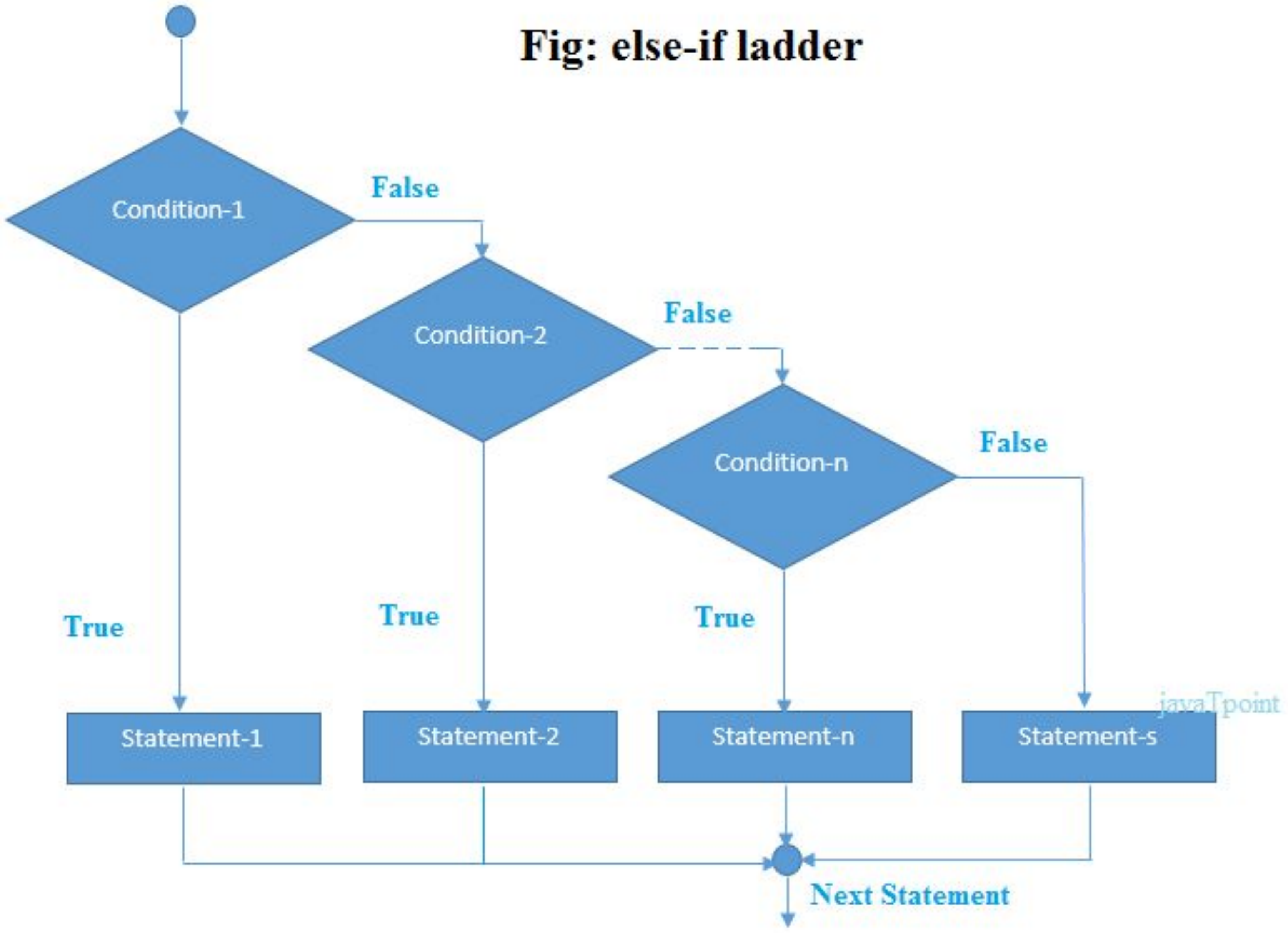


Fig: else-if ladder

Write a program that accepts income of a person in annual basis and display his class according to following conditions

below 50,000	class C
50,001-3,00,000	class B
3,00,001-10,00,000	class A
above 1000000	class A+

Nested if...else

- It is possible to include if...else statement(s) inside the body of another if...else statement.
- This program below relates two integers using either <, > and = similar like in if...else ladder example. However, we will use nested if...else statement to solve this problem.

Example

```
#include <stdio.h>
#include<conio.h>
void main()
{
    int n1, n2;
    printf("Enter two integers: ");
    scanf("%d %d", &n1, &n2);

    if (n1 >= n2)
    {
      if (n1 == n2)
      {
        printf("Result: %d = %d",n1,n2);
      }
      else
      {
        printf("Result: %d > %d", n1, n2);
      }
```

Write a program to calculate whether the given year is leap year or not.

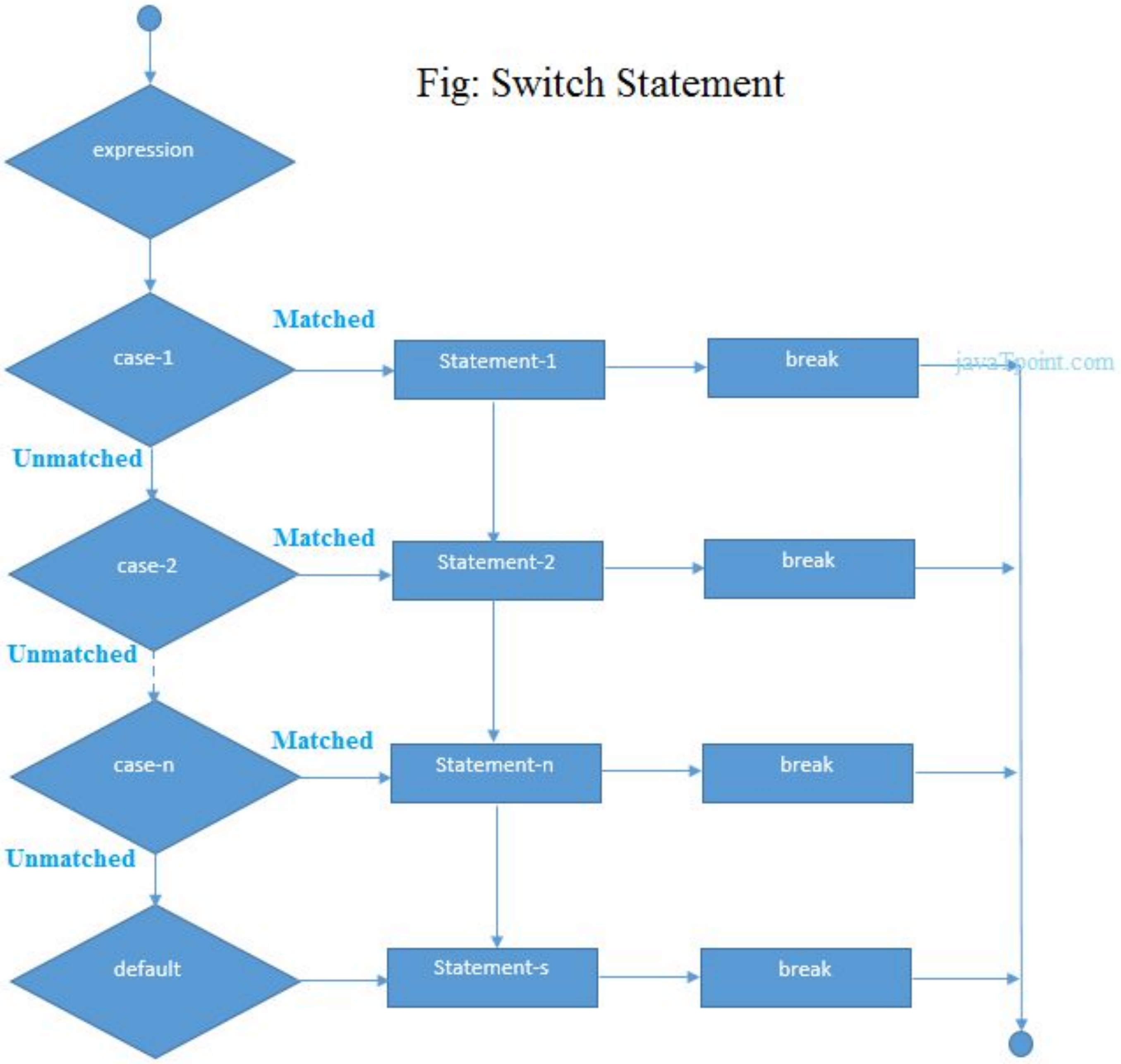
C switch...case Statement

- The if..else..if ladder allows you to execute a block code among many alternatives. If you are checking on the value of a single variable in if...else...if, it is better to use switch statement.
- The switch statement is often faster than nested if...else (not always). Also, the syntax of switch statement is cleaner and easy to understand.

Syntax of switch...case

```
switch (n)
{
   case constant1:
      // code to be executed if n is equal to constant1;
      break;

   case constant2:
      // code to be executed if n is equal to constant2;
      break;
      .
      .
      .
   default:
      // code to be executed if n doesn't match any constant
}
```

Fig: Switch Statement



Write a menu driven program for restaurant menu

1. Momo
2. Pizza
3. Noodles
4. Burger
5. Fry Rice
6. Exit

Example

```
#include <stdio.h>
int main() {
        int ID = 500;
        int password = 000;
        printf("Plese Enter Your ID:\n ");
        scanf("%d", & ID);
        switch (ID) {
            case 500:
                printf("Enter your password:\n ");
                scanf("%d", & password);
switch (password) {
          case 000:
            printf("Welcome Dear Programmer\n");
            break;
          default:
            printf("incorrect password");
            break;
        }
        break;
      default:
        printf("incorrect ID");
        break;
    }
}
```

Repetitive Statement

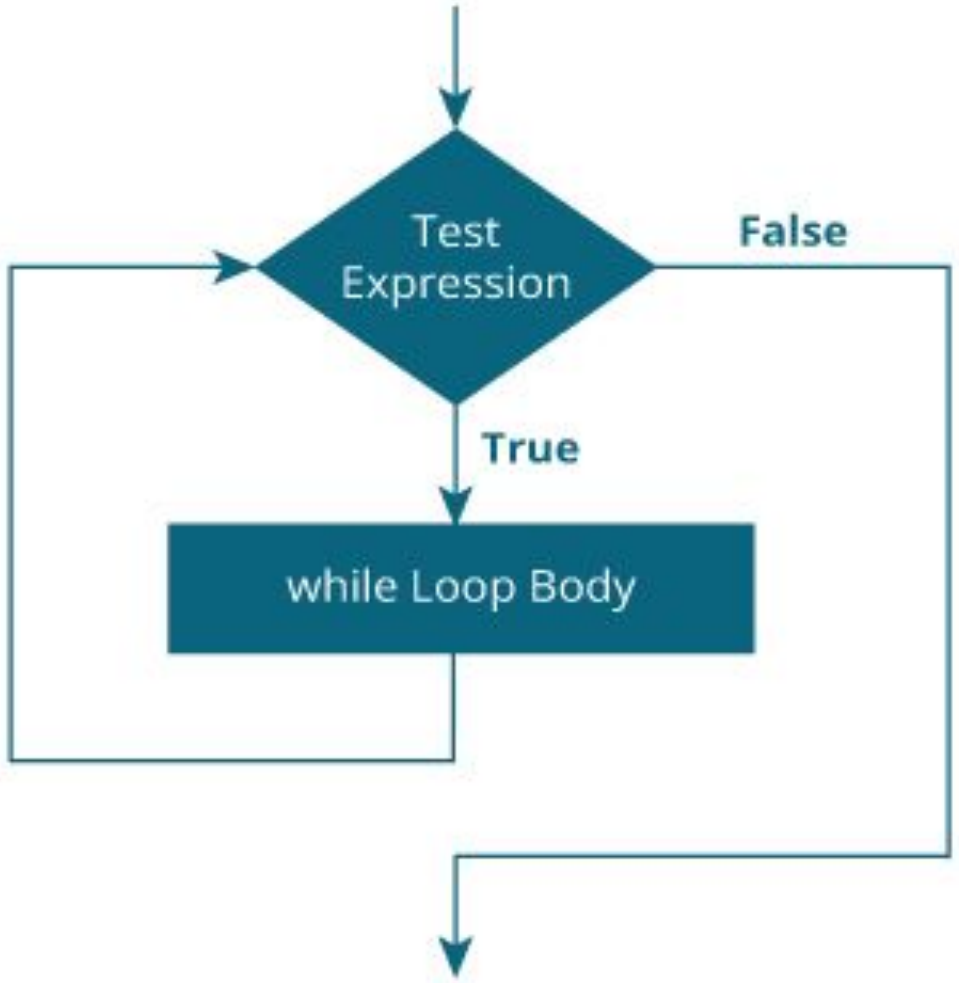
- A repetition statement allows you to specify that an action is to be repeated while some condition remains true.
- Loops are used when we want to execute a part of program or block of statement several times
- There are three types of loops
  - The while loop
  - The do while loop
  - The for loop
- Each loop consist of two segments, one is known as the control statement and the other is the body of the loop
- Loop may be classified either entry-controlled loop or exit-controlled loop.

The while loop

- The syntax of a while loop is:

```
while (testExpression)
{
 //codes
}
```

- **How while loop works?**
- The while loop evaluates the test expression.
- If the test expression is true (nonzero), codes inside the body of while loop is executed. The test expression is evaluated again. The process goes on until the test expression is false.
- When the test expression is false, the while loop is terminated.


Test Expression
False
True
while Loop Body

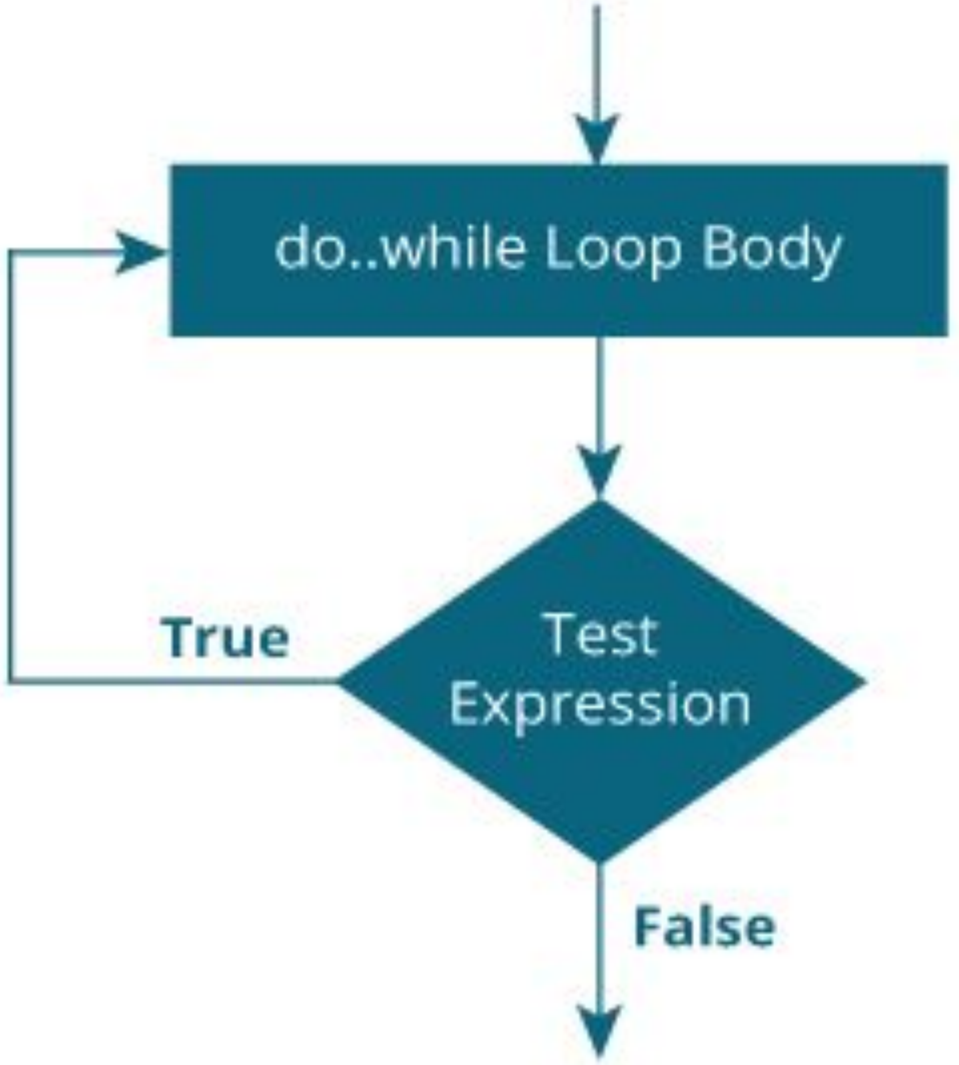
1. Write a program to print Hello World five times.
2. WAP to print all even numbers between 30-60
3. WAP to find sum of first 10 natural numbers
4. WAP to check whether the given number is prime or not.

The do...while loop

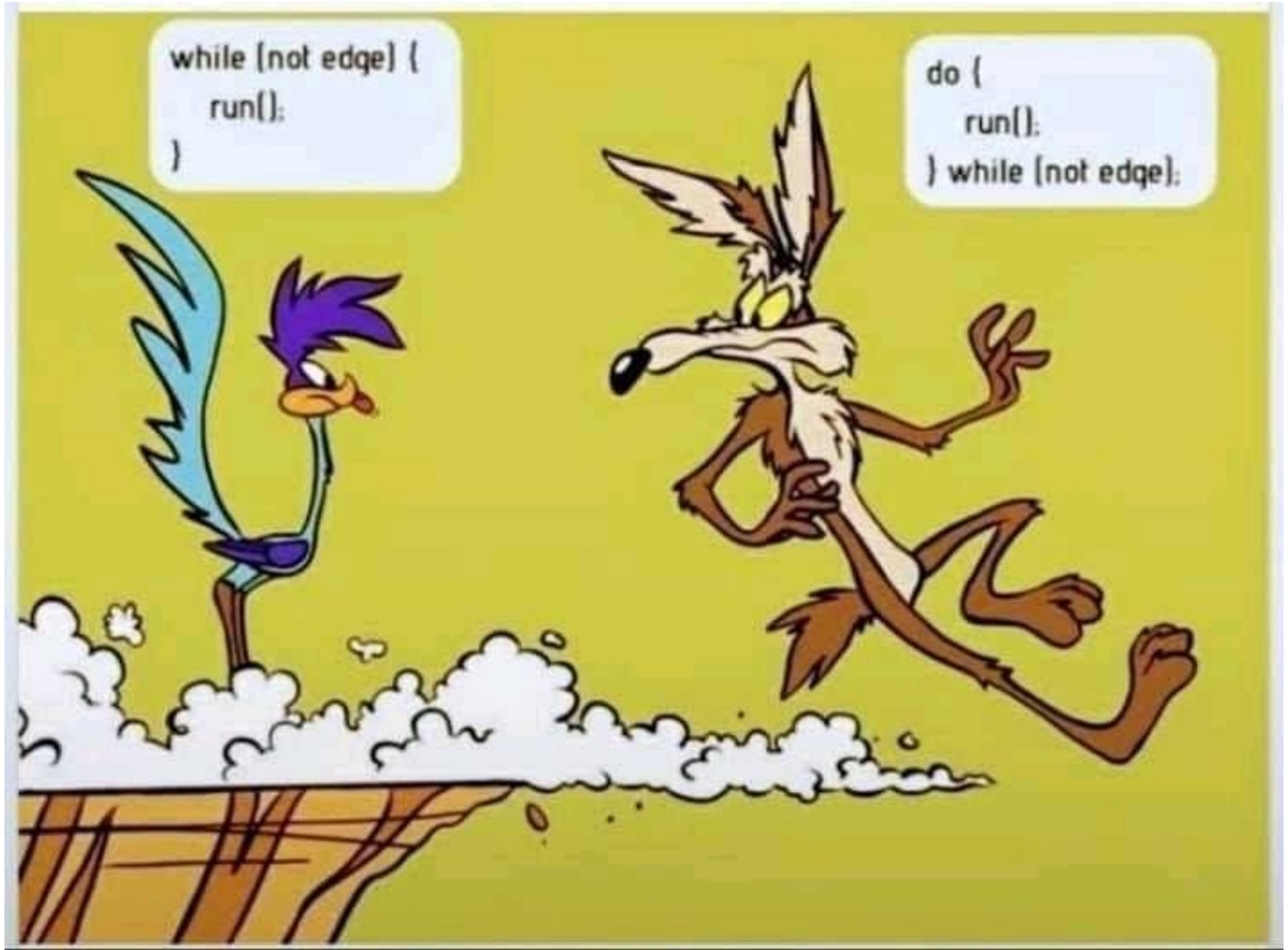
- The do..while loop is similar to the while loop with one important difference. The body of do...while loop is executed once, before checking the test expression. Hence, the do...while loop is executed at least once.
- **do...while loop Syntax**

```
do
{
        // codes
    } while (testExpression);
```

- **How do...while loop works?**
- The code block (loop body) inside the braces is executed once.
- Then, the test expression is evaluated. If the test expression is true, the loop body is executed again. This process goes on until the test expression is evaluated to 0 (false).
- When the test expression is false (nonzero),
  the do...while loop is terminated.


do..while Loop Body
True
Test Expression
False

- Write a program to print Hello World five times using do while loop.



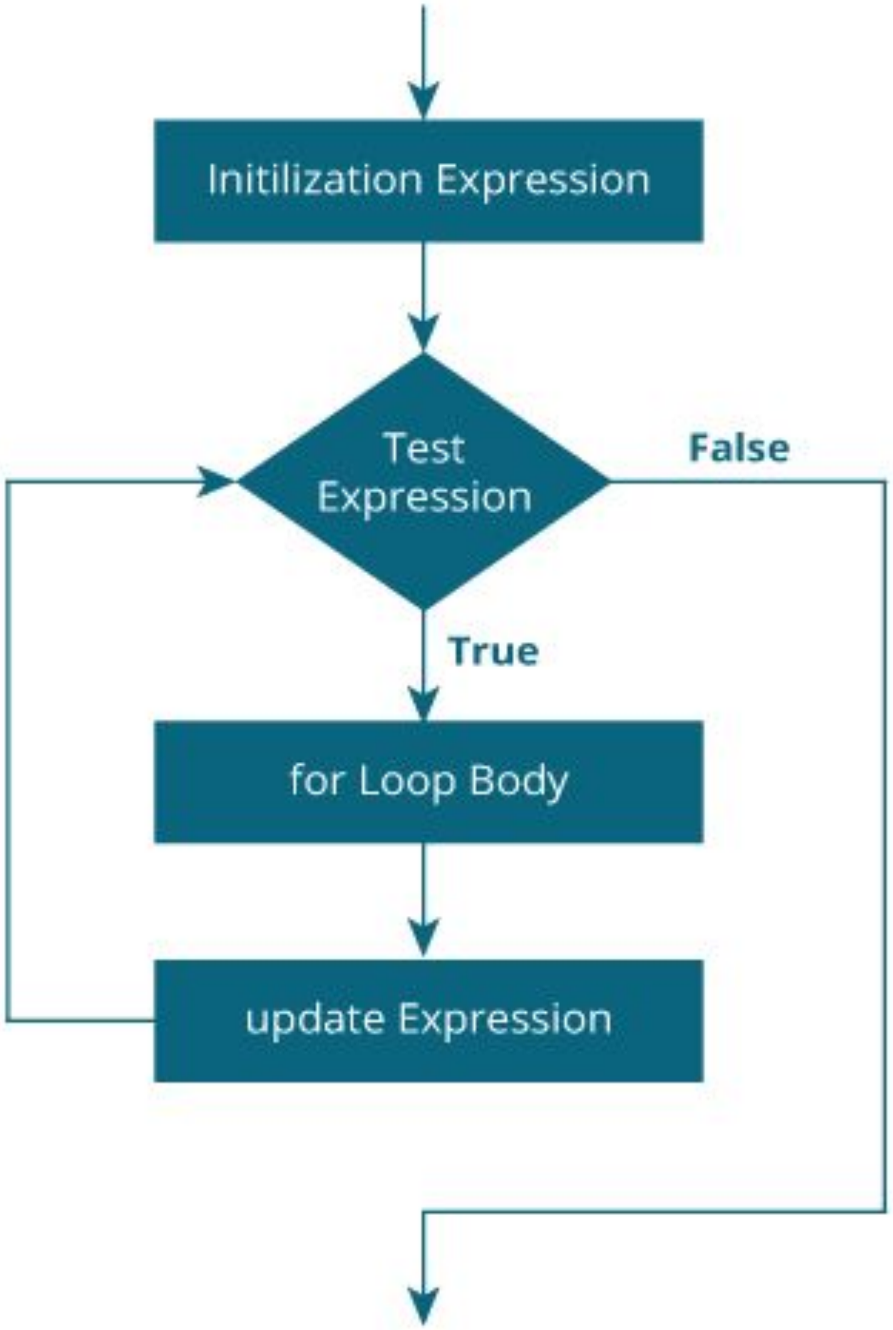
Difference between while & do...while...🤣

The for loop

- The syntax of for loop is:

```
for (initialization; testExpression; updateStatement)
{
     // codes
}
```

- **How for loop works?**
- The initialization statement is executed only once.
- Then, the test expression is evaluated. If the test expression is false (0), for loop is terminated. But if the test expression is true (nonzero), codes inside the body of for loop is executed and the update expression is updated.
- This process repeats until the test expression is false.
- The for loop is commonly used when the number of iterations is known.


Initilization Expression
Test Expression
False
True
for Loop Body
update Expression

- Write a program to print Hello World five times using for loop.

Nested loop in C

- C programming allows to use one loop inside another loop. The following section shows a few examples to illustrate the concept.
- The syntax for a **nested for loop** statement in C is as follows

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

- The syntax for a **nested while loop** statement in C programming language is as follows

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

- The syntax for a **nested do...while loop** statement in C programming language is as follows −

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

C Jump Statements

- Jump statements are used to interrupt the normal flow of program.
- **Types of Jump Statements**
  - Break
  - Continue
  - GoTo

Break Statement

- The break statement is used inside loop or switch statement.
- When compiler finds the break statement inside a loop, compiler will abort the loop and continue to execute statements followed by loop.

Example

```
#include<stdio.h>
void main()
{
    int a=1;
    while(a<=10)
    {
        if(a==5){
                break;
        }
        printf("\nHello %d.",a);
        a++;
    }
    printf("\nEnd of Program.");
}
```

Continue Statement

- The continue statement is also used inside loop.
- When compiler finds the continue statement inside a loop, compiler will skip all the following statements in the loop and resume the loop.

Example

```
#include<stdio.h>
void main()
{
   int a=0;
   while(a<10)
   {
      a++;
      if(a==5){
           continue;
      }
      printf("\nHello %d.",a);
   }
   printf("\nEnd of Program.");
}
```

goto Statement

- The goto statement is a jump statement which jumps from one point to another point within a function.
- **Syntax of goto statement**

```
goto label;
 - - - - - - - - - - -
  - - - - - - - - - - -
label:
 - - - - - - - - - - -
 - - - - - - - - - - -
```

- In the above syntax, label is an identifier. When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code after it.

Example

```
#include<stdio.h>
void main()
{
   printf("\nStatement 1.");
   printf("\nStatement 2.");
   printf("\nStatement 3.");
   goto last;
   printf("\nStatement 4.");
   printf("\nStatement 5.");

   last:
   printf("\nEnd of Program.");
}
```