Flow Control in C

Unit 2

1. If statement

When situations come in our real life we need to make some decisions and based on these decisions, we decide what should we do next. Either we should do this thing-1 or we should do this thing-2.

Similar situations occur in programming also where we need to make some decisions and based on these decisions we execute the next block of statement.

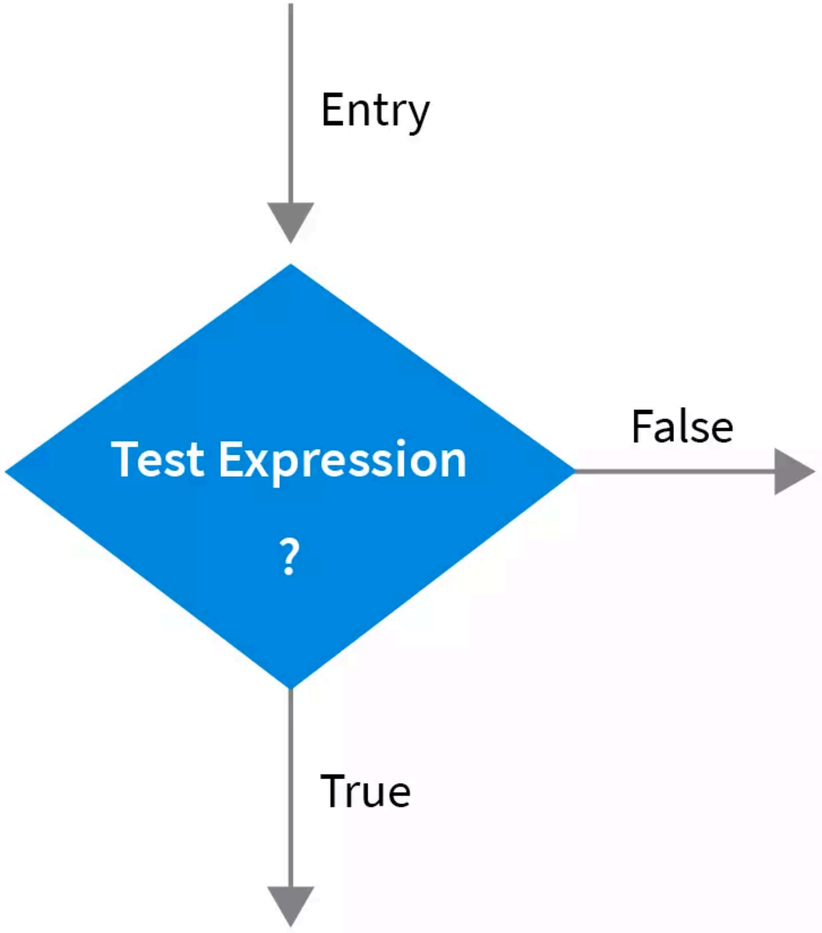
If statement

What is if statement ?

- The if statement is a simple decision-making and branching statement and it is used to control the flow of the program execution.
- If statement is a two-way branching statement and it involves boolean expression.
    - It means depending on the condition the control is transferred either to the true block or false block. It is also called a control statement.f

If statement

Syntax: if(expression)



If statement

How does an if statement work?

- If statement allows to evaluate the test expression first and then, building upon whether the condition of the expression is true(non-zero) or false(zero), it shifts the control to a particular block of statement.
- This point of the program has two paths to follow, one path for the true condition and the other path for the false.

If statement

How does an if statement work?

If a certain condition is true then it will execute a block of the statement below it otherwise not.

Some examples of control statement, using if statement in C:

```
if(gender is Female)
    Person is Female

if(age is more than 60)
    person is retired
```

If statement

Types of If statement

if statement may be implemented in different forms depending on the complexity of testing conditions to be evaluated.

1. Simple if Statement
2. if-else Statement
3. Nested if-else Statement
4. else-if Ladder

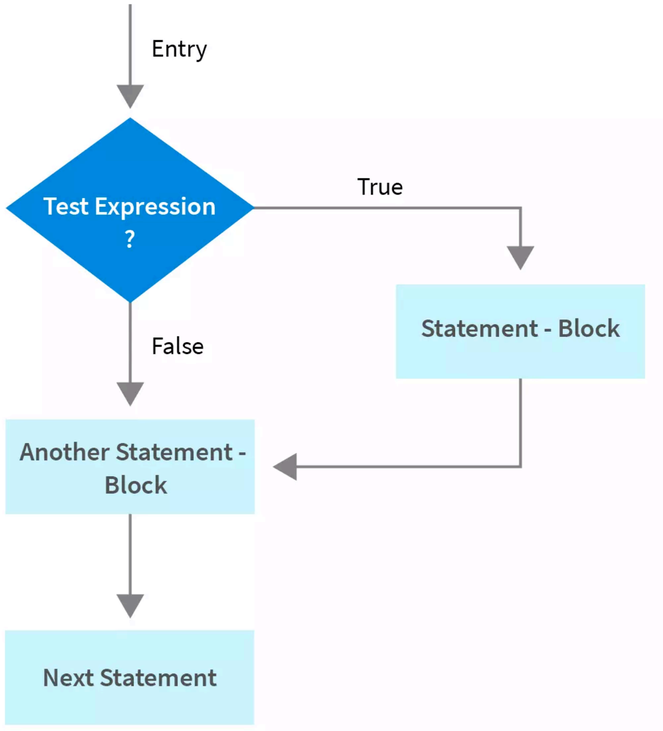
If statement

1. Simple if statement

- If the given condition is true then the statements inside the body of “if” will be executed.
- If the condition is false then the statements inside the body of “if” will be skipped.

The general form of a simple if statement is given below:

```
if(condition)
{
    True block of the statement;
}
another block of statement;
```



If statement

1. Simple if statement

```
if(condition)
{
    True block of the statement;
}
another block of statement;
```

- In the given general form of simple if statement, the 'True block of the statement' can be a either single statement or it can be also a group of statements.
- If the given condition of expression is true:
  -- The 'True block of the statement' will be executed;
  -- Otherwise, the 'block of the statement' will be skipped and the execution of the program will jump to the 'another block of statement'.

If statement

1. Simple if statement Example

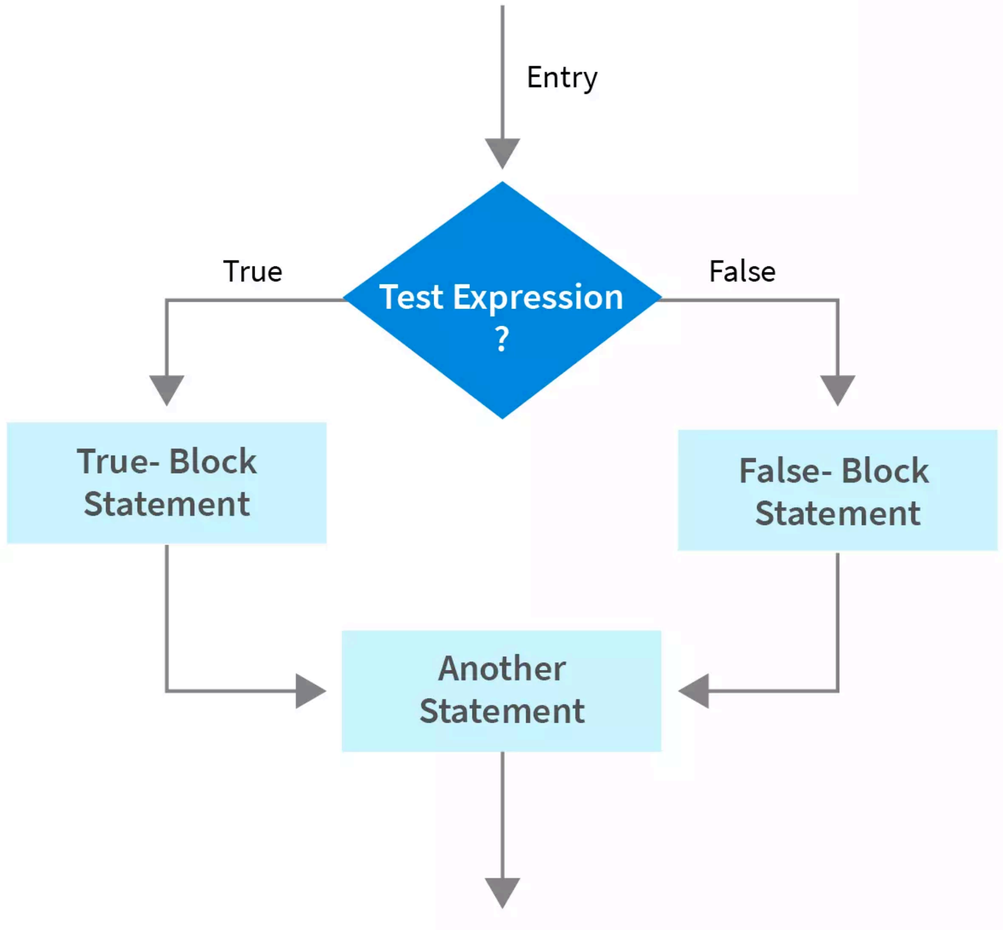
```
#include<stdio.h>
int main(){
    int n;
    printf("Enter a Number:”);
    scanf("%d",&n);
    for(int i = 1; i <= n; i++){
        if(i%2 == 0){
            continue;
        }
        printf("%d\n", i);
    }
    return 0;
}
```

```
Enter a Number: 10
1
3
5
7
9
```

If statement

2. If.....else statement

```
if(test expression) {
   true-block statement(s)
}

else {
   false-block statement(s)
}
   another-statement
```



If statement

2. If.....else statement Example

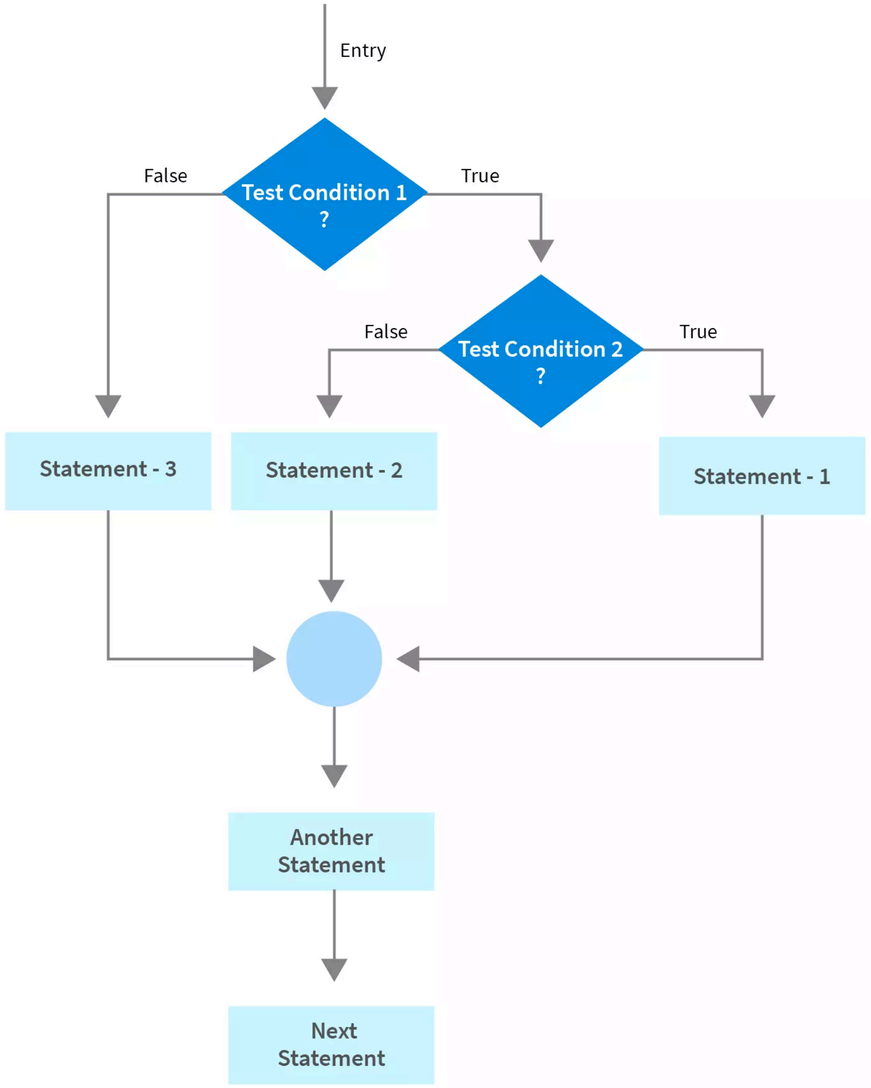
```c
#include <stdio.h>
int main() {
    int n;
    printf("Enter the number: ");
    scanf("%d", &n);

    if  (n%2 == 0) {
        printf("%d is an even number.",n);
    }
    else {
        printf("%d is an odd number.",n);
    }

    return 0;
}
```

Output:

```
Enter the number: 6
6 is an even number.
```

If statement

3. Nested If-else Statement



When multiple decisions are involved, we can use more than one if-else statement in nested form. In the flowchart below we can see:

- If condition-1 is false the statement-3 will be executed, and condition-1 is true then the control is transferred to condition-2.
- If condition-2 is true, statement-1 will be executed; otherwise, statement-2 will be evaluated and then the control is transferred to another block of statement.

If statement

3. Nested If-else Statement: Example

```c
#include <stdio.h>
int main()
{
    int n1, n2;
    printf("Input the value of n1:");
    scanf("%d", &n1);
    printf("Input the value of n2:");
    scanf("%d",&n2);
    if (n1 != n2)
    {
  printf("n1 is not equal to n2\n");
  //Nested if else
  if (n1 > n2)
  {
    printf("n1 is greater than n2\n");
  }
  else
  {
    printf("n2 is greater than n1\n");
  }
    }
    else
    {
  printf("n1 is equal to n2\n");
    }
    return 0;
}
```

Output:

```
Input the value of n1:90
Input the value of n2:80
n1 is not equal to n2
n1 is greater than n2
```

If statement

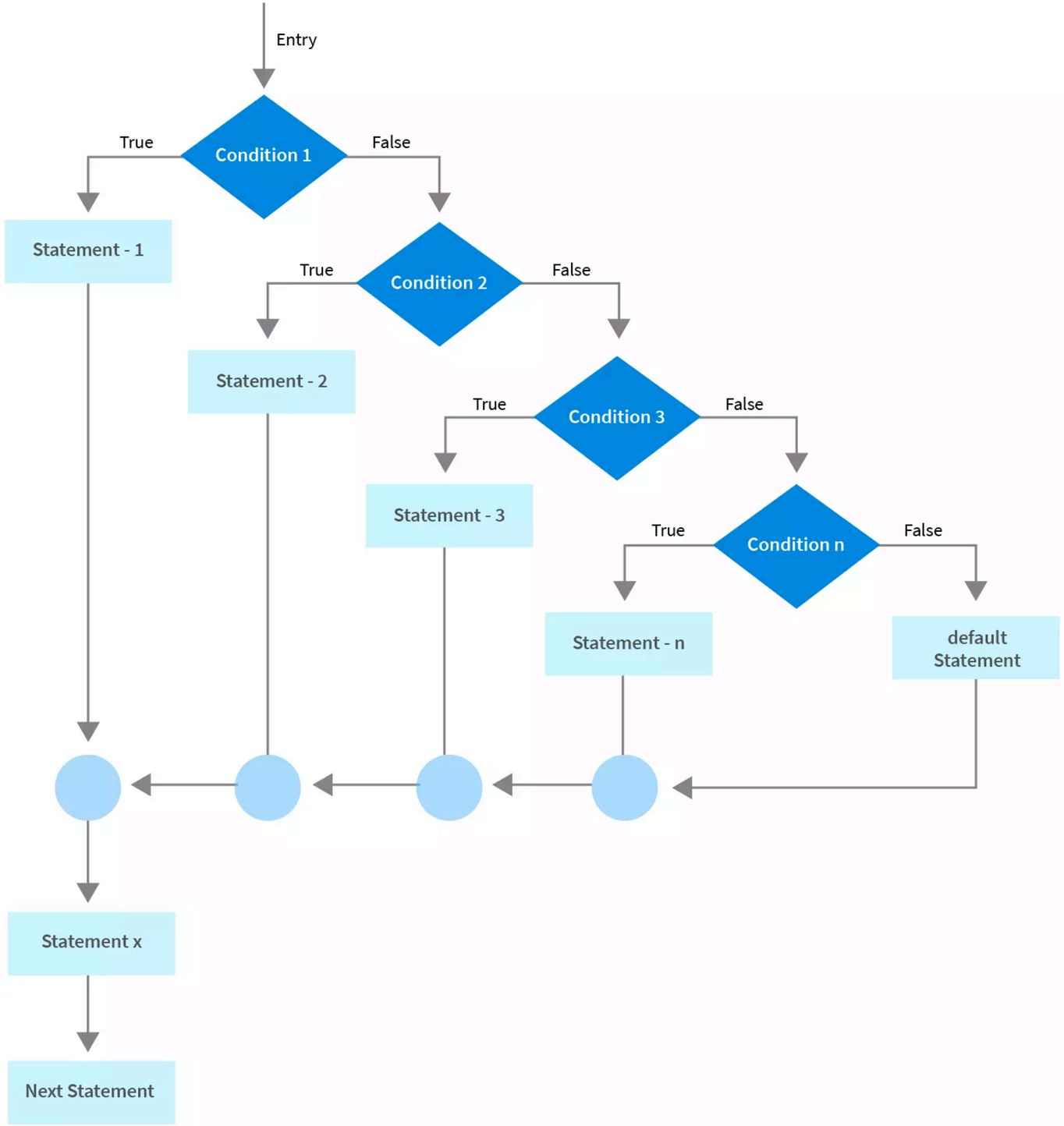
4. Else..if ladder

- There is another way of setting up if statement together when multi-way decisions are involved.
- A multi-way decision is a series of ifs in which the statement linked with each else statement is an if statement.

```
if(test expression) {
   true-block statement
}
else if(test expression){
   block of statement
}
else if(test expression){
 block of statement
}
else {
   false-block statement
}
```

If statement

4. Else..if ladder



If statement

4. Else..if ladder: Example

```c
#include <stdio.h>

int main() {
    char button;
    printf("Input a character:");
    scanf("%c", &button);
    if(button == 'a')
    {
        printf("Hello");
    }
    else if(button == 'b')
    {
        printf("Namastey");
    }
    else if(button == 'c')
    {
        printf("Hola");
    }
    else if(button == 'd')
    {
        printf("Ciao");
    }
    else if(button == 'e')
    {
        printf("Salut");
    }
    else {
        printf("I am still learning more...");
    }
    return 0;
}
```

Output:

```
Input a character: e
Salut
```

If statement

Important Points Need to Remember

- Never put semicolon just after the if(expression).
- A non-zero value is considered as true and a zero(0) value is considered as false in C.
- We **can use more than one condition inside the if statement using the logical operator**.
- We should always **use braces on separate lines to identify a block of statements.**
- We should **always align the opening and closing braces**.
- **Do not ignore placing parentheses for the if condition/expression**.
- Be aware of dangling else statements.
- Avoid using operands that have side effects in a logical binary expression such as (a-- && ++b). The second operand may not be evaluated in any case.

If statement

Advantages and Disadvantages

Advantages

- It checks every condition, It also makes a program more robust by allowing only a portion of code to run if a condition has been met.
- If statements are required to make decisions in the programs. Technically, this could be done with loops and goto(break). But in reality, the if statement is most concise.

Disadvantages

During execution as it checks every condition:

- This makes it difficult to test the code.
- It is a little bit complex in terms of reading conditions of programs as compared to the switch case.
- It takes more time for each possible condition because it does fall through all the if statements compared to switch case.

2. Switch statement in C

- The Switch Statement in C is another C programming language's **decision-making statement.**
- The Switch statement in the C programming language **can execute statements from a wide array of possible choices based on a provided condition**.
- You can't perform this with the **if-else statement** of the C programming language, which **lets you choose only between two possibilities based on a given condition**.
- The creators of the C programming developed the Switch statement for the situations where the if-else statement was not that efficient and took a toll on the readability of the code.

Switch Statement

What is Switch statement in C?

Switch Statement

What is Switch statement in C?

The Switch statement of the C programming language is one of the decision-making statements of the C programming language. You might already be familiar with the if-else statement of the C programming language, another decision-making statement of the C programming language.

One might raise the question,

- Why are there two decision-making statements in the C programming language?
- What is the difference between them?
- When to use the if-else statement and when to use the switch statement?

The answer is quite simple,

There are two decision-making statements because they both perform well in different situations and are necessary at times.

Switch Statement

Syntax of Switch statement in C

```
switch(expression)
{
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......

default:
 //code to be executed if all cases are not matched;
}
```

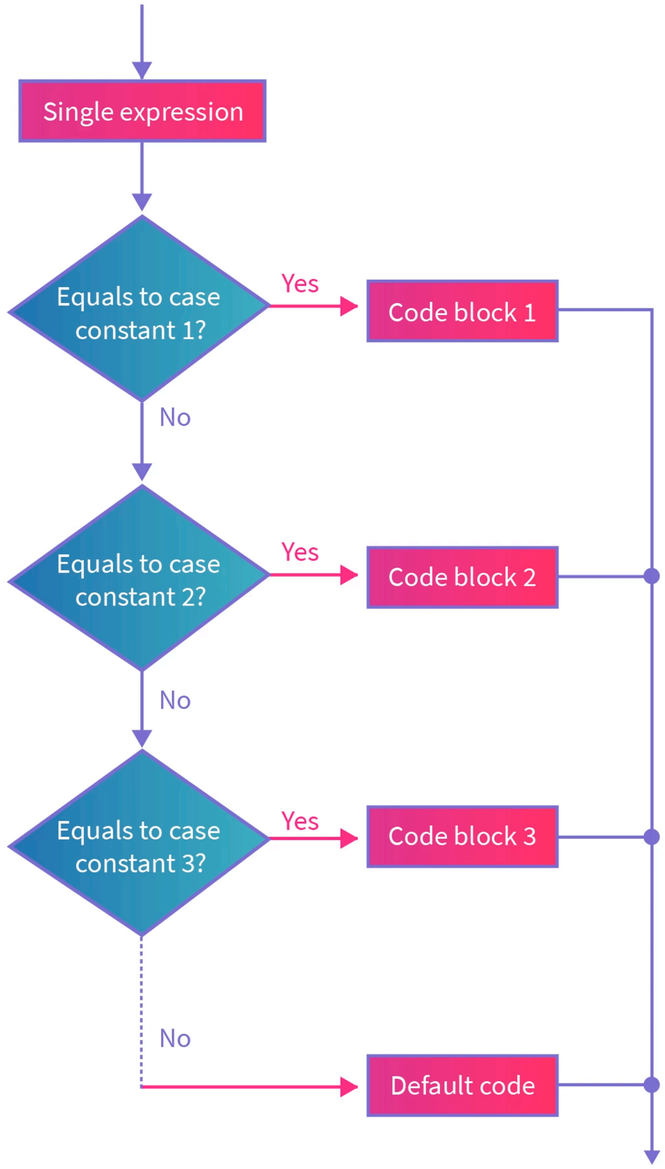
Switch Statement

Switch statement in C

- The **expression** in the syntax **can be Arithmetic, Relational or Logical expression.**
- Still, the **condition is that the programmer should provide an expression that always returns a single value** as its result, and the **result should not be a float or double**.
- The **values defined in the case statements should always be constants**; you can't use a variable with a case statement.
- The **break** and the **default** statements are both optional.

Switch Statement

How Switch statement working in C



Switch Statement

Rules for Switch Statement in C Language

the result of the expression should not return a Float or decimal value

```
switch(x==5) // this is valid.
switch(x/3) // this is not valid.
```

Case values must not be duplicate

```
switch(x)
{
    case a:
        // set of statement
    case a:
        //set of statements
}// this kind of switch statement will throw you a compile-time error.
```

Switch Statement

Rules for Switch Statement in C Language

You can't define ranges within a case statement, nor can you use a single case label for more than one value. A case label can hold only one value.

```
switch(x)
{
    case 1,10:
        // set of statements
    case 2-5:
        //set of statements
}// these 2 case statements will also throw a compile time error
```

Switch Statement

Important Facts About Switch statement in C

You should be aware of some essential facts before getting your first hands-on experience with the switch statement of the C programming language. They are listed below:

- **The break statement at the end of each case label is optional**. However, the control will sequentially pass through all other case labels without the break statement. At the end of the case label, the break statement will get the control out of the switch statement, so The Compiler will not execute the other case statements sequentially.
- **The default label for a switch statement is an optional one**. You can use a switch statement without a default label, which would work fine.
- Just like the if-else statement, **you can also nest more than one switch statement inside another. But this would make the code complex and less readable,** so it is generally avoided.

```
switch(x)
{
    switch(y) // this is valid
    {
        //set of statements
    }
}
```

Switch Statement

Switch statement in C: Example

Input:

```
3
5 4
```

Output:

```
5 * 4 = 20
```

```c
#include <stdio.h>
#include <conio.h>
void main()
{
    int choice;
    printf("Select an option from the list below:\n");
    printf("1. Addition\n");
    printf("2. Subtraction\n");
    printf("3. Multiplication\n");
    printf("4. Division\n");
    printf("5. Modulus\n");
    printf("6. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    int a, b;
    // read two numbers
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    switch (choice)
    {
    case 1:
        printf("%d + %d = %d\n", a, b, a + b);
        break;
    case 2:
        printf("%d - %d = %d\n", a, b, a - b);
        break;
    case 3:
        printf("%d * %d = %d\n", a, b, a * b);
        break;
    case 4:
        printf("%d / %d = %d\n", a, b, a / b);
        break;
    case 5:
        printf("%d %% %d = %d\n", a, b, a % b);
        break;
    case 6:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice\n");
    }
}
```

Switch Statement

Difference between if..else and Switch statement in C

```
if (condition) {
    // Block of code if condition true
} else {
    // Block of code is condition false
}
```

```
switch (condition) {
case identifier1:
//block of code
break;

case identifier2:
//block of code
break;

case identifier3:
//block of code
break;

case identifiern:
//block of code
break;

default:
//block of code
}
```

Switch Statement

Difference between if..else and Switch statement in C

```
if (month == 'January' || month == 'March' || month == 'May' || month == 'July' || month == 'August' || month == 'October' || month == 'December') {
    cout << '31';
} else if (month == 'February') {
    cout << '28 or 29';
} else {
    cout << '30';
}
```

```
switch (month) {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        cout << "31";
        break;
    case 2:
        cout << "28 or 29";
        break;
    case 4:
    case 6:
    case 9:
    case 11:
        cout << "30";
        break;
    default:
        cout << "Not a valid month!";
        break
}
```

3. Ternary Operator

What is ternary operator?

Ternary Operator in C is an operator which takes three operands or variables, unlike the other operators which take one or two operands. Ternary operator in C is also known as the Conditional Operator. **It is a way to shorten the simple if-else code of the block.**

Using the Ternary operator in c is a way to shorten the if-else code block in C/C++.

Ternary Operator in C takes three arguments:

1. The **first argument** in the Ternary Operator in C is the **comparison condition**.
2. The **second argument** in the Ternary Operator in C is the **result if the condition is true**.
3. The **third argument** in the Ternary Operator in C is the **result if the condition is false**.

So, according to the above three arguments in the ternary operator in c, we can say that the Ternary operator in C allows us to execute different code depending on the first argument, i.e. based on condition.

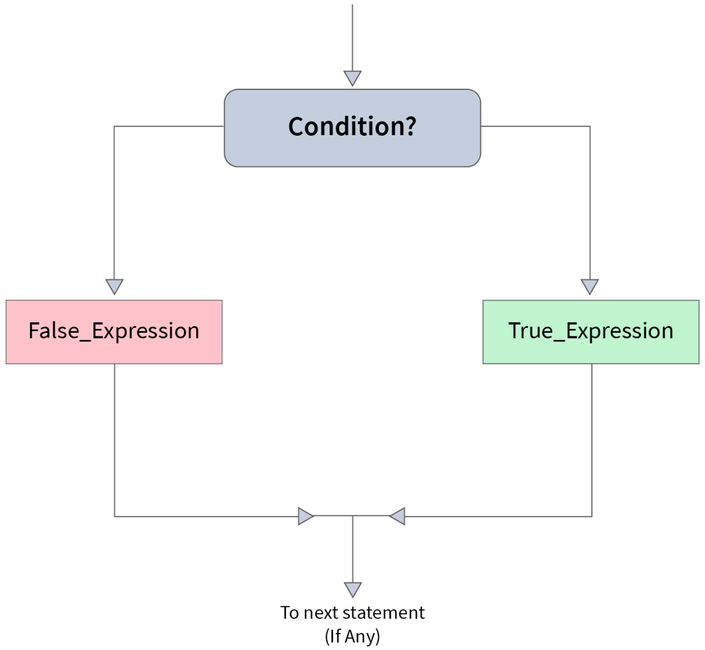
Ternary Operator

Syntax of Ternary operator

```
exp1 ? exp2 : exp3
```

Working of Syntax:

- If the condition in the ternary operator is met (true), then the exp2 executes.
- If the condition is false, then the exp3 executes.



NOTE: Ternary operators in C, like if-else statements, can be nested

```c
int mxNumber = 10 > 15 ? 10 : 15;
```

Ternary Operator

Ternary operator: Example 1 using if else

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

int main() {
  int num1, num2, mxNumber;
  num1 = 10;
  num2 = 20;

  if (num1 >= num2) // condition checking
  {
    mxNumber = num1; // if condition true
  } else {
    mxNumber = num2; // if condition false
  }

  printf("Maximum Number from %d and %d is %d", num1, num2, mxNumber); //output
  return 0;
}
```

Output:

```
Maximum Number from 10 and 20 is 20
```

Ternary Operator

Ternary operator: Example 1 using switch

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

int main()
{
    int num1, num2,mxNumber;
    num1 = 100;
    num2 = 20;

    // result = condition ? exp1 : exp2;
    // isn't the if-else block ? just in one line.

    mxNumber = num1 >= num2 ? num1 : num2;

    printf("Maximum Number from %d and %d is %d",num1,num2, mxNumber); //output
    return 0;
}
```

Output:

```
Maximum Number from 100 and 20 is 100
```

4. go to statement

- Goto statement in C is a jump statement that is used to jump from one part of the code to any other part of the code in C. Goto statement helps in altering the normal flow of the program according to our needs. This is achieved by using labels, which means defining a block of code with a name so that we can use the goto statement to jump to that label.
- Goto statement can be used in two ways:
    1. either to skip some lines and move to a block below in the code
    2. to repeat some lines of code by going above in the code.

go to statement

What is go to statement in C

- we may need to jump from one line to another in our program to either goto skip a few lines or to first execute some block of code and then arrive at this one. This is exactly where the goto statement is used.
- Goto statement is a form of jump statement, it is used to jump from one block to another block during execution. It is often also termed as an *unconditional jump statement*.
- We can use the goto statement in C to create a loop in the program (though it isn't its purpose), its main purpose is to dictate the execution of the program. This means that the goto statement can change the flow of the program, the sequence in which instructions will be executed. We can also use the goto statement in C to break out of multiple loops instead of applying many break conditions.
- Nonetheless, we usually avoid goto as it makes the program complex to understand and reduces readability.

go to statement

Syntax of go to statement in C

The syntax of goto statement in C can be broken into two parts:

1. Defining the Label

```
label_name:
```

- The label_name is used to give a name to a block of code, hence it acts as an identifier for that block. When a goto statement is encountered, the program's execution control goes to the label_name: and specifies that the code will be executed from there.
- We need to always use **:** (colon) after the label_name
- Each label_name has to be unique in the scope where it has been defined and cannot be a reserved word, just like in variables

go to statement

Syntax of go to statement in C

2. Transferring the Execution Control

```
goto label_name;
```

- The above statement jumps the execution control of the program to the line where label_name is used.
- Now by combining the above two parts, we get the full syntax of the goto statement in C. However there's a catch.
- We can combine the above two parts in two different ways.

go to statement

Syntax of go to statement in C

Style 1: Transferring the Control From Down to the Top

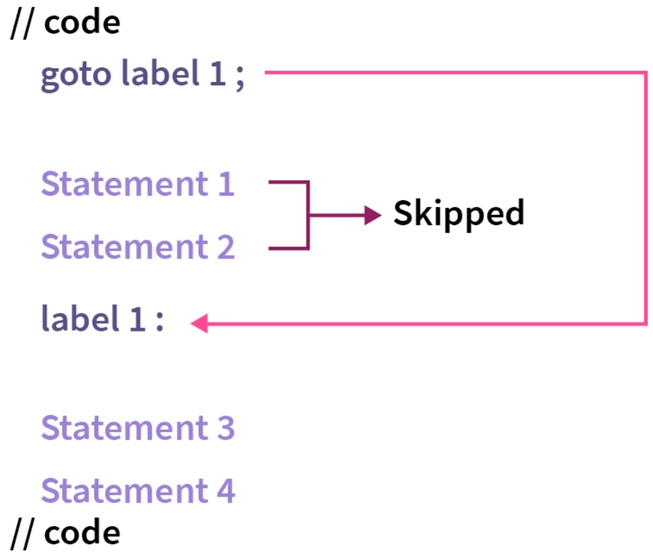
```
label_name:
.
.
.
goto label_name;
```

Style 2: Transferring the control from top to down

```
 goto label_name;
 .
 .
 .
 label_name:
```

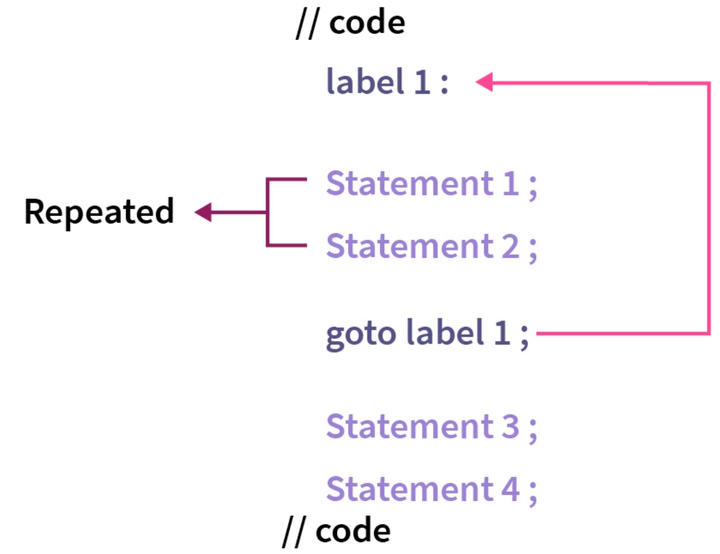
go to statement

Flow Diagram of go to statement in C



Jumped to where label 1 has been declared

- As visible in the flow diagram, as soon as we arrive at the goto statement, the control of the code is transferred to wherever the label has been defined.
- In this case, the label has been defined below the goto statement. Hence, the statements between the goto statement and the label declaration are skipped.
- On the other hand, the label can also be declared before the goto statement. In this case, no statement is skipped, instead, the statements between the goto statement and the label declaration are repeated



Jumped to where label 1 has been declared

go to statement

go to statement in C

- As visible in the flow diagram, as soon as we arrive at the goto statement, the control of the code is transferred to wherever the label has been defined.
- In this case, the label has been defined below the goto statement. Hence, the statements between the goto statement and the label declaration are skipped.
- On the other hand, the label can also be declared before the goto statement. In this case, no statement is skipped, instead, the statements between the goto statement and the label declaration are repeated

go to statement

Styles of Implementing go to statement in C

- There are two different styles of implementing goto statements in C.
  - Either the label is declared above the call of the goto statement, which is the first case,
  - the label is declared after the call of the goto statement.
- In the first style, the flow control shifts from the lower to some upper part of the code, while in the second style the flow control shifts from the upper part to the lower part of the code, maybe skipping some lines in between.

go to statement

Style 1: Transferring the Control From Down to the Top

```
#include <stdio.h>

int main()
{
    statement1;
    ...

    label_name:
        statement2;
        statement3;
        ...
        if(condition)
            goto label_name;

    return 0;
}
```

In the pseudo-code, when and if the condition is true the program execution control will be transferred to label_name.

go to statement

Example 1: To print numbers using the goto statement

This is exactly how a goto statement can create a loop in a program, without using for or while loops. However, in most cases goto statements are used only for flow control, to dictate where the control of the program should be transferred next.

Output

```
1 2 3 4 5 6 7 8 9 10
```

```c
#include <stdio.h>

int main()
{
    // we will print numbers from start to end

    // initialize start and end variables
    int start = 1, end = 10;

    // initialize variable to keep track of which number is to be printed
    int curr = start;

    // defining the label
    print_line :

        // print the current number
        printf("%d ", curr);

        // check if the current has reached the end
        // if not, that means some numbers are still to be printed
        if(curr<end)
        {
            // increment current
            curr++;
            // use goto to again repeat
            goto print_line;
        }

        // if the current has reached the end,
     //the statements inside if will not be executed
        // the program terminates


    return 0;
}
```

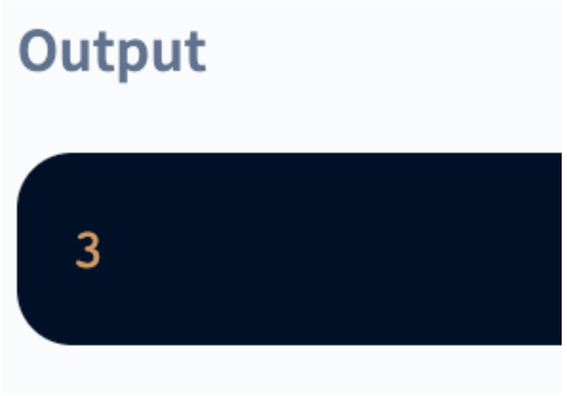
go to statement

Style 2: Transferring the Control From Top to Down

```c
#include <stdio.h>

int main()
{
    statement1;
    ...

    if(condition)
        goto label_name;

    statement2;
    ...

    label_name:

        statement3;
        statement4;
        ...

    return 0;
}
```

go to statement

Example 2: To Find Ceil Division of Two Numbers




```c
#include <stdio.h>

int main()
{

    // we need to find ceil division of a by b

    // initialize a and b
    int a = 5 , b = 2;

    // variable to store division
    int ans = a/b;

    // if a is perefctly divisible by b, just print it
    if(a%b==0)
    {
        // goto statement directs the code to the print_line label
        goto print_line;
    }

    // else 1 needs to be added to the answer, for ceil division
    ans += 1;

    // defined label
    print_line :

        printf("%d", ans);


    return 0;
}
```

go to statement

How Does the goto Statement Work in C?

Goto statement helps in transferring the execution of the program from one line to another. That is why it is a jump statement, as it enables us to jump from one part of our program to another.

This is done by using the goto statement and a label name as defined above in the syntax. Whenever the compiler arrives at a goto statement, it transfers the execution of the program from that line to the line where the label has been defined. Then the execution again starts from that point. Another point to note is that the label can be defined before or after using the goto statement, but it should be present in the code.

As it is clear, we jumped from one part of the code to another, hence goto is an unconditional jump statement.

go to statement

Program to Understand the goto Statement in C

Output

```
The absolute value is 11
```

We skip the 17th line if the number is positive by using the goto statement. When the code enters the if block in line 10 in case of a positive number, it is directed to where the positive label has been declared, which means to the 21st line. Else, the number is multiplied by -1 to get its absolute value and this value is printed.

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

int main()
{
    // initialize variable whose absolute value is to be printed
    int n = -11;

    // if the number is already positive, print it directly
    if(n>=0)
    {
        // goto statement
        goto positive;
    }

    // to make the number positive multiply by -1
    n = n*(-1);

    // declare positive label

    positive :

        printf("The absolute value is %d", n);

    return 0;

}
```

go to statement

Example: Find the Square Root of Positive Numbers in C

Output

```
Negative number entered
```

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

int main()
{
    // initialize variable whose square root is to be found
    int n = -4;

    // declare answer variable
    int ans;

    if(n<0)
    {
        // call goto statement if the number is negative
        goto negative_number;
    }

    // find square root
    ans = sqrt(n);
    printf("The answer is %d", ans);

    // if the answer is calculated, go to the end of the program
    goto end;

    // declare negative_number label
    negative_number:

        printf("Negative number entered");

    // declare end label
    end :
        return 0;

}
```

go to statement

Example: Break out of nested loops using goto statement in C

Output

```
1 2 3 4 5 Complete
```

In the code we have three loops running, if we wanted to break out of all three of them together at once, we would have to write three break statements, one for each loop. But, in this code as we have defined the label after the three nested loops, we can use just one goto statement and break out of all three nested loops.

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

int main()
{
    // initialize variable to store iterations of loops
    int count = 0;

    // start the nested loops
    for(int i=0;i<10;i++){

        // another nested loop
        for(int j=0;j<10;j++){

            //another nested loop
            for(int k=0;k<10;k++){

                count++;
                printf("%d ",count);

                if(count==5)
                {
                    // goto statement
                    goto end;
                }

            }
        }
    }

    // declare end label
    end :

        printf("Complete");

    return 0;

}
```

go to statement

When Should We Use the goto Statement?

Goto statement is used for flow control in programs. As discussed above, it transfers the code execution of code from one part of the program to another.

Hence, in any case, where you need to make a jump from one part of the code to another, you can use the goto statement. It can be used to move to any place within the function where the goto statement is called, depending upon where the label referred to by the goto statement has been declared.

It is important to note that using goto statements we can even jump to a part of the code in our program which we have already passed and executed once if the label has been defined above the goto statement. This was the first style discussed above. This is unlike many other jump statements like a break, which transfer the code execution to a part somewhere below the current execution.

So, use goto statements when you want to jump to another part of the code, which can be above or below your current execution point.

go to statement

Style 1: Transferring the Control From Down to the Top

Advantages of goto Statement in C

Goto is a jump statement that can alter the normal flow of execution of code. Using the goto statement, you can not only jump to a part of the code below the current flow but also a part above the current flow. This also enables goto statements to initiate loops in the program, without using for or while in the code.

We can also use goto statement when a certain condition is satisfied, and skip some lines of code altogether by moving to another part of the program. Additionally, goto statements can be helpful when you want to break out of nested loops. Using one goto statement, you can break out of all loops, instead of using multiple break statements.

Disadvantages of goto Statement in C

Goto statements, although help in jumping from one part of the code to another, make code unreadable. Just imagine having more than two or three goto statements in a program, and trying to figure out which goto statement leads the code to which part. As goto statements alter the normal flow of execution of the program, it can become difficult to understand the new flow when goto statements are added. This makes code unreadable and dirty.

go to statement

Understand this Program

Output

```
15
```

This code looks straight enough and runs fine, but only for odd numbers. If you look closely, this code goes into an infinite loop for even numbers because of line 24. This little cycle is being formed due to the goto statement in this line, and it is difficult to figure it out because of the changing flow of execution of the code due to the goto statements.

This is just a small example of how goto statements can cause unwanted loops and errors in programs if not used with caution.

```c
#include <stdio.h>

int main()
{

    // initialize variable
    int n = 5;

    // initialize answer
    int ans = n;

    // declare first label
    start :

        ans += n;

        if(ans%2==1){
            // first goto statement
            goto odd;
        }

    if(ans%2==0){
        // second goto statement
        goto start;
    }

    // declare second label
    odd :
        printf("%d", ans);


    return 0;
}
```

go to statement

Reasons to avoid go to statements

Goto statements lead to unstructured programs, where it is difficult to predict what might happen next because the flow of execution changes again and again. This kind of code, which has no well-defined structure is called spaghetti code.

This kind of code is not only difficult to read and understand but is also very difficult to update or maintain. This kind of code might be understandable the first time you write it but is extremely difficult to comprehend and update by others. Also, when goto statements are present in the code, it becomes very difficult to figure out how did the flow of execution reach the place of the bug, and where it went next. This not only makes the code unreadable but also leads to difficult debugging.

This is the most important reason why goto statements should be avoided in programs.

Should You Use the goto Statement in C?

Goto statements can be helpful, but only if they are used in the right way. For example, using one goto statement to break out of nested loops can be shorter and more readable than using multiple break statements.

Goto statements can be used, but only when they help in making your program more structured and readable. But, if your code ends up looking unstructured and unreadable like spaghetti code (the kind of code which has no well-defined structure and is difficult to read and understand) because of goto statements, avoid using it.

The most important thing to keep in mind is that everything that can be performed by goto statements, can also be performed by other statements in C, like for loops, while loops, do-while loops, and if-else statements without losing the readability and structure of the code.

go to statement

Summary

- The goto statement in C is used to jump from one block to another block during execution and transfer the flow of execution of the code.
- The syntax of the goto statement can be divided into two parts:
  1. Defining the label.
  2. Transferring the execution control.
- There are two different styles of implementing the goto statement in C:
  1. Transferring the control from down to the top.
  2. Transferring the control from top to down.
- Goto statements can be used in any case where you need to make a jump from one part of the code to another if-else and that it is useful in altering the normal flow of execution of code.
- Although useful, goto statements can make the code unreadable and difficult to maintain and debug, and hence should be used carefully.