

## Break statement(break)

Sometimes it becomes necessary to come out of the loop even before loop condition becomes false then break statement is used. Break statement is used inside loop and switch statements. It cause immediate exit from that loop in whichit appears and it is generally written with condition. It is written with the keywordas **break**. When break statement is encountered loop is terminated and control is transferred to the statement, immediatly after loop or situation where we want to jump out of the loop instantly without waiting to get back to conditional state.

When break is encountered inside any loop, control automatically passes to thefirst statement after the loop. This break statement is usually associated with **if**statement.

Example :

```
void main()
{
int j=0;
for(;j<6;j++)
if(j==4) break;
}
```

Output:

0 1 2 3

## Continue Statement:

Continue statement is used for continuing next iteration of loop after skipping some statement of loop. When it encountered control automatically passes through the beginning of the loop. It is usually associated with the if statement. It is useful when we want to continue the program without executing any part of the program.

The difference between break and continue is, when the break encountered loop is terminated and it transfer to the next statement and when continue is encountered control come back to the beginning position.

In while and do while loop after continue statement control transfer to the test condition and then loop continue where as in, for loop after continue control transferred to the updating expression and condition is tested.

Example:- void

```
main()
{
int n;
for(n=2; n<=9; n++)
{
if(n==4) continue;
printf("%d", n);
}
}
Printf("out of loop");
}
```

Output: 2 3 5 6 7 8 9 out of loop

## SWITCH STATEMENT:

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

## Syntax

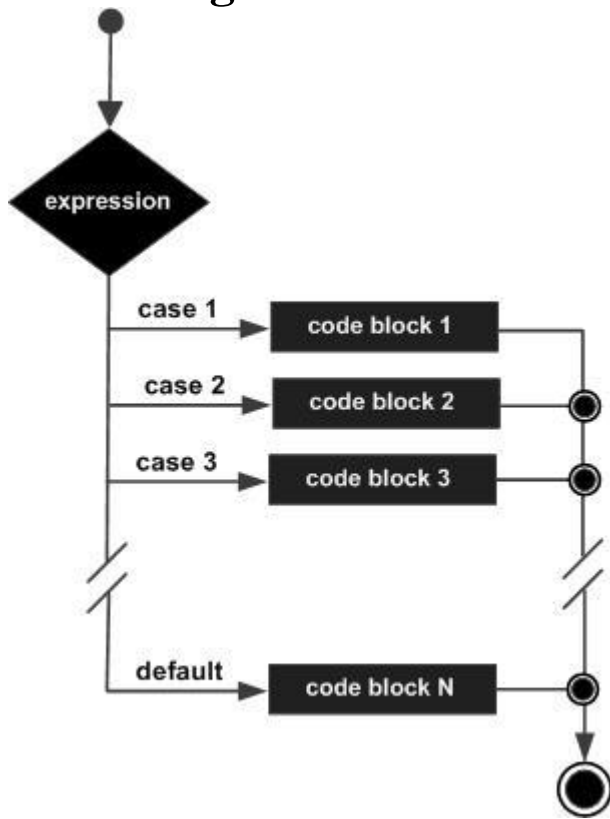
The syntax for a **switch** statement in C programming language is as follows –

```
switch(expression) {  
  
    case constant-expression :  
        statement(s);  
        break; /* optional */  
  
    case constant-expression :  
        statement(s);  
        break; /* optional */  
  
    /* you can have any number of case statements */  
    default : /* Optional */  
        statement(s);  
}
```

The following rules apply to a **switch** statement –

- The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through* to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

# Flow Diagram



## Example

```
#include <stdio.h>

int main () {

    /* local variable definition */
    char grade = 'B';

    switch(grade) {
        case 'A' :
            printf("Excellent!\n" );
            break;
        case 'B' :
        case 'C' :
            printf("Well done\n" );
            break;
        case 'D' :
            printf("You passed\n" );
            break;
        case 'F' :
            printf("Better try again\n" );
            break;
        default :
            printf("Invalid grade\n" );
    }

    printf("Your grade is %c\n", grade );

    return 0;
}
```

```
}
```

When the above code is compiled and executed, it produces the following result –

```
Well done  
Your grade is B
```

## C – else..if statement

The else..if statement is useful when you need to check multiple conditions within the program, nesting of if-else blocks can be avoided using else..if statement.

### Syntax of else..if statement:

```
if (condition1)  
{  
    //These statements would execute if the condition1 is true  
}  
else if(condition2)  
{  
    //These statements would execute if the condition2 is true  
}  
else if (condition3)  
{  
    //These statements would execute if the condition3 is true  
}  
.  
.  
else  
{  
    //These statements would execute if all the conditions return false.  
}
```

### Example of else..if statement

Lets take the same example that we have seen above while discussing nested if..else. We will rewrite the same program using else..if statements.

```
#include <stdio.h>  
int main()  
{  
    int var1, var2;  
    printf("Input the value of var1:");  
    scanf("%d", &var1);  
    printf("Input the value of var2:");  
    scanf("%d",&var2);  
    if (var1 !=var2)  
    {  
        printf("var1 is not equal to var2\n");  
    }  
    else if (var1 > var2)  
    {  
        printf("var1 is greater than var2\n");  
    }  
}
```

```
}
else if (var2 > var1)
{
    printf("var2 is greater than var1\n");
}
else
{
    printf("var1 is equal to var2\n");
}
return 0;
}
```

Output:

```
Input the value of var1:12
Input the value of var2:21
var1 is not equal to var2
```

As you can see that only the statements inside the body of “if” are executed. This is because in this statement as soon as a condition is satisfied, the statements inside that block are executed and rest of the blocks are ignored.

## C goto Statement

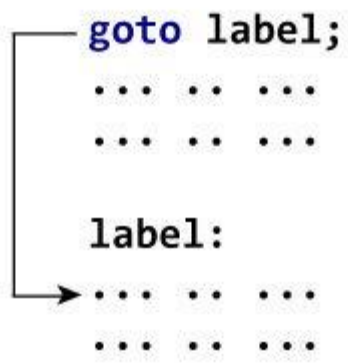
The `goto` statement allows us to transfer control of the program to the specified `label`.

### Syntax of goto Statement

```
goto label;
... ..
... ..
label:
statement;
```

The `label` is an identifier. When the `goto` statement is encountered, the control of the program jumps to `label:` and starts executing the code.

```
goto label;  
... ..  
... ..  
  
label:  
... ..  
... ..
```



Working of goto Statement