APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

STUDY MATERIALS

# MODULE 2

## if statement

```
if (testExpression)
{
  // statements
}
```

The if statement evaluates the test expression inside parenthesis. If test expression is evaluated to true (nonzero) ie, condition satisfied, statements inside the body of if is executed. If test expression is evaluated to false (0) ie, condition not satisfied statements inside the body of if is skipped.

Example #1: C if statement

// Program to display a number if user enters negative number

// If user enters positive number, that number won't be displayed

```
#include <stdio.h>
main()
{
   int number;
   printf("Enter an integer: ");
   scanf("%d", &number);
   // Test expression is true if number is less than 0
   if (number < 0)
   {
      printf("You entered %d.\n", number);
   }
   printf("The if statement is easy.");
}
```

Output

```
Enter an integer: -2
You entered -2.
The if statement is easy.
```

## if...else statement

The if...else statement executes some code if the test expression is true (nonzero) and some other code if the test expression is false (0).

Syntax of if...else

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

If test expression is true, codes inside the body of if statement is executed and, codes inside the body of else statement is skipped.

If test expression is false, codes inside the body of else statement is executed and, codes inside the body of if statement is skipped.

Example #2: C if...else statement

```
// Program to check whether an integer entered by the user is odd or even
#include <stdio.h>
int main()
{
   int number;
   printf("Enter an integer: ");
   scanf("%d",&number);
   // True if remainder is 0
   if( number%2 == 0 )
      printf("%d is an even integer.",number);
   else
      printf("%d is an odd integer.",number);
   return 0;
}
```

Output

```
Enter an integer: 7

7 is an odd integer.
```

**Nested if...else statement (if...elseif....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 nested if...else statement allows you to check for multiple test expressions and execute different codes for more than two conditions.

Syntax of nested if...else statement.

```
if (testExpression1)
{
  // statements to be executed if testExpression1 is true
}
else if(testExpression2)
{
  // statements to be executed if testExpression1 is false and testExpression2 is true
}
else if (testExpression 3)
{
        // statements to be executed if testExpression1 and testExpression2 is false and
        testExpression3 is true
}
.
.
else
{
  // statements to be executed if all test expressions are false
}
```

Example #3: C Nested if...else statement

// Program to relate two integers using =, > or <

```
#include <stdio.h>
int main()
{
   int number1, number2;
   printf("Enter two integers: ");
   scanf("%d %d", &number1, &number2);

   //checks if two integers are equal.
   if(number1 == number2)
   {
      printf("Result: %d = %d",number1,number2);
   }

   //checks if number1 is greater than number2.
   else if (number1 > number2)
   {
      printf("Result: %d > %d", number1, number2);
   }

   // if both test expression is false
   else
   {
```

```
        printf("Result: %d < %d",number1, number2);
    }

    return 0;
}
```

Output

```
Enter two integers: 12
23
Result: 12 < 23
```

**While Loop**

A while loop in C programming repeatedly executes a target statement as long as a given condition is true.

Syntax
The syntax of a while loop in C programming language is −

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

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any nonzero value. The loop iterates while the condition is true.
When the condition becomes false, the program control passes to the line immediately following the loop.

Example

```
#include <stdio.h>
main ()
{
  /* local variable definition */
  int a = 10;

  /* while loop execution */
  while( a < 20 )
{
    printf("value of a: %d\n", a);
    a++;
  }
}
```

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

```
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
```

**do while**

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
The syntax of a do...while loop in C programming language is −

```
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.

Example

```
#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;
}
```

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

```
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
```

**Important**

The **do while** loop executes the content of the loop once before checking the condition of the while.
Whereas a **while** loop will check the condition first before executing the content.

**For loop**

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax
The syntax of a for loop in C programming language is −

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

Here is the flow of control in a 'for' loop −

- The init step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the condition is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
- After the body of the 'for' loop executes, the flow of control jumps back up to the increment statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.

- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

Example

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

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

```
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
```

## Switch

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.

Example

```
#include <stdio.h>

int main () {

  /* local variable definition */
  char grade;
printf(“enter our grade”);
scanf(“%c”, &grade);
  switch(grade)
 {
    case 'A' :
      printf("Excellent!\n" );
      break;
    case 'B' :
       printf("Very Good!\n" );
       break;
    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 −

```
Enter your Grade
B
Very Good
Your grade is B
```

**Break**

The break statement in C programming has the following two usages −

- When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- It can be used to terminate a case in the switch statement (covered in the next chapter).

If you are using nested loops, the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

Syntax

The syntax for a break statement in C is as follows −

```
break;
```

Example

```
#include <stdio.h>
main ()
 {
   /* local variable definition */
   int a = 10;

   /* while loop execution */
   while( a < 20 ) {

      printf("value of a: %d\n", a);
      a++;

      if( a > 15) {
```

```
            /* terminate the loop using break statement */
            break;
         }
      }
   }
```

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

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

## Continue

The continue statement in C programming works somewhat like the break statement. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.
For the for loop, continue statement causes the conditional test and increment portions of the loop to execute. For the while and do...while loops, continue statement causes the program control to pass to the conditional tests.

Syntax
The syntax for a continue statement in C is as follows −

```
continue;
```

## goto and Labels

A goto statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

NOTE − Use of goto statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.

Syntax
The syntax for a goto statement in C is as follows −

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

Here label can be any plain text except C keyword and it can be set anywhere in the C program above or below to goto statement.

Example

```
#include <stdio.h>

main ()
{

  /* local variable definition */
  int a = 10;

  /* do loop execution */
  LOOP:do {

    if( a == 15) {
      /* skip the iteration */
      a = a + 1;
      goto LOOP;
    }

    printf("value of a: %d\n", a);
    a++;

  }while( a < 20 );

}
```

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

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

*try it now*

A KTU STUDENTS PLATFORM

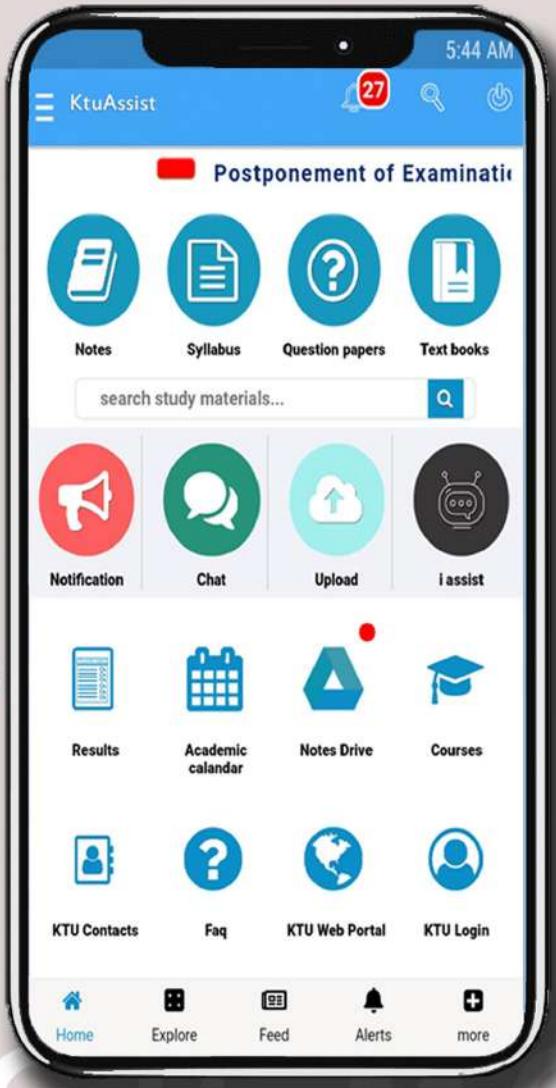
SYLLABUS

NOTES

TEXT BOOKS

QUESTION PAPERS

KTU NOTIFICATION




DOWNLOAD IT FROM GOOGLE PLAY

CHAT

FAQ

LOGIN

CALENDAR

MUCH MORE

DOWNLOAD APP

ktuassist.in

instagram.com/ktu_assist

facebook.com/ktuassist