Computer Fundamentals and Programming in C

Y.Ravikanth,
Lecturer in computer Applications,
Government Degree College, Pattikonda.

DECISION CONTROL STATEMENTS

- Decision control statements are used to alter the flow of a sequence of instructions.
- These statements help to jump from one part of the program to another depending on whether a particular condition is satisfied or not.
- These decision control statements include:

If statement

If else statement

If else if statement

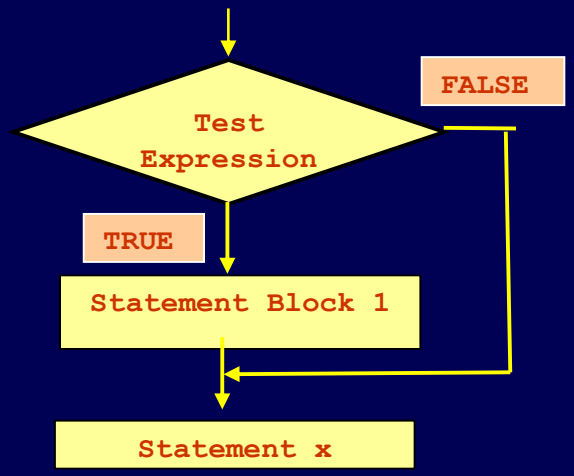
Switch statement

IF STATEMENT

- If statement is the simplest form of decision control statements that is frequently used in decision making. The general form of a simple if statement is shown in the figure.
- First the test expression is evaluated. If the test expression is true, the statement of if block (statement 1 to n) are executed otherwise these statements will be skipped and the execution will jump to statement x.

SYNTAX OF IF STATEMENT

```
if (test expression)
{
        statement 1;
        .............
        statement n;
}
statement x;
```

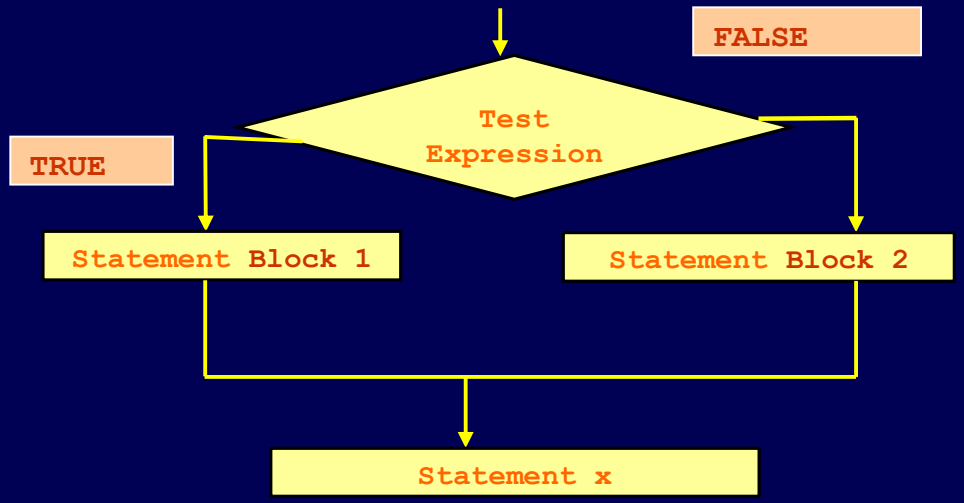


IF ELSE STATEMENT

- In the if-else construct, first the test expression is evaluated. If the expression is true, statement block 1 is executed and statement block 2 is skipped. Otherwise, if the expression is false, statement block 2 is executed and statement block 1 is ignored. In any case after the statement block 1 or 2 gets executed the control will pass to statement x. Therefore, statement x is executed in every case.

SYNTAX OF IF STATEMENT

```
if (test expression)
{
        statement_block 1;
}
else
{
        statement_block 2;
}
statement x;
```



PROGRAMS TO DEMONSTRATE THE USE OF IF AND IF-ELSE STATEMENTS

```
// Program to demonstrate the use of if-statement

#include<stdio.h>
int main()
{
          int x=10;
          if ( x>0)
                    x++;
          printf("\n x = %d", x);
          return 0;
}
```

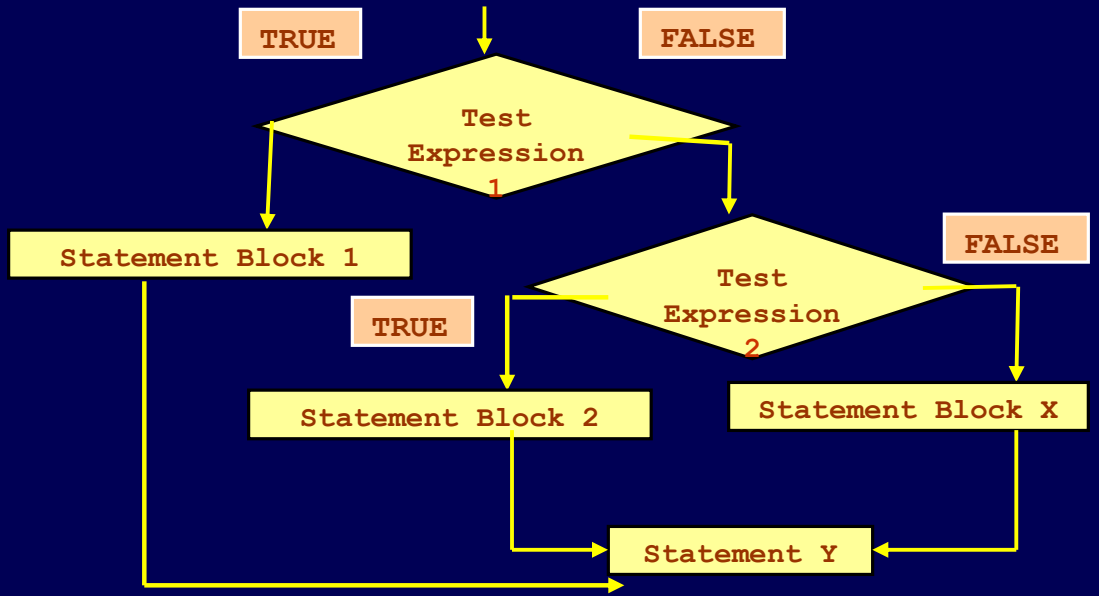
```
// PROGRAM TO FIND WHETHER A NUMBER IS EVEN OR ODD

#include<stdio.h>
main()
{
          int a;
          printf("\n Enter the value of a : ");
          scanf("%d", &a);
          if(a%2==0)
                    printf("\n %d is even", a);
          else
                    printf("\n %d is odd", a);
          return 0;
}
```

IF ELSE IF STATEMENT

- C language supports if else if statements to test additional conditions apart from the initial test expression. The if-else-if construct works in the same way as a normal if statement.

SYNTAX OF IF-ELSE STATEMENT

```
if ( test expression 1)
{
            statement block 1;
}
else if ( test expression 2)
{
            statement block 2;
}
.............................
else if (test expression N)
{
            statement block N;
}
else
{
            Statement Block X;
}
Statement Y;
```



SWITCH CASE

- A switch case statement is a multi-way decision statement. Switch statements are used:

  When there is only one variable to evaluate in the expression

  When many conditions are being tested for

- Switch case statement advantages include:

  Easy to debug, read, understand and maintain

  Execute faster than its equivalent if-else construct

```
switch(grade)
{
        case 'A':
                printf("\n Excellent");
                break;
        case 'B':
                printf("\n Good");
                break;
        case 'C':
                printf("\n Fair");
                break;
        default:
                printf("\n Invalid Grade");
                break;
}
```

```c
// PROGRAM TO CLASSIFY A NUMBER AS POSITIVE, NEGATIVE OR ZERO
#include<stdio.h>
main()
{
        int num;
        printf("\n Enter any number : ");
        scanf("%d", &num);
        if(num==0)
                printf("\n The value is equal to zero");
        else if(num>0)
                printf("\n The number is positive");
        else
                printf("\n The number is negative");
        return 0;
}
```

```c
// PROGRAM TO PRINT THE DAY OF THE WEEK

#include<stdio.h>
int main()
{
        int day;
        printf("\n Enter any number from 1 to 7 : ");
        scanf("%d",&day);
        switch(day)
        {
                case 1: printf("\n SUNDAY"); break;
                case 2 : printf("\n MONDAY"); break;
                case 3 : printf("\n TUESDAY"); break;
                case 4 : printf("\n WEDNESDAY"); break;
                case 5 : printf("\n THURSDAY"); break;
                case 6 : printf("\n FRIDAY"); break;
                case 7 : printf("\n SATURDAY"); break;
                default: printf("\n Wrong Number");
        }
        return 0;
}
```

ITERATIVE STATEMENTS

- Iterative statements are used to repeat the execution of a list of statements, depending on the value of an integer expression. In this section, we will discuss all these statements.

  While loop

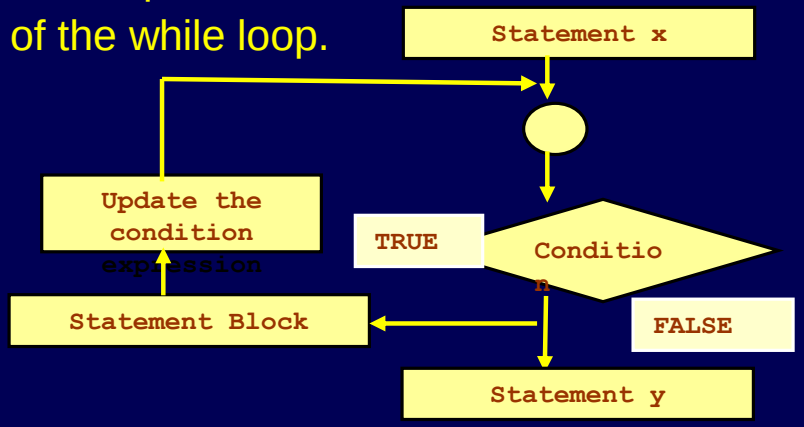
  Do-while loop

  For loop

WHILE LOOP

- The while loop is used to repeat one or more statements while a particular condition is true.
- In the while loop, the condition is tested before any of the statements in the statement block is executed.
- If the condition is true, only then the statements will be executed otherwise the control will jump to the immediate statement outside the while loop block.
- We must constantly update the condition of the while loop.

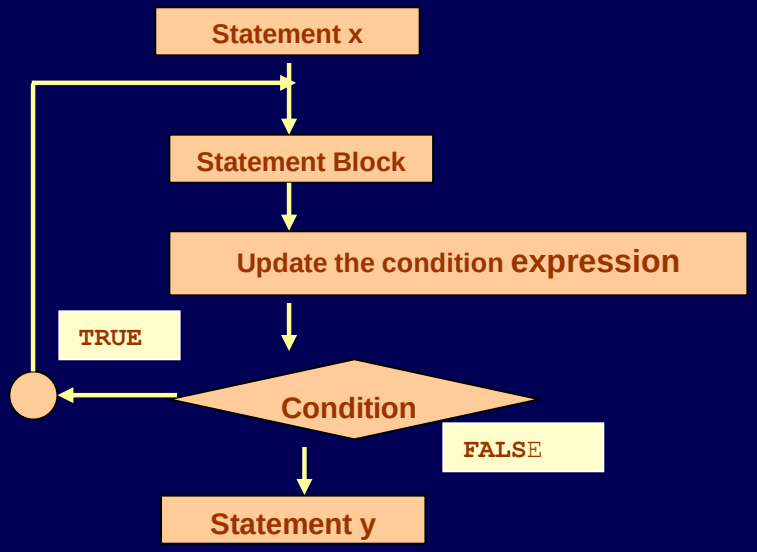
```
while (condition)
{
        statement_block;
}
statement x;
```



DO WHILE LOOP

- The do-while loop is similar to the while loop. The only difference is that in a do-while loop, the test condition is tested at the end of the loop.
- The body of the loop gets executed at least one time (even if the condition is false).
- The do while loop continues to execute whilst a condition is true. There is no choice whether to execute the loop or not. Hence, entry in the loop is automatic there is only a choice to continue it further or not.
- The major disadvantage of using a do while loop is that it always executes at least once, so even if the user enters some invalid data, the loop will execute.
- Do-while loops are widely used to print a list of options for a menu driven program.

```
Statement x;
do
{
                statement_block;
} while (condition);
statement y;
```



Program to demonstrate the use of while loop and do while loop

```
// PROGRAM TO PRINT NUMBERS FROM 0 TO 10 USING WHILE LOOP

#include<stdio.h>
int main()
{
        int i = 0;
        while(i<=10)
        {
                printf("\n %d", i);
                i = i + 1;              // condition updated
        }
        return 0;
}
```

```
// PROGRAM TO PRINT NUMBERS FROM 0-10 USING DO-WHILE LOOP

#include<stdio.h>
int main()
{
        int i = 0;
        do
        {
                printf("\n %d", i);
                i = i + 1;
        } while(i<=10);
        return 0;
}
```

FOR LOOP

- For loop is used to repeat a task until a particular condition is true.
- The syntax of a for loop

```
for (initialization; condition; increment/decrement/update)
{
    statement block;
}
Statement Y;
```

- When a for loop is used, the loop variable is initialized only once.
- With every iteration of the loop, the value of the loop variable is updated and the condition is checked. If the condition is true, the statement block of the loop is executed else, the statements comprising the statement block of the for loop are skipped and the control jumps to the immediate statement following the for loop body.
- Updating the loop variable may include incrementing the loop variable, decrementing the loop variable or setting it to some other value like, i +=2, where i is the loop variable.

PROGRAM FOR FOR-LOOP

Look at the code given below which print first n numbers using a for loop.

```
#include<stdio.h>
int main()
{
        int i, n;
    printf("\n Enter the value of n :");
    scanf("%d", &n);
        for(i=0; i<= n; i++)
                printf("\n %d", i);
        return 0;
}
```

BREAK STATEMENT

- The break statement is used to terminate the execution of the nearest enclosing loop in which it appears.
- When compiler encounters a break statement, the control passes to the statement that follows the loop in which the break statement appears. Its syntax is quite simple, just type keyword break followed with a semi-colon.

  break;
- In switch statement if the break statement is missing then every case from the matched case label to the end of the switch, including the default, is executed.

CONTINUE STATEMENT

- The continue statement can only appear in the body of a loop.
- When the compiler encounters a continue statement then the rest of the statements in the loop are skipped and the control is unconditionally transferred to the loop-continuation portion of the nearest enclosing loop. Its syntax is quite simple, just type keyword continue followed with a semi-colon.

  continue;
- If placed within a for loop, the continue statement causes a branch to the code that updates the loop variable. For example,

```
int i;
for(i=0; i<= 10; i++)
{       if (i==5)
                continue;
        printf("\t %d", i);
```

GOTO STATEMENT

- The **goto** statement is used to transfer control to a specified label.
- Here label is an identifier that specifies the place where the branch is to be made. Label can be any valid variable name that is followed by a colon (:).
- Note that label can be placed anywhere in the program either before or after the goto statement. Whenever the goto statement is encountered the control is immediately transferred to the statements following the label.
- Goto statement breaks the normal sequential execution of the program.
- If the label is placed after the goto statement then it is called a forward jump and in case it is located before the goto statement, it is said to be a backward jump.

```
int num, sum=0;
read:   // label for go to statement
        printf("\n Enter the number. Enter 999 to end : ");
        scanf("%d", &num);
        if (num != 999)
        {
                if(num < 0)
                        goto read; // jump to label- read
                sum += num;
                goto read;              // jump to label- read
        }
        printf("\n Sum of the numbers entered by the user is = %d", sum);
```