BRAINWARE UNIVERSITY

[ESCM201-ESCD201] CLASS NOTES [Programming for Problem Solving]

MODULE 3: Branching and Looping

I. Statements in C, are of following types:

1. Simple statements: Statements that ends with semicolon

2. Compound statements: are also called as block statements that are written in a pair of curly braces.

3. Control Statements: The statements that help us to control the flow of execution in a program.

There are two types of control statements in C

a) **Branching**

b) **Looping**

Branching is deciding what actions to take and looping is deciding how many times to take a certain action.

a. Branching statements: The statements that help us to jump from one statement to another statement with or without condition is called branching statements. These statements are also called ***as selection statements*** since they select some statements for execution and skip other statements. These statements are also called as **decision-making statements** because they make decision based on some condition and select some statements for execution.

Branching Statements are of two types:

i) Conditional Branching statements: The statements that help us to jump from one statement to another statement based on condition.

Example:

- simple if statement,
- if…else statement,
- nested if…else statement,
- else…if ladder (cascaded if statement), and
- switch statement

ii) Unconditional Branching statements: The statements that help us to jump from one statement to another statement without any conditions. Example:

- goto statement,
- break statement,
- continue statement and
- return statement

Conditional Branching Statements : The statements that help us to jump from one statement to another statement based on condition.

a. Simple if Statement

This is basically a "one-way" decision statement. This is used when we have only one alternative. It takes n expression in parenthesis and a statement or block of statements. If the expression is TRUE then the statement or block of statements gets executed otherwise these statements are skipped.
NOTE: Expression will be assumed to be TRUE if its evaluated value is non-zero.

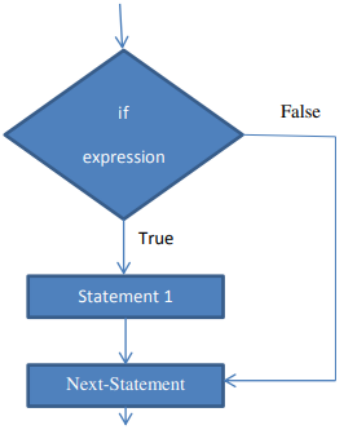
The syntax is shown below:

```
if (condition)
{
statement1;
}
Next-Statement;
```

Where,

If the condition is evaluated to TRUE, then statement1 is executed and jumps to Next-Statement for execution. If the condition is evaluated to FALSE, then statement1 is skipped and jumps to Next-Statement for execution.

The flow diagram (Flowchart) of simple if statement is shown below:



Example: Program to illustrate the use of if statement.

/* Program to find a given number is positive */

```
#include<stdio.h>
int main()
{
int n;
printf("Enter any non-zero integer: \n");
scanf("%d", &n);
if(n>0)
printf("Number is positive number ");
 return 0;
}
```

Output: Enter any non-zero integer: 8

Number is positive number

THE if…else STATEMENT

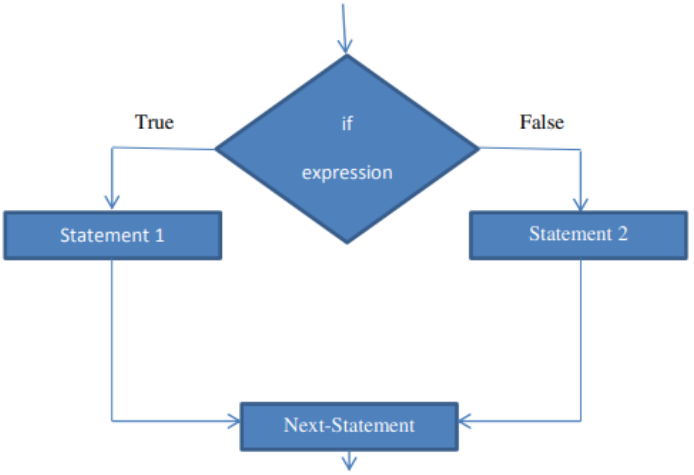
This is basically a "two-way" decision statement. This is used when we must choose between two alternatives.

The syntax is shown below:

```
if(condition)
{
statement1;
}
else
{
statement2;
}
Next-Statement;
```

Firstly, the condition is evaluated to true or false. If the condition is evaluated to TRUE, then statement1 is executed and jumps to Next-Statement (Skip statement2) for execution. If the expression is evaluated to FALSE, then statement2 is executed (Skip
statement1) and jumps to Next-Statement for execution.

The flow diagram is shown below:



Example: Program to illustrate the use of if else statement.

```
#include<stdio.h>
int main()
{
int n;
printf("Enter any non-zero integer: \n") ;
scanf("%d", &n)
if(n>0)
printf("Number is positive number");
else
printf("Number is negative number");
 return 0;
}
```

Output 1:
Enter any non-zero integer:
8
Number is positive number
Output 2:
Enter any non-zero integer:
-9
Number is negative number

THE nested if STATEMENT

An if-else statement is written within another if-else statement is called nested if statement. This is used when an action has to be performed based on many decisions. Hence, it is called as multi-way decision statement.

The syntax is shown below:

```
if(condition1)
{
        if(condition2)
                statement1;
        else
                statement2;
}
else
{
        if(condition3)
                statement3
        else
                statement4
}
Next-Statement;
```

• Here, firstly condition1 is evaluated to true or false.

If the **condition1** is evaluated to **TRUE**,
then
condition 2 is evaluated to true or false.
If the **condition** 2 is evaluated to **TRUE**,
then
statement1 is executed.
If the **condition2** is evaluated to **FALSE**,
then
statement2 is executed.

If the **condition** 1 is evaluated to FALSE,
then

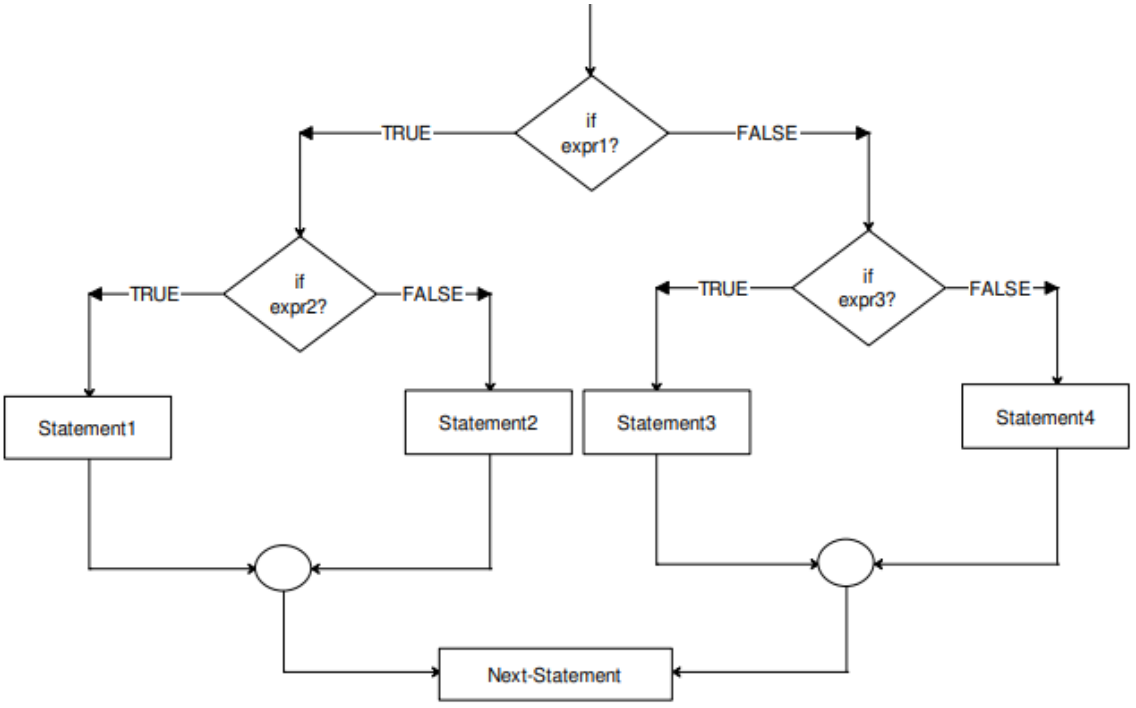
condition3 is evaluated to true or false.
If the **condition3** is evaluated to TRUE,
then

statement3 is executed.

If the **condition3** is evaluated to FALSE,
then

statement4 is executed.

• The flow diagram is shown below:



Example: Program to select and print the largest of the 3 numbers using nested "if-else" statements.

```
#include<stdio.h>
int main()
{
int a,b,c;
printf("Enter Three Values: \n");
scanf("%d %d %d ", &a, &b, &c);
printf("Largest Value is: ") ;
if(a>b)
{
```

```
        if(a>c)
        printf(" %d ", a);
else
        printf(" %d ", c);
}
else
{
        if(b>c)
        printf(" %d", b);
        else
        printf(" %d", c);
}
return 0;
}
```

Output:
Enter Three Values:
17 18 6
Largest Value is: 18

THE CASCADED if-else STATEMENT (THE else if LADDER STATEMENT)

This is basically a "multi-way" decision statement. This is used when we must choose among many alternatives. This statement is alternative for nested if statement to overcome the complexity problem involved in nested. if statement.

This statement is easier to understand and trace the flow of execution.

The syntax is shown below:

```
if(condition1)
{
        statement1;
}
else if(condition2)
{
        statement2;
}
else if(condition3)
{
        statement3
}
.
.
.
else if(conditionN)
```

```
{
 statementN;
else
{
default statement;
}
Next-Statement;
```

The **conditions** are evaluated in order (i.e. top to bottom).
If an expression is evaluated to TRUE, then

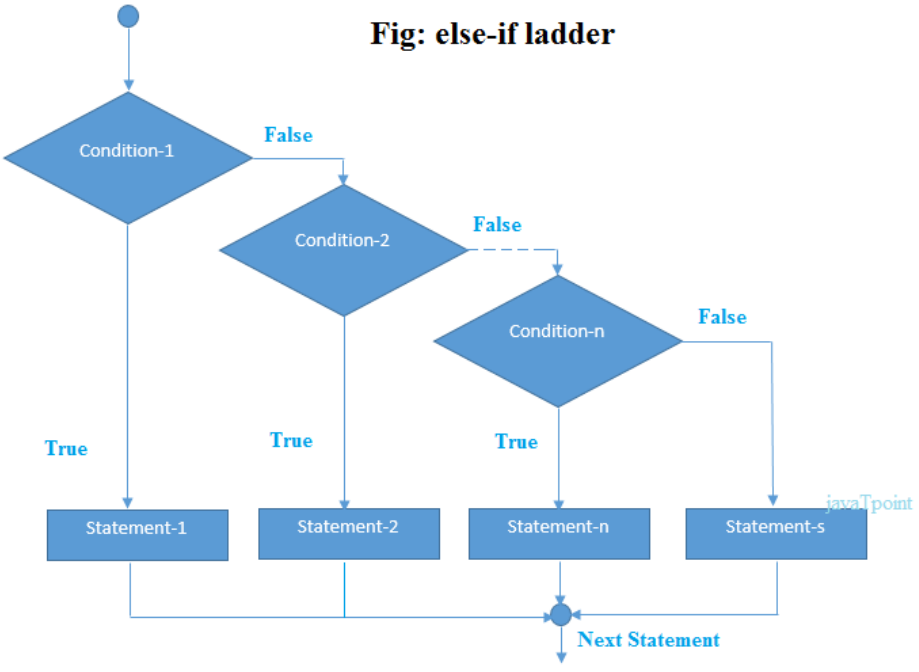
→ Statement associated with the expression is executed &

→ Control comes out of the entire else if ladder

For example,
if **condition1** is evaluated to TRUE,
then **statement1** is executed and jumps out of entire else...if ladder.
if **condition2** is evaluated to TRUE,
then **statement2** is executed and jumps out of entire else...if ladder.
If **all the condition** are evaluated to FALSE,
then last statement (**default statement**) is executed.

Flowchart is shown below:

Fig: else-if ladder



switch Statement:

This is a multi-branch statement similar to the else…if ladder (with limitations) but clearer and easier to code. This is used when we must choose among many alternatives. This statement is to be used only when we have to make decision using integral values (i.e., either integer values or characters).

Why we should use Switch statement?

1. One of the classic problems encountered in nested if-else / else-if ladder is called problem of Confusion.
2. It occurs when no matching else is available for if.
3. As the number of alternatives increases the Complexity of program increases drastically.

To overcome these, C Provides a multi-way decision statement called "Switch Statement"

Syntax:

```
switch ( expression )
{
 case 1 :
statement1 ;
break ;
 case 2 :
statement2 ;
break ;
 ...
 default : statement ;
}
```

Where,

Expression can be either arithmetic expression that reduces to integer value, character constant, integer constant or an identifier of type integer.

case can be either arithmetic expression that reduces to integer value, character constant or integer constant. But it cannot be an identifier. The value of expression is tested for equality against the values of each of the labels specified in the case statements in the order written until a match is found. The statements associated with that case statement are then executed until a break statement or the end of the switch statement is encountered.

When a break statement is encountered execution jumps to the statement immediately following the switch statement.

The default section is optional -- if it is not included the default is that nothing happens and execution simply falls through the end of the switch statement.

The flow of switch statement is shown below:

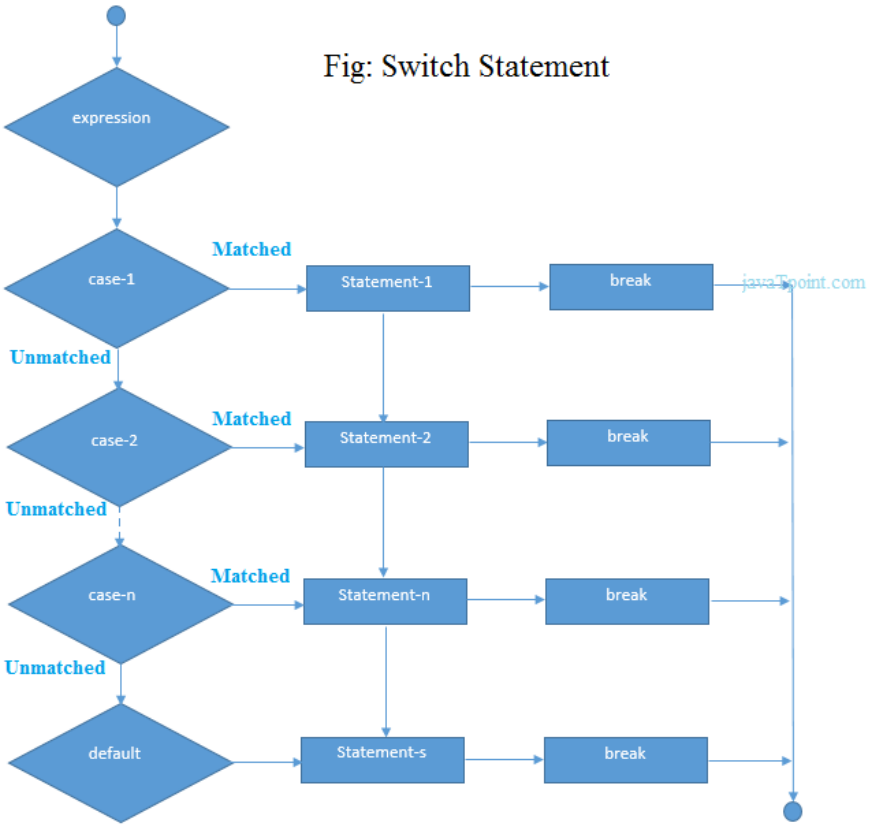


Fig: Switch Statement

For Example :- Program to simulate a basic calculator.

```
#include <stdio.h>
int main()
{
int num1, num2, result ;
char op ;
 printf ( " Enter number operator number\n" ) ;
scanf ("%f %c %f", &num1, &op, &num2 ) ;
switch ( op )
{
case '+' : result = num1 + num2 ;
           break ;
case '-' : result = num1 - num2 ;
           break ;
case „*‟: result = num1 * num2 ;
           break ;
```

```
 case „%‟: result=num1%num2;
          break;
case „/‟ : if ( num2 != 0.0 )
 {
          result = num1 / num2 ;
          break ;
 }
 // else we allow to fall through for error message
default : printf ("ERROR -- Invalid operation or division by 0.0" ) ;
}
printf( "%f %c %f = %f\n", num1, op, num2, result) ;
return 0;
}
```

Looping statements:

The statements that help us to execute set of statements repeatedly are called **as looping statements.**

Initialization: Before a loop can start, some preparation is usually required. We call this preparation as initialization. Initialization must be done before the execution of the loop body. Initialization sets the variable which will be used in control expression to control the loop.

Updating: The control expression that controls the loop must be updated inside the loop. Hence, the actions that cause these changes are known as update.

There are 3 types of loops supported in C.

1. while loop
2. do while loop
3. for loop

while loop: - The while statement is typically used in situations where it is not known in advance how many iterations are required.

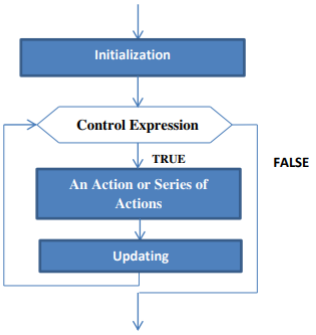
Syntax:

```
Initialization
while (control expression)
{
          //body of loop
          //update
```

}

Here, first the control variable is initialized then control expression is evaluated to TRUE or FALSE. If it is evaluated to TRUE, body of loop is executed then variable is updated. The process of executing the body of loop is repeated until the control expression is FALSE. Once control expression is evaluated to FALSE, then the control is transferred out of the loop. The execution resumes from the statement following the loop.

Flow chart of while:



Example: Program to sum all integers from 1 to 10.

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

for statement

The for statement is most often used in situations where the programmer knows in advance how many times a particular set of statements are to be repeated. The for statement is sometimes termed as counter controlled loop.

Syntax:

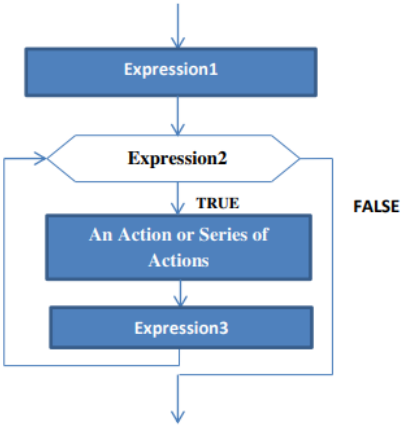
```
for ( expression1 ; expression2 ; expression3 )
{
 // body of loop
}
```

Where,

expression1:- This is usually an assignment/initialization statement to set a loop control variable(s) for example. But it is not restricted to only initialization, it can be any valid C statement.

expression2:- This is usually a control expression which determines when loop will terminate. But, this can be any statement of C that evaluates to TRUE or FALSE.
expression3:- This usually defines how the loop control variable(s) will change each time the loop is executed i.e. updating.
Body of loop:- Can be a single statement, no statement or a block of statements.
NOTE: All three expressions are optional. But semicolons must be present as in syntax in the loop.

Flow chart of for:



Example 2:- To print out all numbers from 1 to 10.

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

Multiple Initialisations in for loop

C has a special operator called **the comma operator** which allows separate expressions to be tied together into one statement.

For example it is possible to initialize two variables in a for loop as follows:-

```
for ( x = 0, sum = 0; x <= 100; x++ )
{
printf( "%d\n", x) ;
sum += x ;
}
```

Any of the four sections associated with a for loop may be omitted but the semi-colons must be present always.

For Example 1:- expression 3 is omitted

```
for ( x = 0; x < 10; )
 printf( "%d\n", x++ ) ;
```

For Example 2:- expression 1 is omitted in for loop and it is before for loop

```
x = 0 ;
for ( ; x < 10; x++ )
printf( "%d\n", x ) ;
```

For Example 3:- An infinite loop may be created as follows

```
 for ( ; ; )
 body of loop;
```

For Example 5:- expression 3 is a printf statement. This example prints from 1 to 100.

```
 for ( x = 1; x <= 100; printf( "%d\n", x++ ) ) ;
```

do…while statement

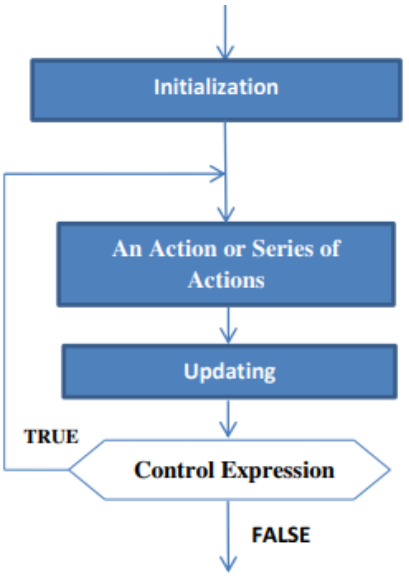
The terminating condition in the for and while loops is always tested before the body of the loop is executed -- so of course the body of the loop may not be executed at all.

In the do…while statement on the other hand the body of the loop is always executed at least once as the condition is tested at the end of the body of the loop.

Syntax:

```
Initialization
do
{
        // body of loop
        Updating
} while ( control expression ) ;
```

Flow chart of do while loop



For Example : To sum all numbers from 1 to 10.

```
void main()
{
 int i, sum=0;
i = 1;
do
{
sum = sum + i ;
i++;
} while ( i <= 100 ) ;}
```

Nested loops:-

It is possible to build a nested structure of for loops.

In this example, the outer for loop will iterate for "m" times and inner loop will iterate for "n" times, hence the total iterations from nested loop is m*n times.

For Example: Program to produce the following table of values

```
 1 2 3 4 5
 2 3 4 5 6
 3 4 5 6 7
 4 5 6 7 8
 5 6 7 8 9
 #include <stdio.h>
int main()
{
int j, k ;
 for ( j = 1; j <= 5; j++ )
{
for ( k = j ; k < j + 5; k++ )
```

```
{
printf( "%d ", k ) ;
}
printf( “\n” ) ;
}
return 0;
}
```

Programming examples on Looping:

1. WACP to find the sum of numbers from 1 to N where N is entered from keyboard.

Program:

```
// sum of numbers from 1 to N i.e. sum = 1+2+3+4+5+…+N
#include<stdio.h>
int main()
{
 int N, sum=0, i;
printf(“Enter the value of N\n”);
scanf(“%d” , &N);
 i=1;
 while(i<=N) {
 sum = sum + i;
 i++;  }
printf(“Sum = %d\n” , sum);
 return 0; }
```

2. WACP to find the sum squares of numbers from 1 to N where N is entered from keyboard.

Program:

```
#include<stdio.h>
 #include<math.h>
 int main()
 {
 int N, sum=0, i;
 printf(“Enter the value of N\n”);
scanf(“%d” , &N);
 i=1;
 while(i<=N)
 {
 sum = sum + pow(i,2);
 i++;
 }
printf(“Sum = %d\n” , sum);
 return 0;
}
```

WACP to evaluate the following f(x) = 1 + X + X^2 + X^3+ … + X^N , where X and N are entered from keyboard.

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

4. WACP to find and print reverse number of a given integer number.

Program:

```
#include<stdio.h>
int main()
{
int N, reverse=0,rem,temp;
printf("Enter the value of N\n");
scanf("%d" , &N);
temp=N;
while(N>0)
{
rem = N%10;
reverse = reverse * 10 + rem;
N = N/10;
}
printf("Given Number = %d\n", temp);
printf("Reversed Number = %d\n", reverse);
return 0;
}
```