

Module 3

Conditional Branching and Loops

Normal flow of execution in a C program is sequential, i.e. the statements in a C program runs in a linear fashion.

Consider the following case for a c program

Statement_1: int a=1,b=2,c;

Statement_2: c=a+b;

.
.
.

Statement_n: printf("%d",c);

In above scenario first statement_1 will be executed then statement_2 and so on until the last statement statement_n got executed.

If we want to change this order of execution for some programming convenience like skipping some set of statements or repeating some set of statements then we use conditional and iteration in our program.

3.1 Writing and evaluation of Conditionals and Consequent Branching:

If we want that some set of statement only get executed when certain conditions are met then we use conditionals.

IF Statement

Consider the following case for a c program

Statement_1: int a=1,b=2,c;

Statement_K-1: printf("addition of two numbers");

if(Condition)

{

Statement_k: a=1,b=2

Statement_K+1: c=a+b;

.

.

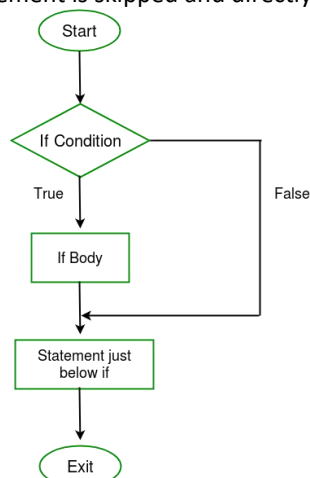
.

Statement_n: printf("%d",c);

}

Statement_n+1: printf("end of program");

In above scenario after the execution of statement_1 to Statement_k-1 the condition given in if statement is checked. If the Condition is true then the statements in the if block i.e. from statement_k to statement_n get executed and then statement_n+1 get executed. If the condition is false then the whole block corresponding to if statement is skipped and directly statement_n+1 get executed.



Flow chart of if statement

Example: if statement

```
#include <stdio.h>

int main() {
    int i = 10;

    if (i > 15)
    {
        printf("10 is less than 15");
    }

    printf("I am Not in if");
}
```

Note: Consider the following scenario.

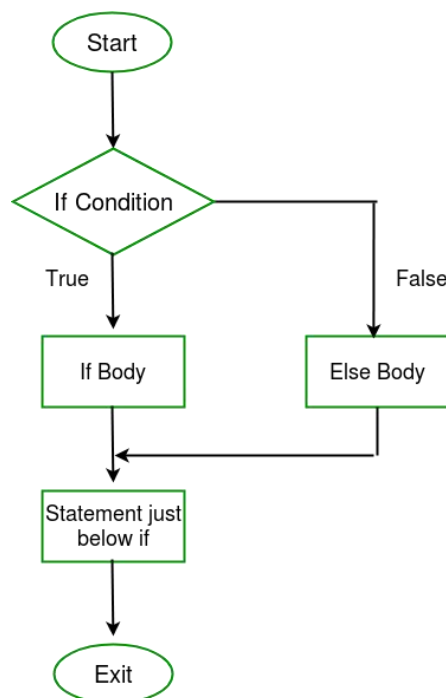
```
if(condition)
    statement1;
    statement2;
```

Here if the condition is true, if block will consider only statement1 to be inside its block. We must use pair of braces { } for multiple statements in an if block.

if- else statement:

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false. Here comes the else statement. We can use the else statement with if statement to execute a block of code when the condition is false.

```
if (condition)
{
    // Executes this block if condition is true
}
else
{
    // Executes this block if condition is false
}
```



Flow chart of if statement

Example: if-else statement

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

```
int main() {  
    int i = 20;  
  
    if (i%2==0)  
        printf("number is even");  
    else  
        printf("number is odd");  
  
    return 0;  
}
```

nested-if

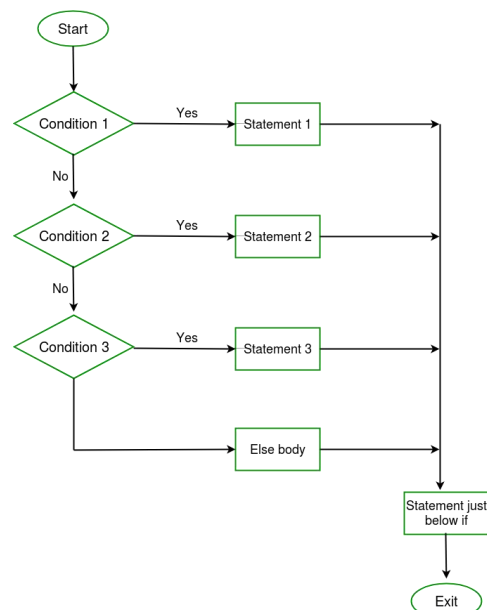
A nested if is an if statement that is the target of another if statement. Nested if statements means an if statement inside another if statement.

```
if (condition1)  
{  
    // Executes when condition1 is true  
    if (condition2)  
    {  
        // Executes when condition2 is true  
    }  
}
```

if-else-if ladder or else-if

Here, a user can decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

```
if (condition)  
    statement;  
else if (condition)  
    statement;  
.  
.  
else  
    statement;
```



Flow chart of if-else-if ladder

Example: if-els-if ladder

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

```
int main() {
```

```
    int i = 20;
```

```
    if (i == 10)
```

```
        printf("i is 10");
```

```
    else if (i == 15)
```

```
        printf("i is 15");
```

```
    else if (i == 20)
```

```
        printf("i is 20");
```

```
    else
```

```
        printf("i is not present");
```

```
}
```

Switch statement:

The control statement that allows us to make a decision from the number of choices is called a switch, or more correctly a switchcase-default, since these three keywords go together to make up the control statement. They most often appear as follows:

```
switch ( integer expression )
```

```
{
```

```
case constant 1 :
```

```
    do this ;
```

```
    break;
```

```
case constant 2 :
```

```
    do this ;
```

```
    break;
```

```
case constant 3 :
```

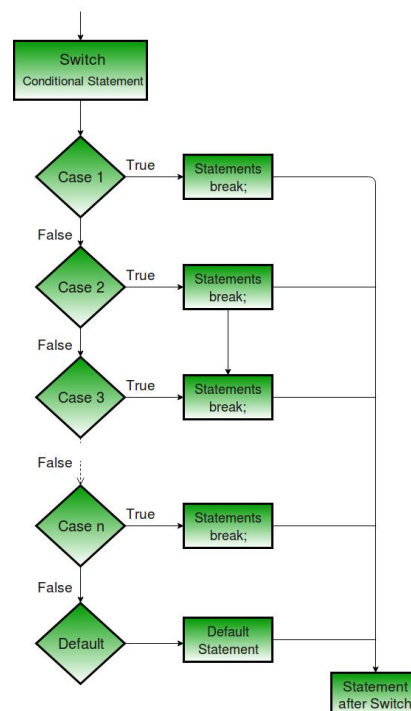
```
    do this ;
```

```
    break;
```

```
default :
```

```
    do this ;
```

```
}
```



Note: if we do not use the break statement then all statements after that case statement for which condition is satisfied get executed. If we use the break statement and no condition is satisfied then statement in default case get executed.

```
main( )
{
    int i = 2 ;
    switch ( i )
    {
        case 1 :
            printf ( "I am in case 1 \n" ) ;
        case 2 :
            printf ( "I am in case 2 \n" ) ;
        case 3 :
            printf ( "I am in case 3 \n" ) ;
        default :
            printf ( "I am in default \n" ) ;
    }
}
```

The output of this program would be:

```
I am in case 2
I am in case 3
I am in default
```

3.2 Iteration and Loops

Loops in programming comes into use when we need to repeatedly execute a block of statements. For example: Suppose we want to print "Hello World" 10 times. This can be done in two ways as shown below:

1. Iterative Method

Iterative method to do this is to write the printf() statement 10 times.

// C program to illustrate need of loops

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

```
int main()
{
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
    printf( "Hello World\n");
}
```

```
    return 0;
```

```
}
```

2. Loops

In Loop, the statement needs to be written only once and the loop will be executed 10 times as shown below.

In computer programming, a loop is a sequence of instructions that is repeated until a certain condition is reached.

An operation is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number.

Counter not Reached: If the counter has not reached the desired number, the next instruction in the sequence returns to the first instruction in the sequence and repeat it.

Counter reached: If the condition has been reached, the next instruction "falls through" to the next sequential instruction or branches outside the loop.

There are mainly two types of loops:

Entry Controlled loops: In this type of loops the test condition is tested before entering the loop body. For Loop and While Loop are entry controlled loops.

Exit Controlled Loops: In this type of loops the test condition is tested or evaluated at the end of loop body. Therefore, the loop body will execute atleast once, irrespective of whether the test condition is true or false. do – while loop is exit controlled loop.

Loop statements are used to execute the statements repeatedly as long as an expression is true. When the expression becomes false then the control transferred out of the loop. There are three

kinds of loops in C.

a) while b) do-while c) for

a. while statement

while loop will be executed as long as the exp is true.

Syntax:

```
while (exp)
{
    statements;
}
```

The statements will be executed repeatedly as long as the exp is true. If the exp is false then the control is transferred out of the while loop.

Example:

```
int digit = 1;
While (digit <=5) FALSE
{
    printf ("%d", digit); TRUE Cond Exp
    Statements; ++digit;
}
```

The while loop is top tested i.e., it evaluates the condition before executing statements in the body. Then it is called entry control loop.

b. do-while statement

The do-while loop evaluates the condition after the execution of the statements in the body.

Syntax: do Statement; While<exp>;

Here also the statements will be executed as long as the exp value is true. If the expression is false the control come out of the loop.

Example:

```
-int d=1; do
{
    printf ("%d", d); FALSE
    ++d;
} while (d<=5); TRUE Cond Exp
statements exit
```

The statement with in the do-while loop will be executed at least once. So the do-while loop is called a bottom tested loop.

c.for statement

The for loop is used to executing the structure number of times. The for loop includes three expressions. First expression specifies an initial value for an index (initial value), second expression that determines whether or not the loop is continued (conditional statement) and a third expression used to modify the index (increment or decrement) of each pass.

Note: Generally for loop used when the number of passes is known in advance.

Syntax:

```
for (exp1;exp2;exp3)
{
```

```
}
```

Where **expression-1** is used to initialize the control variable. This expression is executed this expression is executed is only once at the time of beginning of loop.

Where **expression-2** is a logical expression. If expression-2 is true, the statements will be executed, other wise the loop will be terminated. This expression is evaluated before every execution of the statement.

Where **expression-3** is an increment or decrement expression after executing the statements, the control is transferred back to the expression-3 and updated.

There are different formats available in for loop. Some of the expression of loop can be omit.

Format - I

for(; exp2; exp3) Statements;

In this format the initialization expression (i.e., exp1) is omitted. The initial value of the variable can be assigned outside of the for loop.

Example 1

```
int i = 1;
for( ; i<=10; i++ )
printf ("%d \n", i);
```

Format - II

for(; exp2 ;) Statements;

In this format the initialization and increment or decrement expression (i.e expression-1 and expression-3) are omitted. The exp-3 can be given at the statement part.

Example 2

```
int i = 1;
for( ; i<=10; )
{
printf ("%d \n",i); i++;
}
```

Format - III

for(; ;) Statements;

In this format the three expressions are omitted. The loop itself assumes the expression-2 is true. So Statements will be executed infinitely.

Example 3

```
int i = 1;
for ( ; i<=10; )
{
printf ("%d \n",i); i++;
}
```

Nested Looping Statements

Many applications require nesting of the loop statements, allowing one loop statement to be embedded within another loop statement.

Definition

Nesting can be defined as the method of embedding one control structure within another control structure.

While making control structures to reside one within another, the inner and outer control structures may be of the same type or may not be of the same type. But, it is essential for us to ensure that one control structure is completely embedded within another.

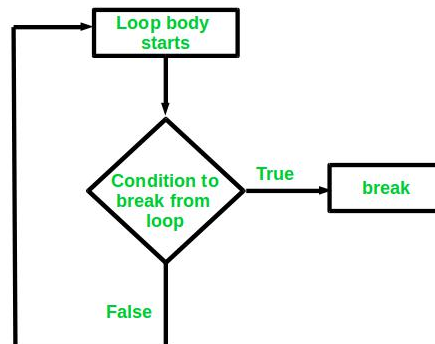
/*program to implement nesting*/ #include <stdio.h>

```
main()
{
int a,b,c,
for (a=1,a<=2, a++)
{
printf ("%d",a)
for (b=1,b<=2,b++)
{
print f("%d",b)
for (c=1,c<=2,c++)
{
print f( " My Name is Sunny \n");
}
}
}
}
```


Jump Statement:

1. **break:** This loop control statement is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stops there and control returns from the loop immediately to the first statement after the loop.

Syntax: break;

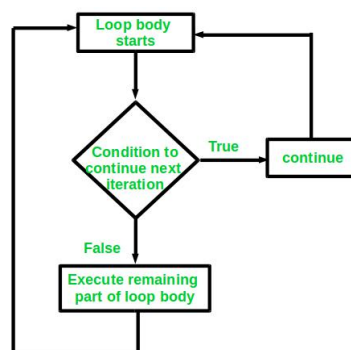
**Example**

```
for ( ; ; ) //normal loop
{
    //break Condition within loop
    scanf ("%d",&n);
    if ( n < -1)
        break;
    sum = sum + n;
}
```

2. **continue:** This loop control statement is just like the break statement. continue statement is opposite to that of break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.

As the name suggest the continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and next iteration of the loop will begin.

Syntax: continue;

**Example**

```
while (x<=100)
{
    if (x <= 0)
    {
        printf ("zero or negative value found \n");
        continue;
    }
}
```

The above program segment will process only the positive whenever a zero or negative value is encountered, the message will be displayed and it continue the same loop as long as the given condition is satisfied.

3. **goto:** The goto statement is a jump statement which is sometimes also referred to as unconditional jump statement. The goto statement can be used to jump from anywhere to anywhere within a function.

Syntax:

goto label;

.

.

label:

Note: label can come before or after the goto statement

Example

```
#include <stdio.h> main();
{
    int a,b;
    printf ("Enter the two numbers");
    scanf ("%d %d",&a,&b);
    if (a>b)
        goto big;
    else goto small;
    big :printf ("big value is %d",a);
    goto stop;
    small :printf ("small value is %d",b);
    goto stop; stop;
}
```

4. **return:** This statement returns the flow of the execution to the function from where it is called. This statement does not mandatorily need any conditional statements. As soon as the statement is executed, the flow of the program stops immediately and return the control from where it was called. The return statement may or may not return anything for a void function, but for a non-void function, a return value is must be returned.

Syntax: return[expression];