Rajmani Kumar,
Lecturer, Dept. of BCA
S.U.College, Hilsa (Nalanda)
Patliputra University, Patna

BCA-2nd Year **Paper-III**

Control Structure

In C, programs are executed sequentially in the order of which they appear. This condition does not hold true always. Sometimes a situation may arise where we need to execute a certain part of the program. Also it may happen that we may want to execute the same part more than once. Control statements enable us to specify the order in which the various instructions in the program are to be executed. They define how the control is transferred to other parts of the program. Control statements are classified in the following ways:

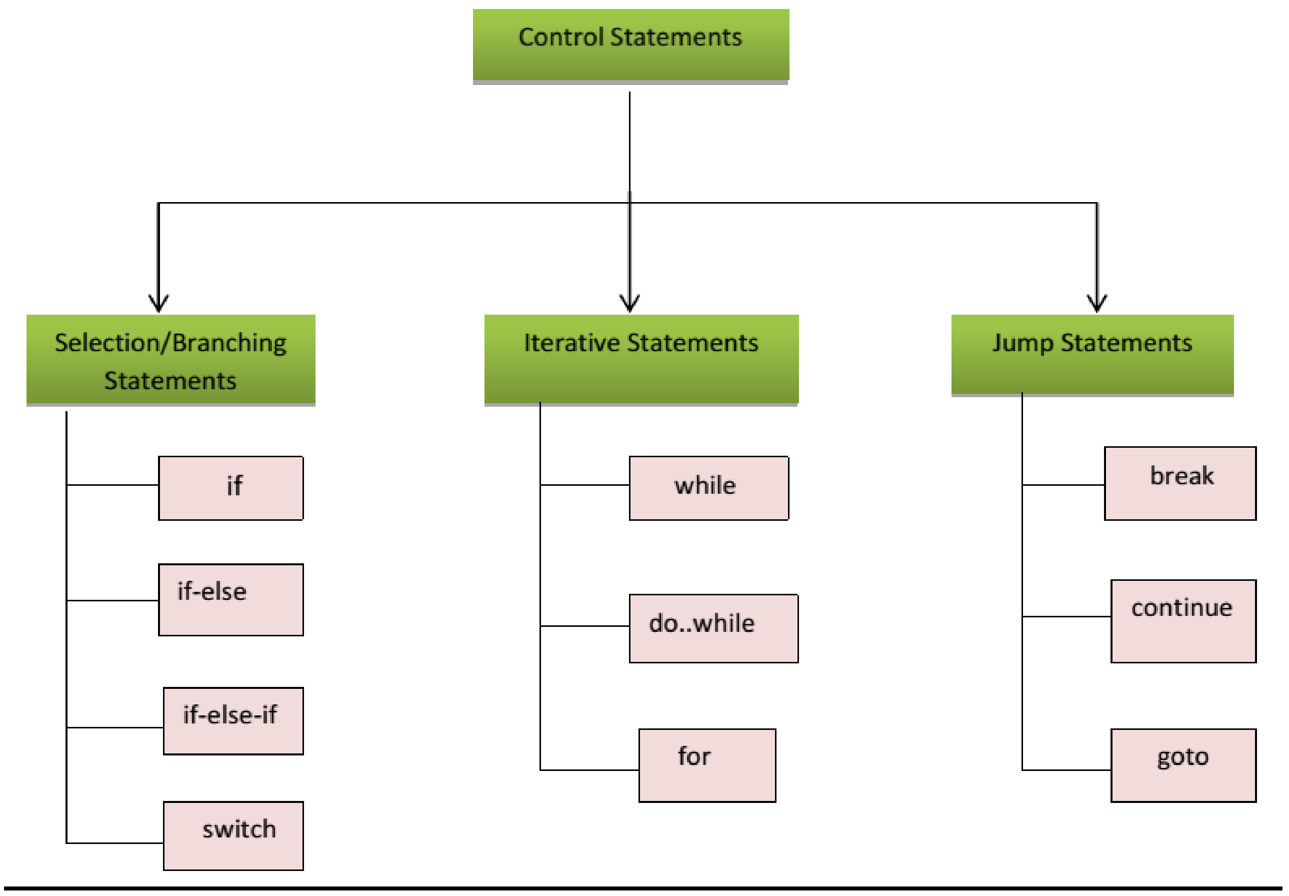

Fig: 1 Classification of control statements

SELECTION STATEMENTS

The selection statements are also known as *Branching* or *Decision Control Statements.*

Introduction to Decision Control Statements

Sometime we come across situations where we have to make a decision. E.g. *If the weather is sunny, I will go out & play, else I will be at home.* Here my course of action is governed by the kind of weather. If it's sunny, I can go out & play, else I have to stay indoors. I choose an option out of 2 alternate options. Likewise, we can find ourselves in situations where we have to select among several alternatives. We have decision control statements to implement this logic in computer programming.

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

if Statement

The keyword *if* tells the compiler that what follows is a decision control instruction. The *if* statement allows us to put some decision -making into our programs. The general form of the *if* statement is shown Fig 2:

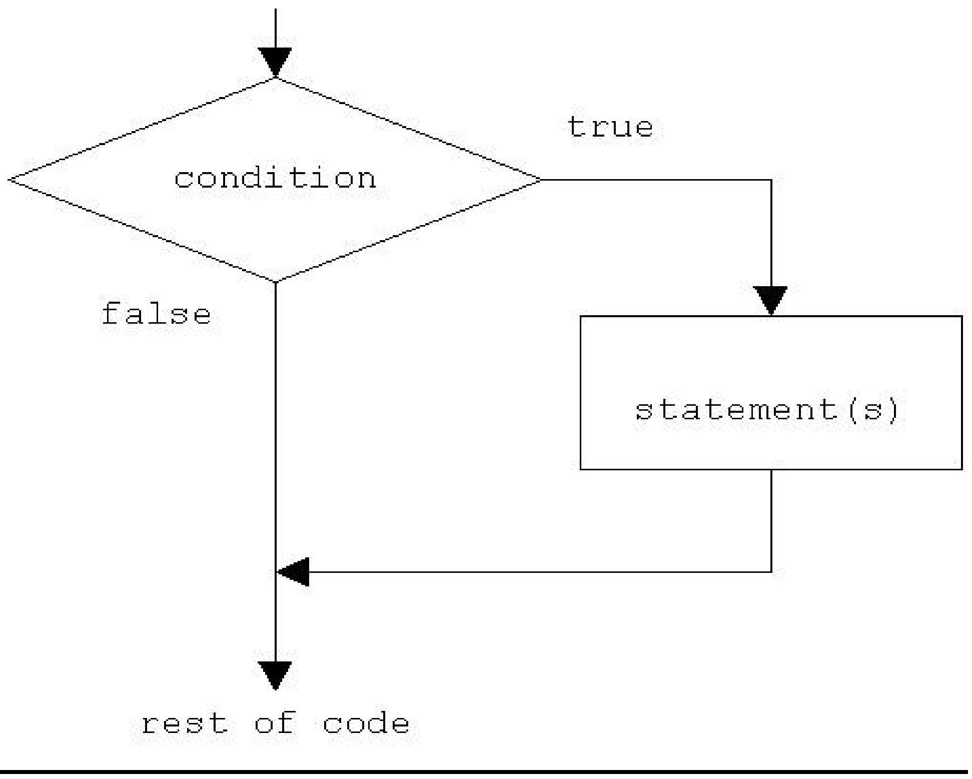

Fig 2: *if* statement construct

Syntax of if statement:

```
if (condition )
{
Statement 1;
…………..
Statement n;
}
//Rest of the code
```

If the condition is true(nonzero), the statement will be executed. If the condition is false(0), the statement will not be executed. For example, suppose we are writing a billing program.

```
if (total_purchase >=1000)
printf("You are gifted a pen drive.\n");
```

Multiple statements may be grouped by putting them inside curly braces {}. For example:

```
if (total_purchase>=1000)
{
gift_count++;
printf("You are gifted a pen drive.\n");
}
```

For readability, the statements enclosed in {} are usually indented. This allows the programmer to quickly tell which statements are to be conditionally executed. As we will see later, mistakes in indentation can result in programs that are misleading and hard to read.

Programs:

1. Write a program to print a message if negative no is entered.

```
#include<stdio.h> int main()
{
int no;
printf("Enter a no : ");
scanf("%d", &no); if(no<0)
{
printf("no entered is negative"); no = -no;
}
printf("value of no is %d \n",no);
return 0; }
```

Output:

```
Enter a no: 6 value of no is 6
```
Output:
```
Enter a no: -2 value of no is 2
```

2. Write a program to perform division of 2 nos
```
#include<stdio.h> int main()
{
int a,b; float c;
printf("Enter 2 nos : ");
scanf("%d %d", &a, &b);
if(b == 0)
{
printf("Division is not possible");
}
c = a/b;
printf("quotient is %f \n",c);
return 0;

}
```

Output:
```
Enter 2 nos: 6 2 quotient is 3
```
Output:
```
Enter 2 nos: 6 0
Division is not possible
```

if-else Statement
The *if* statement by itself will execute a single statement, or a group of statements, when the expression following *if* evaluates to true. By using *else* we execute another group of statements if the expression evaluates to false.
```
if (a > b)
{ z = a;
printf("value of z is :%d",z);
}
else
{ z = b;
printf("value of z is :%d",z);
}
```

The group of statements after the *if* is called an 'if block'. Similarly, the statements after the else form the 'else block'.

Programs:
3. Write a program to check whether the given no is even or odd
```
#include<stdio.h>
int main()
{
int n;
printf("Enter an integer\n"); scanf("%d",&n);
if ( n%2 == 0 )
printf("Even\n");
 else
printf("Odd\n");
return 0;
}
```

Output:
```
Enter an integer 3
Odd
```
Output:
```
Enter an integer 4
Even
```

4. Write a program to check whether a given year is leap year or not
```
#include <stdio.h>
int main()
{
int year;
printf("Enter a year to check if it is a leap year\n");
scanf("%d", &year);
if ( (year%4 == 0) && (( year%100 != 0) || ( year%400 == 0 ))
printf("%d is a leap year.\n", year);
else
printf("%d is not a leap year.\n", year);
return 0;
}
```

Output:
```
Enter a year to check if it is a leap year 1996
1996 is a leap year
```
Output:

```
Enter a year to check if it is a leap year 2015
2015 is not a leap year
```

Nested *if-else*

An entire *if-else* construct can be written within either the body of the *if* statement or the body of an *else* statement. This is called 'nesting' of *if*s. This is shown in the following structure.

```
if (n > 0)
{
if (a > b)
z = a;
}
else
z = b;
```

The second *if* construct is nested in the first *if* statement. If the condition in the first *if* statement is true, then the condition in the second *if* statement is checked. If it is false, then the *else* statement is executed.

Program:
5. Write a program to check for the relation between 2 nos

```
#include <stdio.h>
int main()
{
int m=40,n=20;
if ((m >0 ) && (n>0))
{
printf("nos are positive");
if (m>n)
{
printf("m is greater than n");
}
else
{
printf("m is less than n");
}
}
else
{
printf("nos are negative");
}
return 0;
}
```

Output

```
40 is greater than 20
```

***else-if* Statement:**

This sequence of if statements is the most general way of writing a multi−way decision. The expressions are evaluated in order; if an expression is true, the statement associated with it is executed, and this terminates the whole chain. As always, the code for each statement is either a single statement, or a group of them in braces.

```
If (expression)
statement
else if (expression)
statement
else if (expression)
statement
else if (expression)
statement
else
statement
```

The last *else* part handles the ``none of the above'' or default case where none of the other conditions is satisfied. Sometimes there is no explicit action for the default; in that case the trailing can be omitted, or it may be used for error checking to catch an “impossible” condition.

Program:

6. The above program can be used as an eg here.

```
#include <stdio.h>
int main()
{
        int m=40,n=20; if (m>n)
{
        printf("m is greater than n");
}
        else if(m<n)
{
        printf("m is less than n");
}
        else
{
        printf("m is equal to n");
        }
}
```

Output:

```
m is greater than n
```

switch case:

This structure helps to make a decision from the number of choices. The *switch* statement is a multi−way decision that tests whether an expression matches one of a number of constant integer values, and branches accordingly [3].

```
switch( integer expression)
{
        case constant 1 :
        do this;
        case constant 2 :
        do this ;
        case constant 3 :
        do this ;
        default :
        do this ;
}
```

The integer expression following the keyword switch is any C expression that will yield an integer value. It could be an integer constant like 1, 2 or 3, or an expression that evaluates to an integer. If a case matches the expression value, execution starts at that case. All case expressions must be different. The case labelled *default* is executed if none of the other cases are satisfied. A default is optional; if it isn't there and if none of the cases match, no action at all takes place. Cases and the default clause can occur in any order.

Consider the following program:

```
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
```

Here the program prints case 2 and 3 and the default case. If you want that only case 2 should get executed, it is up to you to get out of the switch then and there by using a

***break* statement.**

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

The output of this program would be:

```
I am in case 2
```

Program

7. WAP to enter a grade & check its corresponding remarks.

```
#include <stdio.h> int main ()
{
char grade;
printf("Enter the grade");
scanf("%c", &grade);
switch(grade)
{
case 'A' :printf("Outstanding!\n" );
break;
case 'B' : printf("Excellent!\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;
}
```

Output

```
Enter the grade
B
Excellent
Your grade is B
```

ITERATIVE STATEMENTS

***while* statement**

The *while* statement is used when the program needs to perform repetitive tasks. The general form of a *while* statement is:

while (condition)
statement ;

The program will repeatedly execute the statement inside the *while* until the condition becomes false(0). (If the condition is initially false, the statement will not be executed.) Consider the following program:

```
main( )
{
 int p, n, count;
float r, si;
count = 1;
while ( count <= 3 )
{
printf ( "\nEnter values of p, n and r " ) ; scanf("%d %d %f", &p, &n, &r ) ;
si=p * n * r / 100 ;
printf ( "Simple interest = Rs. %f", si ) ;
count = count+1;
}
}
```

Some outputs of this program.

```
Enter values of p, n and r 1000 5 13.5
Simple Interest = Rs. 675.000000
Enter values of p, n and r 2000 5 13.5
Simple Interest = Rs. 1350.000000
Enter values of p, n and r 3500 5 13.5
Simple Interest = Rs. 612.000000
```

The program executes all statements after the *while* 3 times. These statements form what is called the ‘body’ of the *while* loop. The parentheses after the *while* contain a condition. As long as this condition remains true all statements within the body of the *while* loop keep getting executed repeatedly.

Consider the following program;

```
/* This program checks whether a given number is a palindrome or not */
#include <stdio.h>
int main()
{
int n, reverse = 0, temp;
printf("Enter a number to check if it is a palindrome or not\n");
scanf("%d",&n);
temp = n;
while( temp != 0 )
{
reverse = reverse * 10;
reverse = reverse +temp%10;
temp = temp/10;
}
if ( n == reverse )
printf("%d is a palindrome number.\n", n); else
printf("%d is not a palindrome number.\n", n);
return 0;
}
```

Output:

```
Enter a number to check if it is a palindrome or not
12321
12321 is a palindrome
Enter a number to check if it is a palindrome or not
12000
12000 is not a palindrome
```

do-while Loop

The body of the *do-while* executes at least once. The *do-while* structure is similar to the *while* loop except the relational test occurs at the bottom (rather than top) of the loop. This ensures that the body of the loop executes at least once. The *do-while* tests for a positive relational test; that is, as long as the test is True, the body of the loop continues to execute.

The format of the do-while is

```
do
{ block of one or more C statements; }
while (test expression)
```

The test expression must be enclosed within parentheses, just as it does with a while statement.

Consider the following program

```
// C program to add all the numbers entered by a user until user enters 0.
#include <stdio.h>
int main()
{
 int sum=0,num;
do /* Codes inside the body of do...while loops are at least executed once. */
{
printf("Enter a number\n");
scanf("%d",&num);
sum+=num;
}
while(num!=0);
printf("sum=%d",sum);
return 0;
}
```

Output:

```
Enter a number 3
Enter a number -2
Enter a number 0
sum=1
```

Consider the following program:

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

Output

```
Hello 10
Hello 9
Hello 8
Hello 7
Hello 6
Hello 5
Hello 4
Hello 3
Hello 2
Hello 1
```

Program

8. Program to count the no of digits in a number

```
#include <stdio.h>
int main()
{
int n,count=0;
printf("Enter an integer: ");
scanf("%d", &n);
do
{
n/=10; /* n=n/10 */
count++;
} while(n!=0);
printf("Number of digits: %d",count);
}
```

Output

```
Enter an integer: 34523
Number of digits: 5
```

<u>for Loop</u>

The *for* is the most popular looping instruction. The general form of *for* statement is as under:

```
for ( initialise counter ; test counter ; Updating counter )
{
do this;
and this;
and this;
}
```

The *for* allows us to specify three things about a loop in a single line:

(a) Setting a loop counter to an initial value.

(b) Testing the loop counter to determine whether its value has reached the number of repetitions desired.

(c) Updating the value of loop counter either increment or decrement.

Consider the following program

```
int main(void)
{
int num;
printf(" n n cubed\n");
for (num = 1; num <= 6; num++)
printf("%5d %5d\n", num, num*num*num); return 0;
}
```

The program prints the integers 1 through 6 and their cubes.

```
n n cubed
1 1
2 8
3 27
4 64
5 125
6 216
```

The first line of the *for* loop tells us immediately all the information about the loop parameters: the starting value of num, the final value of num, and the amount that num increases on each looping [5].

Grammatically, the three components of a *for* loop are expressions. Any of the three parts can be omitted, although the semicolons must remain.

Consider the following program:

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

Here, the increment is done within the body of the *for* loop and not in the *for* statement. Note that in spite of this the semicolon after the condition is necessary.

Programs:

9. Program to print the sum of 1st N natural numbers.

```
#include <stdio.h>
int main()
 {
int n,i,sum=0;
printf("Enter the limit: ");
scanf("%d", &n);
for(i=1;i<=n;i++)
{
sum = sum +i;
}
printf("Sum of N natural numbers is: %d",sum);
}
```

Output

```
Enter the limit: 5
Sum of N natural numbers is 15.
```

10. Program to find the reverse of a number

```
#include<stdio.h>
int main()
{
int num,r,reverse=0;
printf("Enter any number: ");
scanf("%d",&num);
for(;num!=0;num=num/10)
{
r=num%10;
reverse=reverse*10+r;
}
printf("Reversed of number: %d",reverse);
return 0;
}
```

Output:

```
Enter any number: 123
Reversed of number: 321
```

NESTING OF LOOPS

C programming language allows using one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax:

The syntax for a nested for loop statement in C is as follows:

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

The syntax for a nested while loop statement in C programming language is as follows:

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

The syntax for a nested do...while loop statement in C programming language is as follows:

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

A final note on loop nesting is that you can put any type of loop inside of any other type of loop. For example, a for loop can be inside a while loop or vice versa.

Programs:

11. program using a nested for loop to find the prime numbers from 2 to 20:

```
#include <stdio.h>
int main ()
{
/* local variable definition */ int i, j;
for(i=2; i<20; i++)
{
for(j=2; j <= (i/j); j++)
if(!(i%j))
break; // if factor found, not prime
if(j > (i/j)) printf("%d is prime\n", i);
}
return 0;
}
```

Output

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
```

```
17 is prime
19 is prime
```

12. *
```
***
*****
*******
*********
```

```
#include <stdio.h>
int main()
{
int row, c, n,I, temp;
printf("Enter the number of rows in pyramid of stars you wish to see ");
scanf("%d",&n);
temp = n;
for ( row = 1 ; row <= n ; row++ )
{
for ( i= 1 ; i < temp ; i++ )
{
printf(" ");
temp--;
for ( c = 1 ; c <= 2*row - 1 ; c++ )
{
printf("*");
printf("\n");
}
}
}
return 0;
}
```

13. Program to print series from 10 to 1 using nested loops.

```
#include<stdio.h>
void main ()
{
int a;
a=10;
for (k=1;k=10;k++)
{
while (a>=1)
{
printf ("%d",a);
a--;
} printf("\n");
a= 10;
}
}
```

Output:

```
10 9 8 7 5 4 3 2 1
10 9 8 7 5 4 3 2 1
10 9 8 7 5 4 3 2 1
10 9 8 7 5 4 3 2 1
10 9 8 7 5 4 3 2 1
10 9 8 7 5 4 3 2 1
10 9 8 7 5 4 3 2 1
10 9 8 7 5 4 3 2 1
10 9 8 7 5 4 3 2 1
10 9 8 7 5 4 3 2 1
```

JUMP STATEMENTS

The break Statement

The *break* statement provides an early exit from *for*, *while*, and *do*, just as from *switch*. A *break* causes the innermost enclosing loop or switch to be exited immediately. When *break* is encountered inside any loop, control automatically passes to the first statement after the loop.

Consider the following example;

```
 main( )
 {
 int i = 1 , j = 1 ;
 while ( i++ <= 100 )
 {
 while ( j++ <= 200 )
 {
 if ( j == 150 )
 break ;
 else
```

```
printf ( "%d %d\n", i, j );
}
}
}
```

In this program when j equals 150, break takes the control outside the inner while only, since it is placed inside the inner *while*.

The *continue* Statement

The *continue* statement is related to *break*, but less often used; it causes the next iteration of the enclosing *for, while*, or *do* loop to begin. In the *while* and *do*, this means that the test part is executed immediately; in the *for*, control passes to the increment step. The *continue* statement applies only to loops, not to switch.

Consider the following program:

```
main( )
{
        int i, j ;
        for( i = 1 ; i <= 2 ; i++ )
        {
                for ( j = 1 ; j <= 2 ; j++ )
                {
                if ( i == j)
                        continue ;
                printf ( "\ n%d %d\n", i, j ) ;
                }
        }
}
```

The output of the above program would be...

```
1 2
2 1
```

Note that when the value of I equals that of j, the *continue* statement takes the control to the *for* loop (inner) by passing rest of the statements pending execution in the *for* loop (inner).

The *goto* statement

Kernighan and Ritchie refer to the *goto* statement as "infinitely abusable" and suggest that it "be used sparingly, if at all.

The *goto* statement causes your program to jump to a different location, rather than execute the next statement in sequence. The format of the *goto* statement is;

goto statement label;

Consider the following program fragment

```
if (size > 12)
goto a;
goto b;
a: cost = cost * 1.05;
flag = 2;
b: bill = cost * flag;
```

Here, if the *if* conditions satisfies the program jumps to block labelled as *a:* if not then it jumps to block labelled as *b:*.