LECTURE NOTE -10

CONTROL STATEMENTS

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:

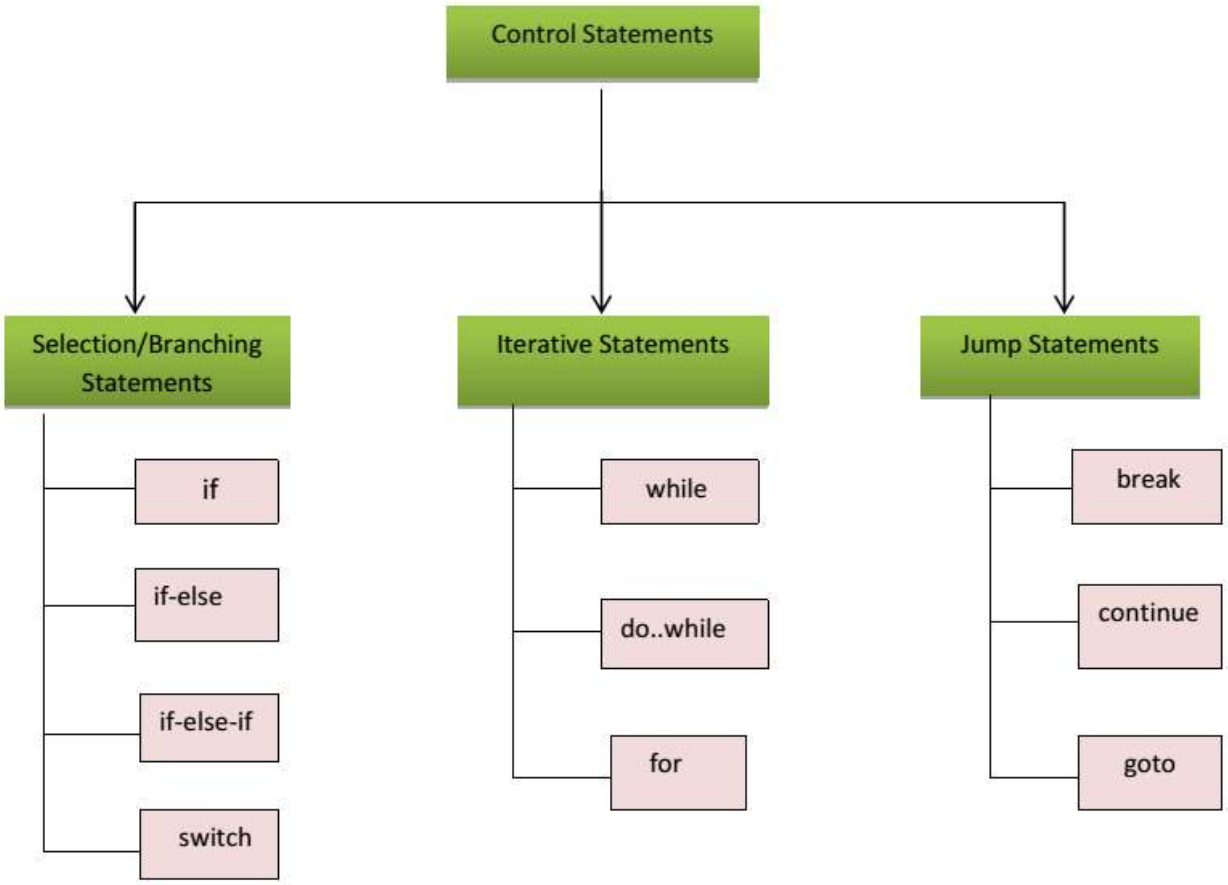

Fig: 1 Classification of control statements

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:

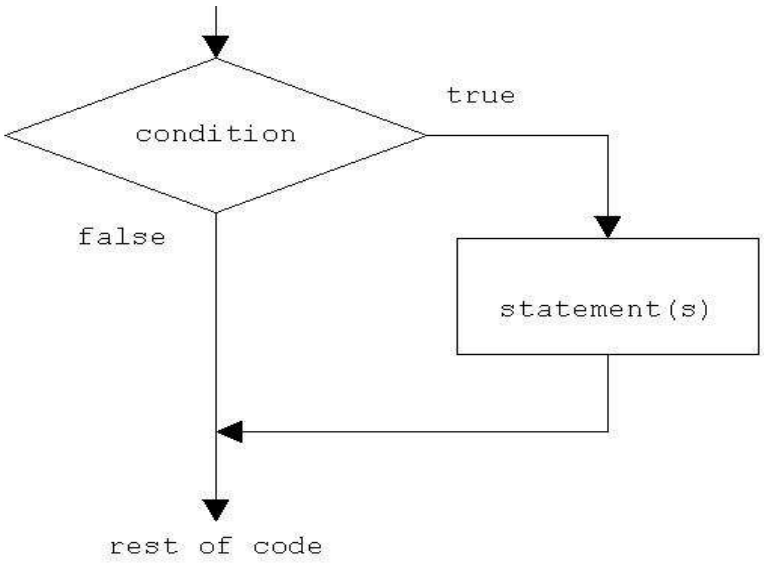

Fig 2: *if* statement construct

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
```