

## MODULE-II

### **CONTROL STATEMENTS**

#### SELECTION/BRANCHING STATEMENTS

The instructions are executed in sequence order in the program. There may be many situations where such execution sequence may not be sufficient.

For example, one want to find the bigger among two numbers. It requires comparison of two numbers, and based on comparison result, the bigger number is found. This can be implemented using if instruction.

The 'if' statement is a powerful decision making statement and is used to control the flow of execution of statements. The 'if' statement may be implemented in different forms depending on the complexity of condition to be tested.

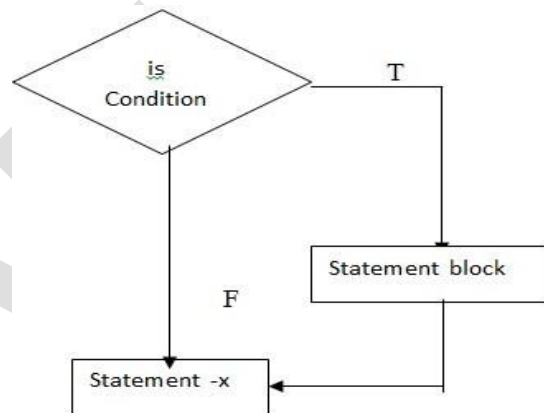
**a). Simple ' If ' statement:** This is one-way branching statement.

The general form of a simple if is as follows;

Syntax:

```
if(Testexpression)
{
.....
Statements
.....
}
```

Flowchart:



Ex: if(marks>=60)

```
{
marks=marks+10;
}
printf("%d", marks);
```

The statement block may be a single statement or a group of statements.

If the test expression is true, the statement block will be executed, otherwise the statement-block will be skipped and the execution will jump to statement-x.

**b). The ' if..else ' statement:**

- This is two-way branching statement.
- If the test expression is true, then the true-block statements will be executed, otherwise the false-block statements will be executed.
- In both the case the control is transferred subsequently to statement-x.

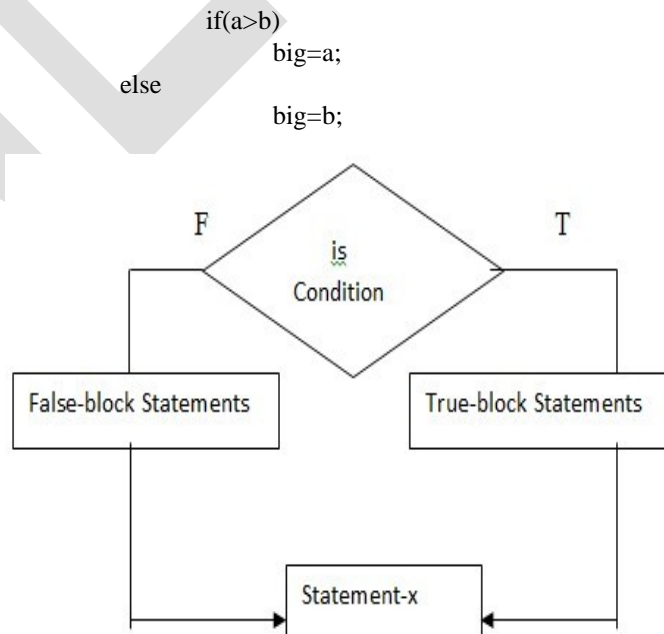
Example:

```
if(marks>=35)
    res="pass";
else
    res="fail";
printf("\n Result=%s",res);
printf("\n biggest=%d",big);
```

The general form is as follows;

Syntax:

```
if(Test expression)
{
True-block statements;
}
else
{
False-block statements;
}
statement-x;
```

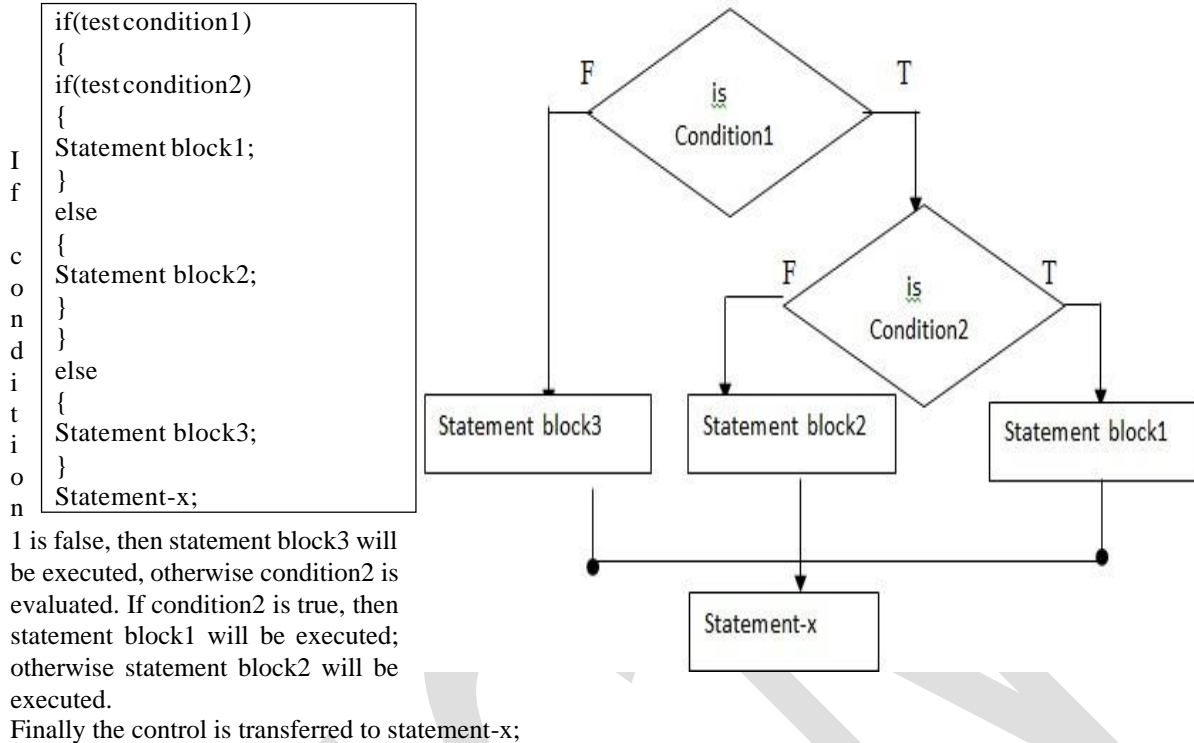


**c). 'Nested if..else ' statement:**

- When a series of decisions are involved, we have to use more than one if..else statement in nested form.

- This is a multi-way branching statement.

Syntax: Flowchart:



```

/* a program to find biggest among two */
main()
{
inta,b,big;
printf("\n Enter two numbers");
scanf("%d %d",&a,&b);
if(a>b)
big=a;
else
big=b;
printf("\n the biggest =%d",big);
getch();
}

```

```

/* a program to find biggest among three numbers */
main()
{
inta,b,c,big;
printf("\n Enter three numbers");
scanf("%d %d %d",&a,&b,&c);
if(a>b)
{ if(a>c)
big=a;
else
big=c;
}
else
{ if(c>b)
big=c;
else
big=b;
}
Printf("\n biggest =%d",big);
getch();
}

```

#### d. The else..if Ladder’:

- When a multi-path decisions are involved, in which the statement associated with *each else is an anotherif*.
- The conditions are evaluated from the top downwards. As soon as a true condition is found, then the statement block associated with it is executed and the control is transferred to statement-x.
- When all the n conditions become false, then the final else containing the default- statement will be executed.

```

if(condition1)
{
statement block-1;
}

```

Example:  
if  
(a  
vg  
>=  
80

```
)  
gr  
ad  
e=  
”di  
sti  
nct  
io  
n;  
els  
e  
if(  
av  
g>  
=6  
0)
```

```

else
if(condition2)
{
Statement block2;
}
..... else
if(condition n)
{
Statement block n;
}
else
default-Statement; statement-
x;

```

```

grade="first
class"; else
if(avg>=50)
grade="sec
ond class";
else
if(avg>=35)
grade="thir
d class"
else
grade="fail";
printf("\n Grade = %s",grade);

```

### Switch statement:

- It is a multi-way decision statement, it tests the value of a given variable or expression against a list of case values and when a match is found, a block of statements associated with that case is executed.
- The general form of the switch is as follows.

```
switch(expression)
```

```
{
case value1:
    block1;
    break;
case value2:
    block2;
    break;
....
....
default:
    default block;
    break;

```

```
main()
```

```
{
char choice; clrscr();
printf("\n enter a choice ");
switch(choice=toupper(getchar()))
{
case 'R':
printf("\n RED COLOR"); break;
case 'W':
printf("\n WHITE COLOR"); break;
case 'B':
printf("\n BLUE COLOR"); break;
default: printf("\n Invalid color choice"); break;
}
getch();
}

```

- ☐ The expression is an integer expression or characters.
- ☐ value1, value 2,... are constants and are known as case labels, and they end with a colon (:).
- ☐ The break statement at the end of each block signal the end of a particular case and causes exit from the switch statement and transfers the control to the statement-x.
- ☐ When the value of the expression does not match with any of the case values, then the default block will be executed.
- ☐ When the case labels are characters, then they must be enclosed in single quotes.

```
main()
```

```
{
int x,y,cho
ice;
clrscr();
printf("\n enter two
numbers"); scanf("%d
%d",&x,&y);
printf("\n AIRTMETIC
OPERATIONS");
printf("\n=====
");
printf("\n 1. ADDITION");
printf("\n 2. SUBTRACTION");
printf("\n 3. MULTIPLICATION");
printf("\n 4. DIVISION");
printf("\n Ener u r
choice:"); scanf("%d",
choice); switch(choice)
{
case 1: printf("\n
addition=%d", (x+y));
break;
case 2: printf("\n subtraction
=%d", (x-y)); break;
case 3: printf("\n multiplication
=%d", (x*y)); break;
case 4: printf("\n Division=
%d", (x/y)); break;

```

```
default: printf("\n Invalid\n");  
choice"); break;  
}  
getch();  
}
```

**ITERATIVE/LOOPING STATEMENTS****The While statement:**

- The while statement is used to carry out looping operations, in which a group of statements is executed repeatedly, until some condition has been satisfied.
- The general form of while statement is  
**while(expression/condition)**

{

**Body of the loop;**

.....

}

The body of the loop will be executed as long as the expression/condition is true. Example:

i=1;

while(i&lt;=10)

{

printf("%d \t",i); i++;

}

The following output will be generated when the program is executed.

12345678910

- This is an entry-controlled loop. The body of the loop is executed only when the test condition is true.

**The do-while statement:**

- This is an exit-controlled loop. The condition is tested at end of the body of the loop is executed.
- The general form of do-while statement is,

**do**

{

**Body of the loop;**

.....

**}while(expression/condition);**

- The body of the loop is executed repeatedly, as long as the value of expression is true.
- The body of the loop will be executed at least once even if the condition fails.

i=1;

do

{

printf("\n %d",i);

i++;

}while(i&lt;=10);

The following output will be generated when the program is executed. 1

2 3 4 5 6 7 8 9 10

- For many applications it is more natural to test for continuation of loop at the beginning rather than at the end of loop. For this reason do-while statement is used less frequently than the while statement.

**The for Statement :**

- The 'for' first statement is third and the most commonly used looping statement in c.
- This statement includes an expressions that specifies an initial value for an index, another expression that determines whether or not the loop is continued, and a third expressions that allows the index to be modified at the end of each pass.

- The general form of the for statement

**for( expression1; expression2; expression3)**

{

**Body of the loop;**

}

- Where expressions1 is used to **initialize** some parameter (called an index ) that controls the looping action, expression2 represents a **condition** that must be true to continue execution, and expression3 is used to alter the value of parameter initially assigned by expression 1. Typically, expression1 is an assignment.
- When theforstatement is executed, expression 2 is evaluated and tested at the beginning of each pass through the loop, and expression 3 is evaluated at the end of each pass.
- The looping action will continue as long as the value of expression2 is true.

for(i=1;i&lt;=10;i++)

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

The following output will be generated when the program is executed. 1

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

- All three expressions need not be included in the for statement, though the semicolons must be present. The first and third expressions may be omitted such as initialization of index and increment/decrement of the index.

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

A program to find the average of a list of numbers

```
#include<stdio.h>
#include<conio.h>
main()
{
intn,count;
floatx,avg,sum=0;
clrscr();
printf("\n how many numbers:");
scanf("%d",&n);
for(count=1;count<=n;count++)
{
printf("\n Enter element:");
scanf("%d",&x); sum=sum+x;
}
avg=sum/n;
printf("\n Average is %f",avg); getch();
}
```

### **Nested Control Structures:**

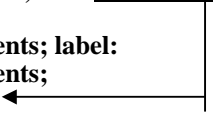

- A control structure can be nested, one within another.
- The inner and outer loops need not be generated by the same type of control structure. Each loop must be controlled by a different index.

A program to display the product table using nested loops

```
#include<stdio.h>
#include<conio.h>
main()
{
inti,j,n;
clrscr();
printf("\n how many numbers:");
scanf("%d",&n);
printf("\n PRODUCT TABLE \t");
i=1;
while(i<=n)
{
for(j=1;j<=n;j++)
{
printf("%d \t",i*j);
}
i++;
}
getch();
}
```

**Goto Statement:**

- This is an unconditional statement. It is used jump in program from one statement to another statement.
- Its general form is : **goto label;**
- The control will be transferred directly to the **label:** statement in the program.
- There are two ways to jump in the program either it may be forward or backward jump.

<b><u>Forward jump</u></b>	<b><u>Backward jump</u></b>
<pre> ..... statements; goto label; ..... Statements; label: Statements; </pre> 	<pre> ..... label: statements; ..... goto label; ..... statements; </pre> 

The statement **label:** Is after the **goto label;** statement, then it is called forward jump, Is before the **goto label;** statement, it is called backward jump.

```

/* a program for forward jump */ main()
{
int x; clrscr();
printf("\n Enter a number"); scanf("%d",&x);
if(x<0) goto end; else
{
printf("\n the root =%lf",sqrt(x)); getch();
exit(1);
}
end:
printf("\n the root is not possible"); getch();
}

```

```

/* a program for
backward jump */
main()
{
i
n
t
x
;
c
l
r
s
c
r
(
)
;
s
t
a
r
t
:
printf("\n Enter a
number");
scanf("%d",&x);
if(x<0)
{
printf("\n the root is not
possible"); goto start;
}
else
printf("\n the root
=%lf",sqrt(x)); getch();
}

```


**The break statement:**

- The break statement is used to terminate a loop or to exit from a switch statement.
- It can be used in a while, do-while and for statements.
- The general form is,  
**break;**
- The break statement causes a transfer of control out of the loop, to the first statement following the loop.

```

while(condition)
{
.....
}
if(condition) break;
Statements;
Exit from
loop

```



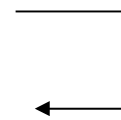


```
do
{
.....
if(condition) break;
.....
} while(condition);
Statements;
```

Exit from  
loop

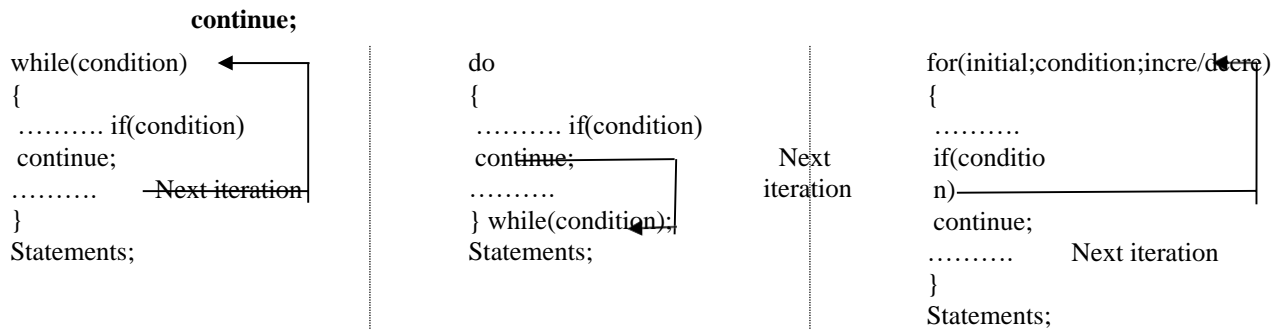
```
for(initial;condition;incre/decre)
{
.....
if(condition)
break;
.....
}
Statements;
```

Exit from  
loop



### **The continue statement:**

- The continue statement is used to bypass the remainder of the current pass through a loop.
- The loop does not terminate, but the remaining loop statements are skipped and the computation proceeds directly to the next pass through the loop.
- It can be used in a while, do-while, and for statements.
- The syntax is,



```

/* find average for a list of Non-Negative
numbers */ #include<stdio.h>
#include<conio.h>
h> main( )
{
    int n, count,
    avg=0;
    float
    x, avg, sum
    =0;
    clrscr();
    printf("\n How many
    numbers:");
    scanf("%d",&n);
    for(count=0; count<n; count++)
    {
        printf("\n Enter x value:");
        scanf("%f",&x);
        if(x
        <0)
            continue;
        else
            sum=sum+x;
            ++avg;
        }
    avg=sum/avg;
    printf("\n Average is:%f",avg);
    printf("\n Number of +ve
    elements:%d",avg); getch();
}

```

### Result

How many numbers: 6 Enter x value:  
 3  
 Enter x value:-3 Enter x  
 value: 2 Enter x value:-2  
 Enter x value: 4 Enter x  
 value:-4 Average is:3.00000  
 Number of +ve elements: 3