Unit
3

Decision Statements and Control Structures

- Conditional Branching Statements
- If Statrement

3.1 Simple if Statement

3.2 if...else Statement

3.3 Nested if...else Statement

3.4 else if Ladder

3.5 switch Statement

3.6 Unconditional Branching Statement: GOTO

3.7 Control Statements / Looping Statements

3.8 for Loop

3.9 while Loop

3.10 do....while Loop

- Difference between while and do...while

3.11 Nested for loop

3.12 Break and Continue Statement

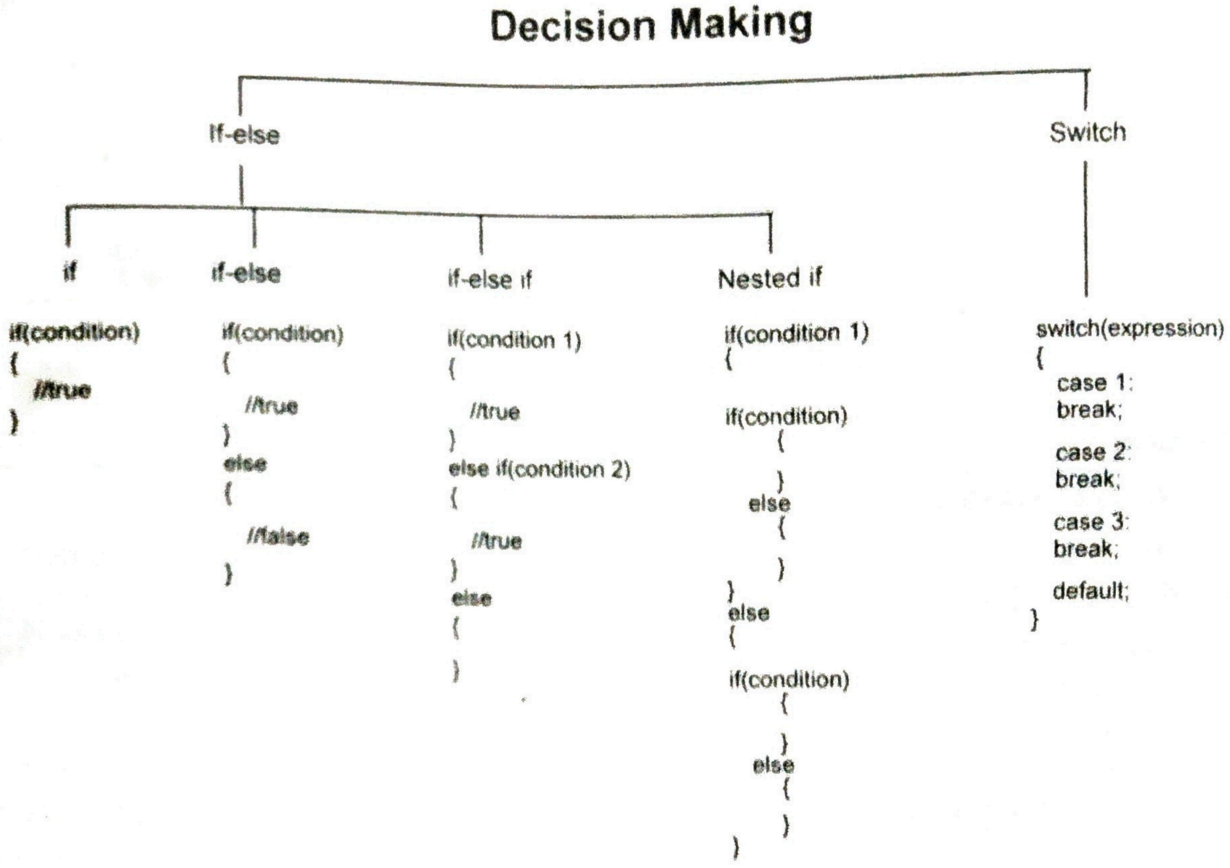
- Programs of conditional structures
- Programs of loop structures
- Programs of various patterns
- important Programs

◆ **Exercise**

◆ **GTU Exam. Paper Solution**

■ Conditional Branching Statements

In programming we need to make some decisions and based on these decisions we will execute the next block of code. For example, in C if x occurs then execute y else execute z. There can also be multiple conditions like in C if x occurs then execute p, else if condition y occurs execute q, else execute r. This condition of C else-if is one of the many ways of importing multiple conditions.



■ IF STATEMENT

if Statement is basically two way decision statement. The different forms of if statements are:

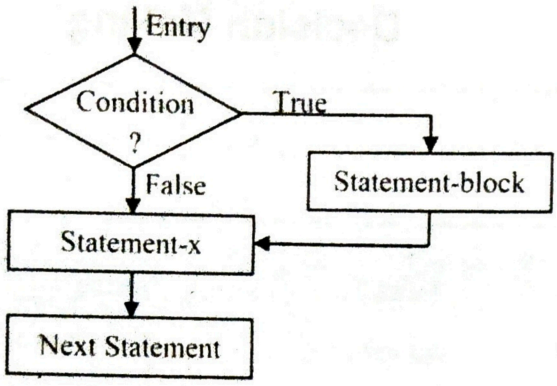
1. Simple if Statement
2. if...else Statement
3. Nested if...else Statement
4. elseif Ladder

3.1 SIMPLE IF STATEMENT

General format of Simple if Statement is:

```
if (condition)
{
Statement-block;
}
Statement-x;
```

The 'statement-block' may be a single statement or group of statements. If the condition is true, the statement-block will be executed; otherwise the statement-block will be skipped and execution will jump to the statement-x.



Example:

```
#include<stdio.h>
void main()
{
    int a,b;
    scanf("%d %d",&a,&b);
    if (a>b)
    {
        printf("Hi\n");
    }
    printf("Hello");
}
```

Output:

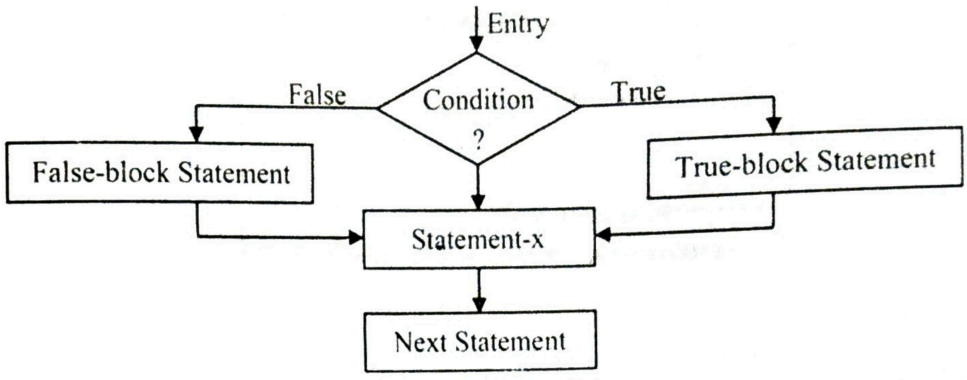
```
5 4                5 6
Hi                 Hello
Hello
```

3.2 IF...ELSE STATEMENT

General format of if...else Statement is:

```
if (condition)
{
true-block statement(s)
}
else
{
false-block statement(s)
}
statement-x
```

If the condition is true, then true-block statement(s) are executed. Otherwise, the false-block statement(s) are executed.



Example:

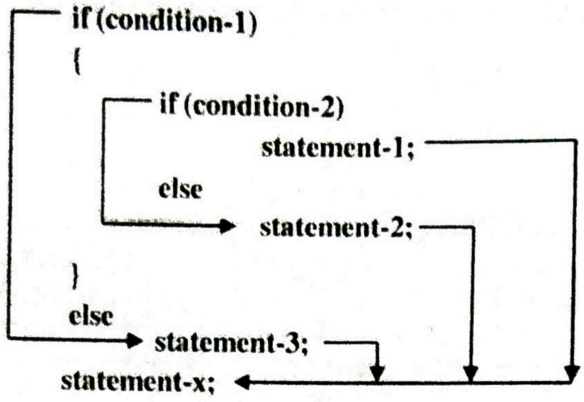
```
#include<stdio.h>
void main()
{
int a,b;
scanf("%d %d",&a,&b);
if (a>b)
{
printf("a is greater than b.\n");
}
else
{
printf("b is greater than a.\n");
}
printf("a and b are compared.");
}
```

Output:

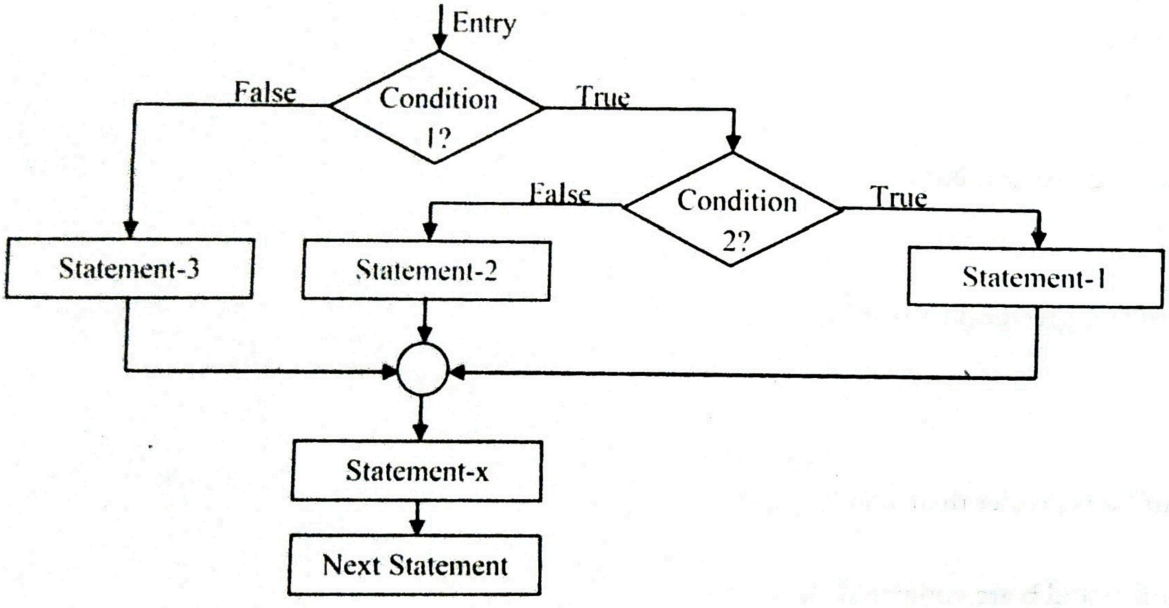
53	5 6
a is greater than b.	b is greater than a.
a and b are compared.	a and b are compared.

3.3 NESTED IF...ELSE STATEMENT

When series of decisions are involved, Nested if.....else Statements can be used. General format of Nested if...else Statement is:



If the condition-1 is false, the statement-3 will be executed; otherwise it continues to perform the condition-2. If the condition-2 is true, the statement-1 will be evaluated; otherwise the statement-2 will be evaluated and then the control is transferred to the statement-x.



Example:

```
#include<stdio.h>
void main()
{
int a,b,c;
scanf("%d %d %d",&a,&b,&c);
if (a>b)
{
if(a>c)
printf("a is greater than b & c.\n");
else
printf("c is greater than a & b.\n");
```

```
}
else
{
if(b>c)
printf("b is greater than a & c.\n");
else
printf("c is greater than a & b.\n");
}
printf("a, b and c are compared.");
}
```

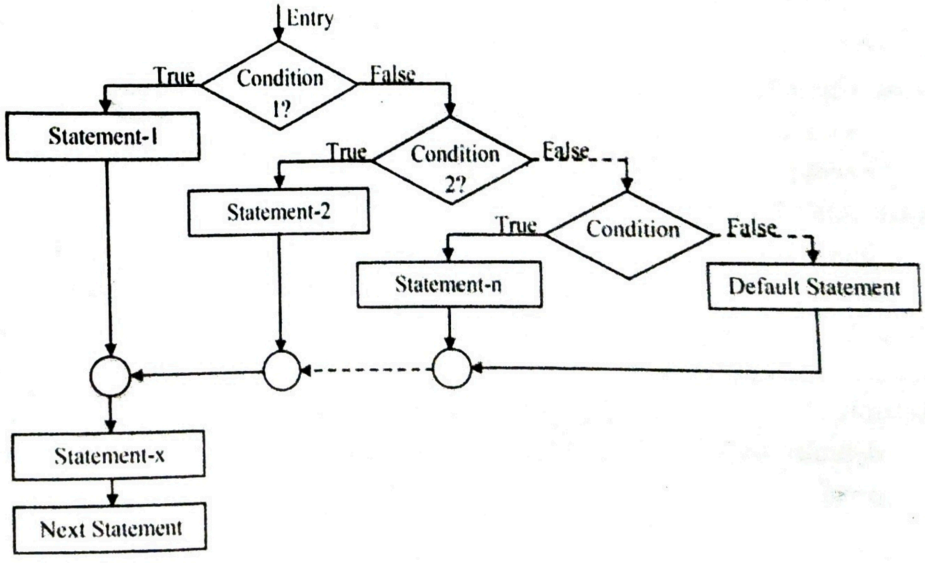
Output:

```
5 1 3
a is greater than b & c.
a, b and c are compared.
```

3.4 ELSE IF LADDER

When multiple decisions are involved, else if Ladder can be used. General format of else if Ladder is:

```
if (condition-1)
        statement-1;
else if (condition-2)
        statement-2;
else if (condition-n)
        statement-n;
else
        default-statement;
statement-x;
```



Here conditions are evaluated from the top, downwards. As soon as true condition is found, the statement associated with it is executed and the control is transferred to the statement-x. When all the n conditions become false then the final else containing the default-statement will be executed.

Example:

```
#include<stdio.h>
void main()
{
int a,b,c;
scanf("%d %d %d",&a,&b,&c);
if (a>b && a>c)
printf("a is greater than b & c \n");
else if (b>a && b>c)
printf("b is greater than a & c \n");
else
printf("c is greater than a & b \n");
printf("a, b and c are compared.");
}
```

Output:

```
2  1  3
c is greater than a & b
a, b and c are compared.
```

3.5 SWITCH STATEMENT

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

```
switch (expression)
{
        case value-1:
            block-1
            break;
        case value-2:
            block-2
            break;
        ..................
        ..................
        default:
            default-block
            break;
}
```

statement-x;

Where 'expression' is an integer expression or characters.

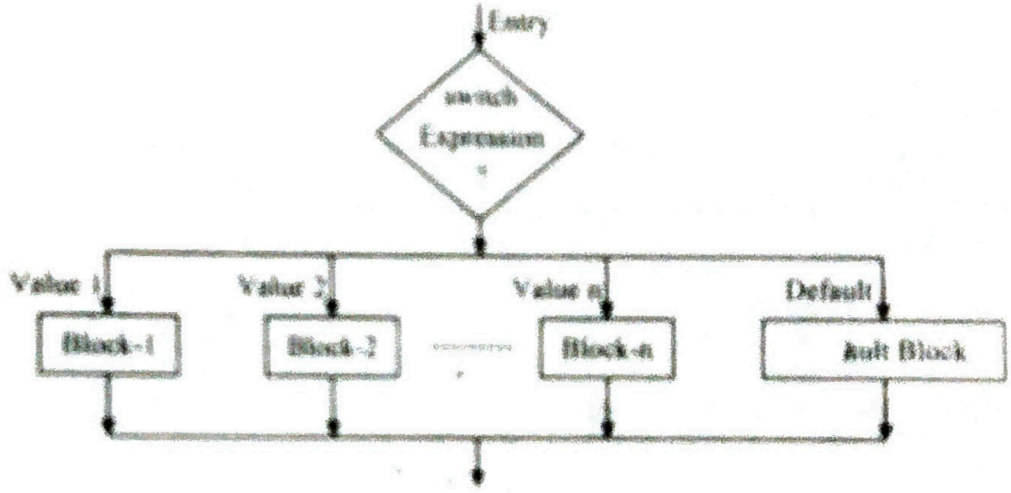
'value-1', 'value-2' ..... are constants or constant expressions and are known as **case labels**

'block-1', 'block-2' ..... are statement lists and may contain zero or more statements.

When switch is executed, the value of the expression is successfully compared against the values 'value-1', 'value-2' ..... If a case is found whose value matches with the value of the expression, then the block of statements that follow the case are executed.

The **break** statement at the end of each block signal the end of particular case and causes an exit from the **switch** statement, transferring the control to the **statement-x** following the **switch**.

The **default** is an optional case. If it present, it will be executed if the value of the expression does not match with any of the case values. If not present, no action takes place if all matches fail and the control goes to the 'statement-x'.



Example:

```
#include<conio.h>
void main()
{
int mark,result;
printf("\n\nEnter mark:");
scanf("%d",&mark);
result=mark/10;
switch(result)
{
case 10:
case 9:
case 8:
printf("Distinction");
break;
case 7:
case 6:
```

```
printf("First Class");
break;
case 5:
case 4:
printf("Second Class");
break;
default :
printf("Fail");
break;
}
getch();
}
```

Output:

89	09
Distinction	Fail

if...else Vs switch-case

The if...else	The switch-case
– It is a two-way branching.	– It is a multi-way branching.
– The nested if...else is complicated.	– It is easier to write.
– The nested else...if can take many levels and it becomes difficult to match the else part to its corresponding if.	– No such problem occurs in switch-case
– Debugging becomes difficult.	– Tracing of errors and debugging is easy.
– The test expression can be a constant value or may involve any type of operators.	– Only constant integer expressions and values are allowed.

3.6 UNCONDITIONAL BRANCHING STATEMENT: GOTO

It is used to branch unconditionally from one point to another in the program. The **goto** requires a label in order to identify the place where the branch is to be made. A label is any valid variable name, and must be followed by colon (:). the general form of **goto** statement is:

```
goto label;
.....
.....
.....
label:
Statement;
```

Forward jump

```
label:
Statement;
.....
.....
.....
goto label;
```

Backward jump

The **label:** can be anywhere in the program either before or after the **goto label;** statement.

Example:

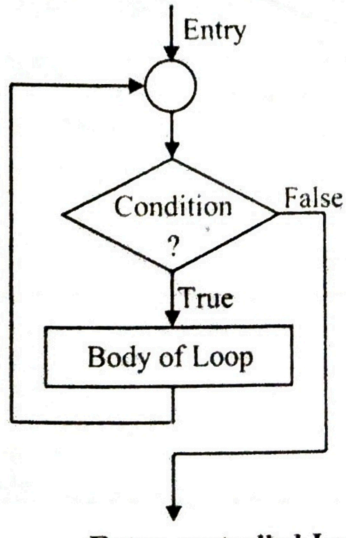
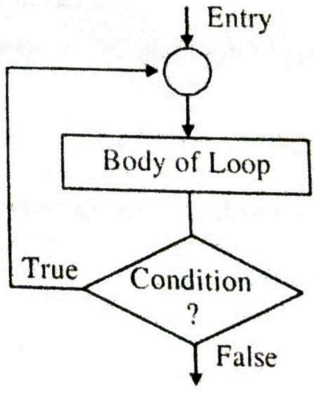
```
#include<stdio.h>
void main()
{
int x;
read:
scanf("%d",&x);
if(x<0)
goto read;
printf("%d\n",x);
}
```

In above program, if we enter value of x=0, then goto statement will be executed. Until we enter non-zero value of x, goto statement will be executed.

3.7 CONTROL STATEMENTS / LOOPING STATEMENTS

Loop control statements in C are used to perform looping operations until the given condition is true. Control comes out of the loop statements once condition becomes false.

There are two types of loop structure: Entry-controlled Loop and Exit-controlled Loop.

Entry-controlled Loop	Exit-controlled Loop
– It is also known as Pre-test Loop. – In the Entry-controlled Loop, the conditions are tested before the start of the loop execution. If the conditions are not satisfied, then the body of the loop will not be executed. – For Example: **while loop** and **for loop**. –  **Entry-controlled Loop**	– It is also known as Post-test Loop. – In the Exit-controlled Loop, the conditions are tested at the end of the body of the loop and therefore the body is executed unconditionally for the first time. – For Example: **do....while loop**. –  **Exit-controlled Loop**

C language provides three looping structures:

1. while Loop
2. do....while Loop
3. for Loop

3.8 FOR LOOP

It is an entry-controlled loop statement. The General form of for loop is:

```
for (initialization ; condition ; increment/decrement)
{
body;
}
```

Here, **initialization** of the control variable is done first. Then the **condition** is evaluated and if it is true, then the **body** of the loop is executed. After executing **body**, **increment/decrement** of variable is done and the **condition** is evaluated again and if it is true the **body** is executed again. This process of repeated execution of the **body** continues until the **condition** becomes false.

Example:

```
#include<stdio.h>
void main()
{
int i;
for(i=1 ; i<=5 ; i++)
{
i = i + 1;
}
printf("%d",i);
}
```

Output:

```
6
```

In above example the body of for loop is executed 5 times for i = 1, 2, 3, 4, 5.

3.9 WHILE LOOP

It is an entry-controlled loop statement. The General form of while loop is:

```
while (condition)
{
body;
}
```

Here, **condition** is evaluated first and if it is true, then the statements in the **body** of the loop are executed. After executing **body**, the **condition** is evaluated again and if it is true the **body** is executed again. This process of repeated execution of the **body** continues until the **condition** becomes false.

Example:

```
#include<stdio.h>
void main()
{
int i=1;
while(i<=5)
{
i = i + 1;
}
printf("%d",i);
}
```

Output:

```
6
```

In above example the body of while loop is executed 5 times for i = 1, 2, 3, 4, 5.

3.10 DO....WHILE LOOP

It is an exit-controlled loop statement. The General form of do....while loop is:

```
do
{
body;
}
while (condition);
```

Here, first **body** is executed and then **condition** is checked. If the **condition** is true, then the **body** is executed again. This process of repeated execution of the **body** continues until the **condition** becomes false. This ensures that **body** is executed at least once even if the **condition** is false first time.

Example:

```
#include<stdio.h>
void main()
{
int i=1;
do
{
i = i + 1;
}
while(i<=5);
printf("%d",i);
}
```

Output:

```
6
```

In above example the body of do....while loop is executed 5 times for i = 1, 2, 3, 4, 5.

■ DIFFERENCE BETWEEN WHILE AND DO...WHILE

While	Do While
While loop is entry controlled loop i.e.The test condition is evaluated first and if the condition is true then the body of the loop is executed.	Do while loop is exit controlled loop i.e. first body of the loop is executed and then condition is checked
While loop will not execute any statement if the condition is false	Do While loop executes at least once

e.g.
void main()
{ int i=11;
while (i<=10)
{
printf("%d", i);
}
}
Output: Blank Screen
	e.g.
	void main()
	{ int i=11;
	do
	{
	printf("%d",i);
	}while (i<=10);
	}
	Output : 11

Semi colon(;) not required after while	Semi colon(;) required after while

3.11 NESTED FOR LOOP

C supports nesting of loops in C. Nesting of loops is the feature in C that allows the looping of statements inside another loop. Let's observe an example of nesting loops in C.

The nested for loop means for loop which is defined inside the 'for' loop. Syntax:

```
for (initialization; condition; update)
{
    for(initialization; condition; update)
    {
        // inner loop statements.
    }
        // outer loop statements.
}
```

Example of nested for loop

```
#include <stdio.h>
int main()
{
   for (int i=0; i<2; i++)
   {
    for (int j=0; j<4; j++)
    {
       printf("%d, %d\n",i ,j);
    }
   }
   return 0;
}
```

Output:

```
0, 0
0, 1
0, 2
0, 3
1, 0
1, 1
1, 2
1, 3
```

In the above example we have a for loop inside another for loop, this is called nesting of loops.

3.12 BREAK AND CONTINUE STATEMENT

Break

When **break** statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. When the loops are nested, the **break** would only exit from the loop containing it. That is, the **break** will exit only a single loop.