UNIT - II
Decision control statements

When program executes, it gets executed line after line. But when some part of the program need to be executed based on conditions, then decision control statements are used.

In C, there are two decision control statements.

If-else branching

The if-else branching is used to divide the program into number of branches and execute them based on conditions. It can be used in following ways based on requirement.

When a single statement needs to be executed under any part, then it can be placed directly under either if or else. But when more than one statement is to be placed under any part, then all the statements must be grouped using { }.

1. Simple if statement : if the condition is true, then the statement under if part is executed.

```
if ( condition )
    statement;
```

2. If-else statement: if the condition is true, the statement under if is executed and if the condition results false, statement under else gets executed

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

3. Nested if: Placing an if within another is called nested if.

```
if(condition)
{
        if( condition )
                statement;
}
```

In this case, if the first if results true, then only the second if gets checked.

4. If-else-if: the else do not take condition. But if the code needs a condition to be placed with else, then an if can be placed along with else, to develop a chain called if-else-if ladder.

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

When the program needs to be divided into multiple branches and only one executed amongst them, then the if-else-if ladder is used.

Switch-case statement

For decision making by comparing equality of values, the switch-case statement can be used. The switch keyword takes a variable and under switch there can be number of cases specified within a block.

The switch block executes from matching case till **end of switch block**. The switch block can also have a "default" option that executes if no matching case exists.

```
switch(var)
{
    case 1: statements;
    case 2: statements;
    …….
    …….
    case n: statements;
    default : statements;
}
```

For controlled execution we may use the "break" keyword that breaks the switch block and passes control to rest of program.

The switch case in C works with ASCII code. So the "int" or "char" data types can only be used in switch-case. The float, double cannot be used for comparing. So in C, the switch supports 256 cases. Later in C++ it supports more than 65000 cases.

In any case, the switch does not support duplicate cases written within a block.

Ex:

```
char ch;
printf("Enter a character:");
scanf("%c",&ch);
switch(ch)
{
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':     printf("VOWEL");  break;
        default :     printf("consonant");
}
```

Iterative statements

The loops are used for iteration. Iteration is repeated execution of process. The loops are controlled with a Boolean expression. If the Boolean expression results true, the loops continue its execution and otherwise stops.

Based on placement of the condition, the loops are classified into two types. i) entry controlled loops, ii) exit controlled loops. The loops that have Boolean expression at begining of loop body are called entry controlled loops and that have at end of loop body are called exit controlled loops.

While statement:

The while keyword takes a Boolean expression (condition) and till the expression results true, the loop continues its execution. The variable used in Boolean expression must be initialized prior its use and we must use appropriate counter in the loop body such that the Boolean expression becomes false at a state. Otherwise it turns into an infinite loop.

Syntax :

```
Initialization;
While(boolen expression)
{
        Statements;
        Counter;
}
```

We can directly place the Boolean value true instead of Boolean expression to make it as infinite loop. The while statement is an entry controlled loop.

Do-while statement: the do statement takes a while at end of loop, and it is called a do-while loop. The difference between the while and do-while is that the Boolean expression is placed at beginning of while and in do-while the Boolean expression is placed at end. So it is called an exit controlled loop.

When we use the do-while, the loop body is executed at least once even if the Boolean expression results false.

Syntax :

```
Initialization;
do{
        statements;
        counter;
}while(Boolean expression) ;
```

for statement:

The for is entry controlled loop and it takes the Boolean expression at beginning of loop body.

Syntax :

```
Initialization;
for(  ; Boolean expression;  )
{
        Statements;
        Counter;
}
```

As in for loop, the program control comes to the location that is before first **;** we can logically shift initialization to that location and as every time the

program control comes to the location that is after second **;** we can logically shift the counter to that location, thus forming logical syntax of for as follows:

```
for(initialization ; condition ; counter)
    Statement;
```

Nested loops

When a loop is placed with in another loop, it is called nested loops. The inner loop executes no. of times for each run of outer loops. The count of total number of executions of statements is equal to the number of executions of outer loop multiplied by number of number of executions of inner loop.

Jump Statements:

continue: is a keyword when executes skips rest of loop once and passes control back to the loop.

Ex:

```
for(i=0;i<20;i++)
{
    if(i%3==0)
        Continue;
    printf("%d ",i);
}
```

The above code displays numbers between 0 and 20 by skipping multiples of 3.

break: is a keyword when executes, breaks the loop and passes control to rest of the program.

```
for(i=0;i<10;i++)
{
    if(i==5)
        break;
    printf("%d ",i);
}
```

Will display 0 1 2 3 4

goto: is keyword , used to make jumps in the program. Fr making jumps in the program, the location to which the program execution is to be jumped must be labeled and the same label is to be specified to the goto keyword. The goto makes program execution go to the location whose label is specified.

But one of the standards of the structured programming languages is "sequential execution of program". With the goto statement, this standard is broken. So it is suggested to programmers not to user goto statements in program.

The programmers are to develop a controlled logic with goto statement, such that it should not develop infinite occurrence.

Ex:

```
void main()
{
        goto LABEL2;
        LABEL1:
                printf("@ label 1");
                goto LABEL3;
        LABEL2:
                printf("@ label 2");
                goto LABEL1;
        LABEL3:
                printf("completed");
}
```

Q. here question may be framed on branching separately or on switch case separately or on iteration separately or on break and continue statements each for 5 marks.

Or

There can be a question asking for controlling statements for 10 marks, then write about branching and loops. (without break, continue)