UNIT - 2

DECISION STATEMENTS AND FUNCTIONS

DECISION MAKING & BRANCHING STATEMENTS:

Decision-making statements in a programming language help the programmer to transfer the control from one part to other parts of the program. Thus, these decision-making statements facilitate the programmer in determining the flow of control. This involves a decision-making condition to see whether a particular condition is satisfied or not. On the basis of real-time application it is essential:

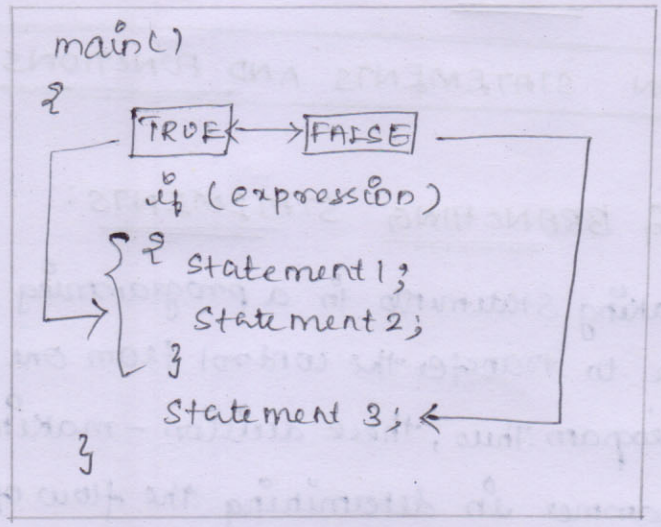
i) To alter the flow of a program.
ii) To test the logical conditions.
iii) To control the flow of execution as per the selection.

These conditions can be placed in the program using decision-making statements. C language supports the control statements as listed below.

i) The if-statements.
ii) The if-else statement.
iii) The if-else-if ladder statement.
iv) The switch case statement.
v) The goto unconditional jump.
vi) The loop statement.

THE if statement:

C uses the keyword if to execute a set of command lines or one command line when the logical condition is true.



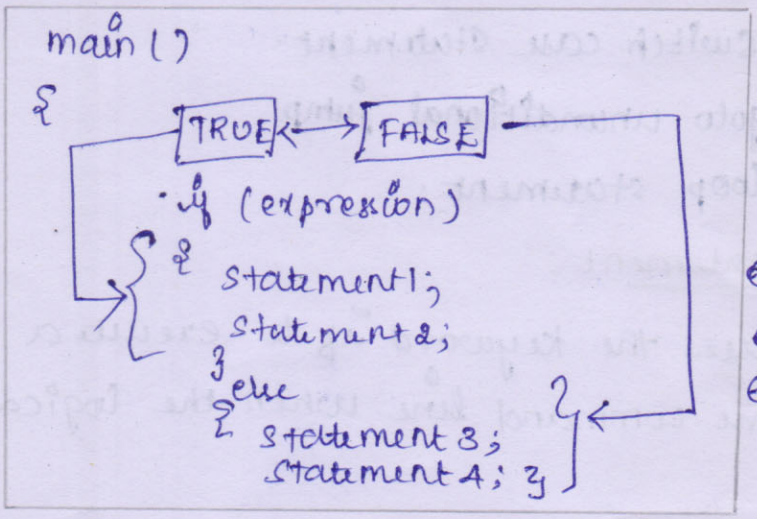
Syntax:

if (condition)
statement;

Program: Voting Eligibility:

```
Void main()
{
    int age;
    clrscr();
    Printf("\n Enter age: ");
    Scanf("%d", &age);
    if (age > 17)
    Printf("\n Eligible for voting: ");
     getch();
}
```

If.. else statement:



Syntax:

if (the condition is true)
Execute the statement 1;
else
Execute the statement 2;

Program: calculate square of numbers whose least significant digit is 5.

```
void main()
{  int s,d;
   clrscr();
   Printf("\n Enter the number: ");
   Scanf["%d",&s);
   d=s%10;
   if(d==5)
   {  s=s/10;
      Printf("\n Square = %d%d", s*s+1, d*d);
   }else
      Printf("\n Invalid Number");
   }
```

Output: Enter the number : 25
Square = 625

Nested if - else statement :

Syntax:

```
if(condition)
{ /* Inside first if block */
  if (condition)
   { statement 1; /* if block */
     Statement 2;
   }
else
  { statement 3;
```

```
    Statement 4; /* else block */
  }}
  else
   { /* Inside else block */
    if (condition)
     { statement 5;
       statement 6;
     }
    else
      { statement 7; /* else block */
        statement 8;
      }}
```

Program: To find the largest number out of three numbers:

```
void main()
{  int x,y,z;
   clrscr();
   Printf("\n Enter Three Numbers x,y,z:");
   Scanf("%d %od %od", &x, &y, &z);
   Printf("\n Largest out of Three Numbers is:");
   if (x>y)
   {      if(x>z)
            Printf("x=%d \n", x);
            else
            Printf("z=%od \n", z);
     } else
     {   if(z>y)
          Printf("z=%od \n", z);
          else
          Printf("y=%od \n", y);
      }}
```

The break statement:

The keyword break allows the Programmer to terminate the loop.

Difference between break and exit()

S.NO	break	exit()
1.	It is a keyword	It is a function.
2.	No header file is needed.	Header file Process.h must be included.
3.	It stops the execution of the loop	It terminates the Program.

The continue Statement:

The continue statement used for continuing the next iteration of the loop statements.

Difference between break and continue:

S.NO	break	continue.
1.	Exits from current block or loop	Loop takes the next iteration.
2.	Control passes to the next statement	Control passes at the beginning of the loop.
3.	Terminates the loop.	Never terminates the program.

The goto statement:

This statement doesnot require any condition. This is an unconditional control jump. This statement passes control anywhere in the program.

Syntax: goto label;

The Switch Statement:

The switch statement is a multi-way branch statement. In the program if there is a possibility to make a choice from a number of options, this structured selection is useful.

NESTED Switch CASE:

The C supports nested switch statements. The inner switch can be part of an outer switch. The inner and outer switch case constants may be the same. No conflicts arise even if they are the same.

Example:

```
Void main()
{  int x, y;
   clrscr();
   Printf("\n Enter a Number: ");
    Scanf("%d", &x);
    Switch(x)
   { case 0:
     Printf("\n Number is Even");
      break;
     Printf("\n Number is odd");
      break;
      default:
       y = x%2;
      switch(y)
      {
          case 0:
          Printf("\n Number is Even");
            break;
            default:
             Printf("\n Number is odd");
      }  }
   getche();
}.
```

Switch case	nested if's
* The switch can only test for equality, i.e. only constant values are applicable. * No two case statements have identical constants in the same switch * character constants are automatically converted to integers. * In switch case statement nested if can be used.	* The if can evaluate logical or relational expressions. * Same conditions may be repeated for the numbe of time. * Characte constants are automatically converted to integers. * In nested if statement switch case can be used.

LOOPING (BRANCHING) STATEMENTS:

Loop:

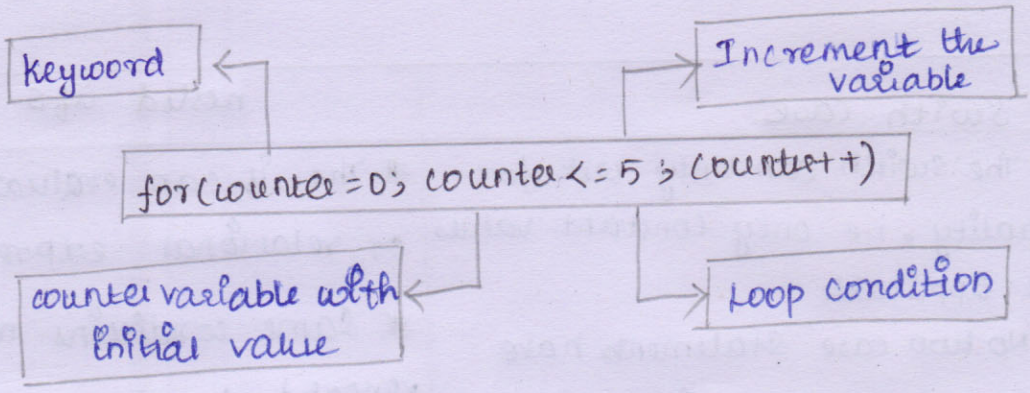
A loop is defined as a block of statements, which are repeatedly executed for a certain number of times.

For Loop:

for loop allows to execute a set of instructions until a certain condition is satisfied. conditions may be predefined or open-ended.

Syntax:

```
for(initialize counter; test condition; re-evaluation parameter)
{
    Statement 1;
    Statement 2;
}
```



Example:

```
void main()
{ int i;
  clrscr();
  for(i=1;i<=15;i++)
   Printf("%5d",i);
   getche();
}
```

Nested for loops:

In nested for loops one or more for statements are included in the body of the loop.

Example:

```
void main()
{ int i,j;
  clrscr();
  for(i=1;i<=3;i++)  /* outer loop */
  {
    for(j=1;j<=2;j++) /* inner loop*/
     Printf("\n i*j : %d", i*j);
  }
}
```

while loop:

The while loop is frequently used in programs for the repeated execution of statement in a loop. until certain condition is satisfied the loop statements are executed.

Syntax:

```
while(test condition)
{
   body of the loop
}
```

Example: Factorial

```
void main()
{
  int a, fact=1;
  clrscr();
  Printf("\n Enter the number:");
  Scanf("%d", &a);
  while(a>=1)
  {
     Printf("%od *", a);
     fact=fact*a;
     a=a-;
  }
  Printf("=%d", fact);
  Printf("\n Factorial of given number is %od", fact);
}
```

output:

```
Enter the number: 5
5* 4 * 3 * 2 * 1 = 120
Factorial of given umber is 120
```

do-while loop:

The difference between the while and do-while loop is the place where the condition is to be tested. In do-while loop condition is checked at the end.

Syntax:

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

S.NO	while loop	do-while loop
1.	condition is specified at the top.	condition is mentioned at the bottom.
2.	Body statements are executed when condition is satisfied	Body statements executes even when the condition is false.
3.	No brackets for a single statement.	Brackets are essential even when a single statement exists.
4.	It is an entry-controlled loop	It is an exit-controlled loop.

Example:

```
void main()
{ int i=1;
  clrscr();
  do
  {
    Printf("\n This is a program of
                do while loop.");
      i++;
  }
while(i<=5);
}
```

FUNCTION:

A function is a self-contained block or sub-program of one or more statements that perform a special task when called.

Why use function?

a) If we want to perform a task repetitively, then it is not necessary to re-write the particular block of the program again and again. Shift the particular block of statements in a user-defined function. The function defined can be called any number of times to perform the task.

b) using functions, large programs can be reduced to smaller ones. It is easy to debug and find out the error in it. It also increases readability.

Function definition:

Function definition as per the format given below.

```
return_type function_name (argument/parameter list)
{
    local variable declaration;
    Statement 1;
    Statement 2;
    return (value);
}
```

/* Body of the function definition */

Working of function:

```
int abc(int, int, int);
void main()
{
    int x, y, z;
    - - -
    - - -
    abc(x, y, z); /* Function call */
          ↳ Actual arguments
    - - - -
    - - - -
int abc(int i, int k, int j) Function definition
 {                              ↳ Formal arguments
    - - -
    - - -
    return(); return value
 }
```

Example:

```
int add(int, int); /* function prototype */
void main()
{
    int x=1, y=2, z;
    z=add(x, y);        /* Function call */
    Printf("z=%d", z);
}
/* Function definition */
int add(int a, int b)
{
    return (a+b);
}
```

Output : z=3

CALL BY VALUE AND REFERENCE:

There are two ways in which we can pass arguments to the functions.

i) call by value

ii) call by Reference.

i) call by value:

In this type, the value of actual argument is passed to the formal arguments and Operation is done on the formal arguments. Any change in the formal argument made does not affect the actual argument because formal arguments are the photocopy of the actual argument. Hence, when a function is called by the call by value method, it does not affect the actual contents of the arguments. changes made in the formal arguments are local to the block of the called function. Once control return back to the calling function, changes made vanish. The example given below illustrates the use of call by value.

Example:

```
int main()
{
  int x, y, change (int, int);
  clrscr();
  Printf("In Enter values of x & y: ");
  Scanf("%od %od", &x, &y);
  change (x, y);
  Printf ("In In main() x = %d y = %d", x, y);
```

```
return 0;
}
change(int a, int b)
{   int k;
    k = a;
    a = b;
    b = k;
    Printf("\n In change() x=%d y=%d", a, b);
}
```

Output:

```
Enter value of x & y : 5  4
In change() x=4 y=5
In main() x=5 y=4
```

ii) Call by Reference:

In this type instead of passing values, addresses (reference) are passed. Function operates on addresses rather than values. Here, the formal arguments are pointers to the actual argument. In this type of, formal arguments point to the actual argument. Hence, changes made in the argument are permanent. The example given below illustrates the use of call by value.

Example:

```
int main()
{
    int x, y, change(int *, int *);
    clrscr();
```

```
Printf("\n Enter values of x & y: ");
Scanf("%d %d", &x,&y );
 change (&x ,&y);
 Printf("\n In main() x=%d  y=%d", x , y);
  return 0;
}
change (int *a, int *b)
{
  int *k;
  *k = *a;
  *a = *b;
  *b = *k;
  Printf("\n In change() x=%d  y=%d", *a ,*b);
}
```

Output:

```
Enter values of x & y :  5  4
In change()  x=4  y=5
In main()  x=4  y=5
```