Page No.
Date :

UNIT 3 CONTROL FLOW

* DECISION MAKING & BRANCHING :

- Used for decision making purposes.
- Used to evaluate one or more conditions & make the decision whether to execute a ~~state~~ set of statement or not.

* TYPES OF CONDITIONAL STATEMENT:

1. If Statement :-

- If expression is true then it executes the statement otherwise jumps to next instruction.
- It checks whether given expression is boolean or not.

- SYNTAX

```
if (expression)
{
  Statement 1;
  Statement 2;
}
```

- Eg:

```
void main ()
{
  int a=5, b=6, c;

  c = a+b;

if (c == 11)
{ printf ("Execute me 1");
  }
```

```
printf ("Execute me 2");
}
```

* OUTPUT:-
Execute me 1.

```
# include <stdio.h>
  main ()
{
  int a=15, b=20;
  if (b>a)
{
  printf ("b is greater");
}
  }
```

· OUTPUT
b is greater.

Page No.
Date :

* If - else STATEMENT ::

- It is used so that we can check any condition & depending on the outcome of any condition we can follow appropriate path.

- SYNTAX :

```
if (expression)
{
    statement 1 ;
    statement 2 ;
}
else
{
    statement 3 ;
    statement 4 ;
}
next - statement ;
```

- If expression is true then statement 1 & statement 2 are executed otherwise statement 3 & statement 4 are executed

```
void main ( )
{
    int marks = 50 ;
    if (marks >= 40)
{
    printf ("student is pass") ;
    }
else
    {
        printf ("student is fail") ;
```

```
    }
  }
```

- OUTPUT:
student is pass.

* PROGRAM TO FIND ODD OR EVEN NUMBER:

```
#include <stdio.h>
int main()
{
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number);

    if (number % 2 == 0)
    {   printf("%d is an even integer.",
                number);

    }
    else
    {  printf("%d is an odd integer.",
                number);

    }
    return 0;
}
```

* OUTPUT
- Enter an integer : 7
7 is an odd integer.

Page No.
Date :

* If-else if STATEMENT :-

- It is used to execute one code from multiple condition.

- SYNTAX :-

```
if (condition 1)
{   statement-1 ;
}
else if (condition 2)
{   statement-2 ;
}
else if (condition 3)
{   Statement-3 ;
}
else if (condition n)
{   statement n ;
}
else
{ default statement
}
statement-x.
```

Eg :-

```
# include <stdio.h>
# include <conio.h>
  void main ()
{
    int number = 0 ;
    clrscr ();
    printf (" Enter a number : ") ;
    scanf (" %d ", & number) ;
```

```
if (number == 10) {
printf (" number is equal to 10");
}
else if (number == 50)
{
printf ("number is equal to 50");
}
else if (number == 100):
{
printf ("number is equal to 100");
}
else {
printf ("number is not equal to 10, 50 or 100");
}
getch ();
}
```

Eg:

```
# include <stdio.h>
int main ()
{ int i = 20;
if (i == 30)
{ printf ("i is 30"); }
else if (i == 15)
{ printf ("i is 15"); }
else if (i == 20).
{ printf ("i is 20"); }
else
{ printf ("i is not present"); }
}
```

Page No.
Date:

* SWITCH STATEMENT :.

- When there are several options & we have to choose only one option from the available ones, we can use switch statement.

- SYNTAX :-

```
switch (expression)
{
   case value 1 :
            statement / block-1 ;
            break;
   case value 2 :
            statement / block-2 ;
            break;
   case value 3 :
            statement / block-3 ;
            break;
   default :
            statement / block;
            break;
}
```

- The Expression following the keyword switch must yield an integer value. It must be integer constants like 1, 2, 3.
- The keyword 'case' is followed by integer or character constant, each constant in each must be different from all the other.

- First the integer expression following the keyword switch is evaluated. The value

it gives is searched against the constant values that follow the case statement. When a match is found the program executes the statement following the case.. If no match is found with any of the case statements, then statement following the default are executed.

Eg:

```
#include <stdio.h>
main()
{
   int a;
printf("plz enter a no. betn 1 & 5 : ");
scanf("%d", &a);
switch (a)
{
 case 1 :
 printf("you choose one");
 break;
 case 2 :
 printf("you chose two");
 break;
  ;
  ;
 case 5 :
 printf("you chose five");
 break;
 default :
 printf("Invalid choice. Enter a no. betn
          1 & 5");
 break;
 }
 }
```

Page No.
Date:

* OUTPUT

- please enter a no. b/w 1 4 5 : 3
- you choose Three

Page No.
Date :

* Loops :-

- Suppose that you have to print table of 2, then you need to write 10 lines of code. By using Loop statement, you can do it by 2 or 3 lines of code only.

* WHILE LOOP :-

- SYNTAX :-

```
Variable initialization ;        / Eg :- int x = 0
While (condition)                / while (x <= 10)
{
    Statement ;
    Variable increement or decreement ;
                                 / Eg :- x++ or x-- or x = x+2
}
```

- It is an entry level loop statement i.e the condition is evaluated first & if it is true then body of loop is executed.

- After executing the body of the loop, the Condition is once again evaluated & if it is true, the body is executed once again. The process of repeated execution of loop continues until the condition finally becomes false & control is transferred out of the loop.

Eg :-

```
# include <stdio.h>
# include <conio.h>
  void main ()
    int x ;
    x = 1 ;
```

Page No.
Date :

```
while (x <= 10)
{
  printf ("%d \t", x);
  x++;
}
getch ();
```

- OUTPUT

1 2 3 4 5 6 7 8 9 10.

Eg:- PROGRAM TO REVERSE THE NUMBER :.

```
# include <stdio.h>
  int main ()
{
    int n, reverse = 0, remainder;
    printf ("Enter an integer : ");
    scanf ("%d", &n);

    while (n != 0)
{
    remainder = n % 10.;
    reverse = reverse * 10 + remainder;
    n /= 10;

}
  print f ("Reversed number = %d", reverse);
    return 0;
  }
```

Page No.
Date :

• EXPLAINATION :-

Suppose we want reverse of 26

	n	n ! = 0	remainder (n % 10)	reverse [rev * 10 + rem]
1.	26	True	26 % 10 R = 6	0 x 10 + 6 = 6
2.	n != 10 = 26/10 = 2	True	2 % 10 R = 2	6 x 10 + 2 = 62

Page No.
Date:

* DO-WHILE LOOP:

- SYNTAX

```
variable initialization ;
do {
statements ;
variable increement or decreement;
}
while (condition) ;
```

- Do-while loop is an exit controlled loop statement.
- Body of loop is executed first & then the condition is evaluated. If it is true then the body of loop is executed once again.
- The process of execution of body of loop is continued until the condition finally becomes false & control is transferred to the statement immediately after the loop.

Eg:- program to print ten multiples of 5

```
# include <stdio.h>
# include <conio.h>
void main ()
{ int a, i ;
  a = 5 ;
  i = 1 ;
  do.
{ printf ("%d \t", a * i) ;
  i ++ ;
}
while (i <= 10) ;
getch () ;
}
```

OUTPUT :-
5, 10, 15, 20, 25, 30, 35, 40, 45, 50

Page No.
Date :

* FOR LOOP :-

- SYNTAX :-

```
for (initialization; Condn; incrmnt/decrmnt)
{
    Statements ;
}
```

- It is an entry controlled looping statement.
- In this loop, more than one variable can be initialised.
- Three actions can be taken at a time like variable initialization, Condn checking & increement/decrement.
- It is more concise & flexible than other loops.

Eg:-

```
# include <stdio.h>
# include <conio.h>
void main ()
{
    int x ;
    for (x=1 ; x<=10 ; x++)
    {
        printf ("%d\t", x);
    }
    getch ();
}
```

OUTPUT

1 2 3 4 5 6 7 8 9 10

Page No.
Date:

∴ program to write multiples of any number

```
#include<stdio.h>
int main()
{
    int i = 1, number = 0;
    printf("Enter a number: ");
    scanf("%d", &number);

for (i = 1; i <= 10; i++)
 {
     printf("%d \n", (number * i));
 }
 return 0;
}
```

∴ OUTPUT

```
Enter a number: 2
2
4
6
8
10
12
14
16
18
20
```

* GO TO STATEMENT :

- It allows us to transfer control of program to the specified identifier.

- SYNTAX :

```
goto label 1 ;
... ...
label 1 :
Statement ;
```

- When the goto statement is encountered the control of the program jumps to label 1 : (identifier) & starts executing the code.

Eg:-

```
# include <stdio.h>
int main ()
{
    int startvalue = 1 , endvalue = 10 ;
    int current = startvalue ;

    print_line :    // Define the label.

    printf ("%d", current) ;

    if (current < endvalue)
    {
        current ++ ;
        goto print_line ;
    }
return 0 ;
```

Page No.
Date :

OUTPUT

1 2 3 4 5 6 7 8 9 10

* BREAK & CONTINUE :-

* CONTINUE :.

- It is used to skip the current iteration of any loop & bring the control to the beginning of iteration.
- The statements following the continue statement are skipped & iteration starts again.

Eg:-

```
# include <stdio.h>
  void main ( )
{
   for (int i=1; i<=10; i++)

{    if (i==5)

     {    continue ;

     }

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

- OUTPUT

1
2
3
4
6
7
8
9
10

Page No.
Date:

* BREAK :

- It is one of the most used jump statement.
- In C, break is used to prematurely exit from a loop or switch statement, before the block has been fully executed.

- As soon as break statement is encountered inside a loop, loop terminates immediately & control is transferred to next statement outside the loop.

Eg:

```
# include <stdio.h>
  int main ()
{
   for (int i=1 ; i<=10 ; i++)
  {
      if (i==5)
   {
      break;        // terminates loop when i=5

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

}
return 0;
}
```

OUTPUT

```
1
2
3
4
```