

## Module 2

### *CO2 – Understand Different Control Structures and Solve Problems*

*Conditional Control Structures – if, if – else, if – else ladder, nested if, switch -case, goto, conditional operator, Relational and logical operators.*

*Different looping statements – while, do-while and for loops, Counter-controlled and Sentinel controlled loops, break and continue statements.*

## Control Statements

- Control statements control the flow of execution of the statements of a program.
- C supports three types of control statements.

They are :

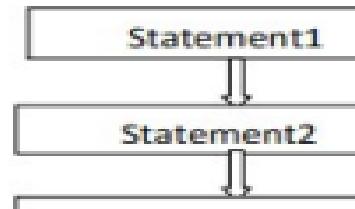
1. *Sequential Control Statements*
2. *Conditional Control Statements*
3. *Iteration or Looping Statements*

### Sequential Control Statements

- It ensures that the instructions are executed in the same order in which they appear in the program.
- The program statements are executed sequentially i.e., one after the another from beginning to end.

Eg:

```
int main()
{
    clrscr();
    int x , y, sum;
    printf("enter the two numbers");
    scanf("%d%d",&x,&y);
    sum=x+y;
    printf("the sum of the two numbers is =%d",sum);
    return 0;
}
```



### Decision Control statements

- ❖ Also called Conditional Control or Selection Control Statements.
- ❖ These statements change the flow of execution depends on a condition-test.
- ❖ If the condition is true, then a set of statements is executed otherwise another set of statements is executed.
- ❖ It helps in making decision about which set of statements is to be executed.
- ❖ Used to execute or skip some set of instructions based on given condition.

- ❖ C supports six types of control statements

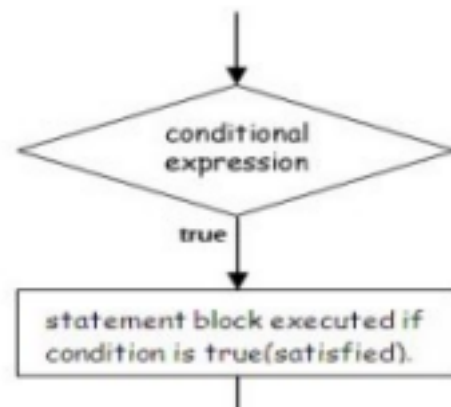
1. *if statement*
2. *if-else statement*
3. *Nested if else statement*
4. *else-if ladder*
5. *case control structure*
6. *conditional operators*

## if Statement

- This is the simplest form of decision control statement.
- If statement is used to test a condition, if condition is true then the code inside the if statement is executed otherwise that code is not executed.

### Syntax:

```
if(test_expression)
{
statement 1;
statement 2;
...
}
```



### Example : Program to check whether the given number is negative.

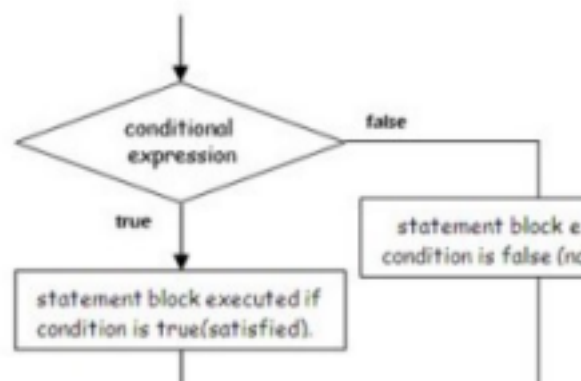
```
int main()
{ int num;
  clrscr();
  printf("enter the number");
  scanf("%d",&num);
  if(num<0)
  { printf("the entered number is negative"); }
  return 0;
}
```

## if- else Statement

- This is a bi-directional control statement.
- This statement is used to test a condition, if the condition is true then one statement (block of statements) is executed otherwise other statement (or block of statements) is executed.

### Syntax:

```
if(test_expression)
{
Body of if Statement;
}
else
```



```
{  
    Body of else Statement;  
}
```

### Example: Program to find the biggest of two numbers

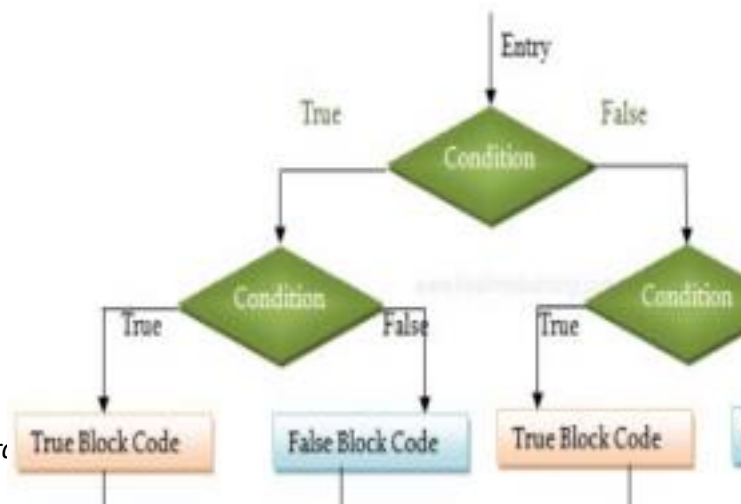
```
int main()  
{  
    int num1, num2;  
    clrscr();  
    printf("enter the two numbers");  
    scanf("%d %d", &num1, &num2);  
    if (num1 > num2)  
    {  
        printf("the number %d is big", num1);  
    }  
    else  
    {  
        printf("the number %d is big", num2);  
    }  
    return 0;  
}
```

### Nested if Statement (Nested if – else Statement)

- It contains multiple if else condition.
- It is used to check the multiple conditions.
- This statement is like executing an if statement inside an else statement.
- In this, an if-else statement within either the body of an if statement or the body of else statement or in the body of both if and else, then this is known as nesting of if else statement.

### Syntax:

```
if(test_expression one)  
{  
    if(test_expression two) {  
        //Statement block Executes when the boolean test expression two is true.    }  
    }  
else  
{  
    //else statement  
    block  
}
```



### Example: Program to demonstrate Nested - if

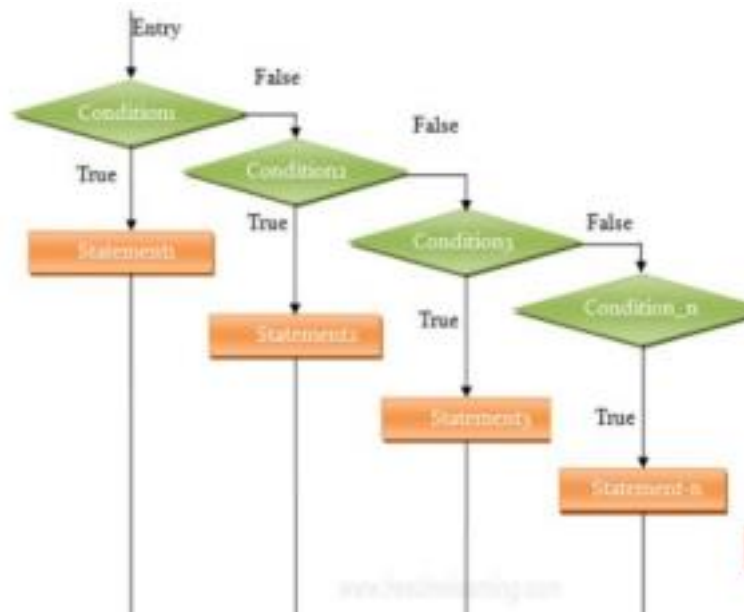
```
void main()
{
    int x = 20, y = 30;
    if (x==20)
    {
        if (y==30)
        {
            printf("value of x is 20, and value of y is 30.");
        }
    }
}
```

### else if Ladder

- After if statement else if is used to check the multiple conditions.
- This is a type of nesting in which there is an if-else statement in every else part except the last else part.
- This type of nesting is called else if ladder.

### Syntax:

```
if(test_expression)
{
    //execute your code
}
else if(test_expression
n)
{
    //execute your code
}
else
{
    //execute your code
}
```



### Example – Program to find the Grade of a student

```
int main()
{
    int marks;
    printf("enter the percentage of the student");
    scanf("%d",&marks);
```

```
if(marks>=80)
    printf("your grade is a");
else if(marks>=70)
    printf("your grade is b");
else if(marks>=60)
    printf("your grade is c");

else if(marks>=50)
    printf("your grade is d");
else printf("you are fail");
return 0;
}
```

### Switch case statement

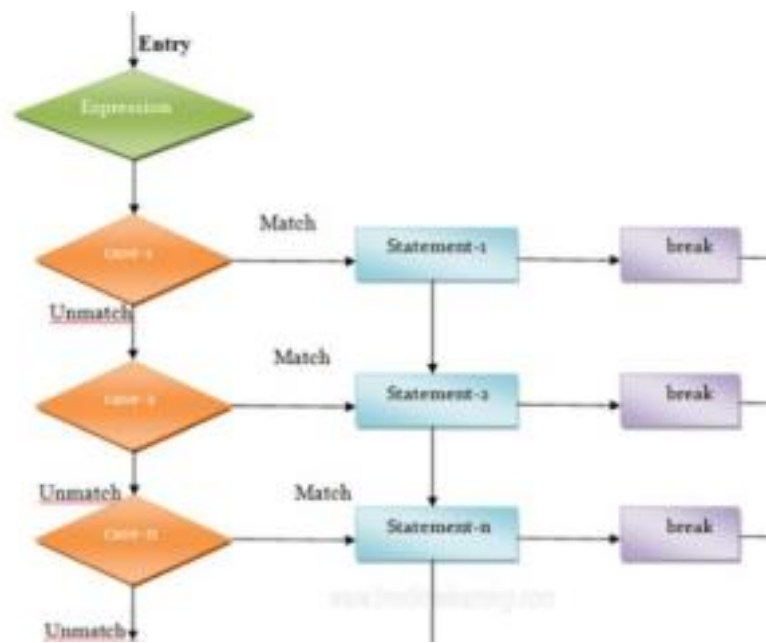
- It is an alternative to the if-else-if ladder.
- Switch is another conditional control statement used to select one option from several options based on given expression value.
- The selection is based upon the current value of an expression which is included with in the switch statement.
- The switch statement evaluates expression and then looks for its value among the case constants.
- If the value matches with case constant, then that particular case statement is executed. □ If no one case constant not matched then default is executed.
- Every case statement terminates with colon ":".
- In switch each case block should end with break statement, i.e.

#### Syntax:

```
switch(variable)
{
case 1:
//execute your
code
break;

case n:
//execute your
code
break;

default:
//execute your
code
break;
}
```



Example : Program to display the traffic control signal lights.

```
int main( )
{
    char L;
    printf("\n Enter your Choice( R,r,G,g,Y,y):");
    scanf("%c", &L);
    switch(L)
    {
        case "R" :
        case "r": printf("RED Light Please STOP");
        break;
        case "Y" :
        case "y": printf("YELLOW Light Please Check and Go");
        break;
        case "G":
        case "g": printf("GREEN Light Please GO");
        break;
        default: printf("THERE IS NO SIGNAL POINT ");
    }
    return 0;
}
```

### **Conditional Operator**

- It is a ternary operator.
- It is used to perform simple conditional operations.
- Conditional operator is used to check a condition and Select a Value depending on the Value of the condition.
- It is used to do operations similar to if-else statement.

Syntax:

Test expression? Value 1 : Value 2

Example : To find the largest of two numbers.

```
int main()
{
    float num1, num2, max;

    printf("Enter two numbers: ");
    scanf("%f %f", &num1, &num2);
    max = (num1 > num2) ? num1 : num2;
    printf("Maximum of %.2f and %.2f = %.2f", num1, num2, max);
    return 0;
}
```

1. C program to check whether the number is divisible by 5 and 11.
2. Program to find the roots of a quadratic equation.
3. Program to check if the number is Armstrong or not.
4. C program to calculate an electricity using the following criterion.

## Iterations/Loop Control Statements

- A loop is used to repeat a block of code until the specified condition is met. □ Iterations or loops are used when we want to execute a statement or block of statements several times.
- The repetition of loops is controlled with the help of a test condition. □ The statements in the loop keep on executing repetitively until the test condition becomes false.
- C programming has three types of loops:
  - a) *for Loop*
  - b) *while Loop*
  - c) *do-while Loop*

### while Loop

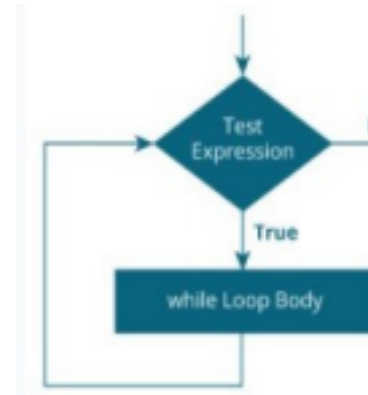
- It is an entry control loop.
- Repeats a statement or group of statements while a given condition is true.
- It tests the condition before executing the loop body.

#### Syntax

```
while (testExpression) {  
    // the body of the loop  
}
```

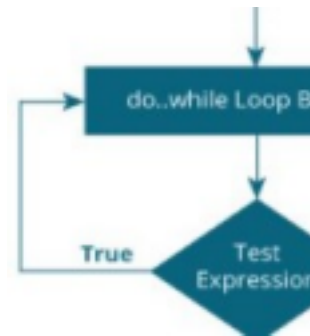
#### Example: To print first 10 natural numbers

```
int main( )  
{ int x = 1;  
    while(x <= 10)  
    {  
        printf("%d\t", x);  
        x++;  
    }  
}
```



### do – while Loop

- It is an exit control loop.
- A do...while loop is similar to a while loop.
- It executes the body of loop at least once.
- If the condition is true, the body of the loop is executed again and the condition is evaluated once more.
- This process goes on until the condition becomes false.
- If the condition is false, the loop ends.



### Syntax

```
do {  
    // the body of the loop  
}  
while (testExpression);
```

### Example: Program to add numbers until the user enters zero.

```
int main()  
{  
    int number, sum = 0;  
    // the body of the loop is executed at least once  
  
    do  
    {  
        printf("Enter a number: ");  
        scanf("%d", &number);  
        sum += number;  
    }  
    while(number != 0);  
    printf("Sum = %d",sum);  
    return 0;  
}
```

### **for Loop**

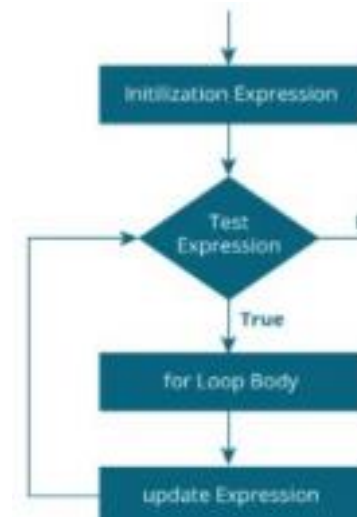
- It is an open ended loop..
- It is used to execute a set of statements repeatedly until a particular condition is satisfied.
- Executes a sequence of statements multiple times.

### Syntax:

```
for (initializationStatement; testExpression; updateStatement)  
{  
    // statements inside the body of loop  
}
```

- The initialization statement is executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.
- If the test expression is evaluated to true, statements inside the body of the for loop are executed, and the update expression is updated.
- Again the test expression is evaluated.
- This process goes on until the test expression is false. When the test expression is false, the loop terminates.





Example: Print numbers from 1 to 10

```
int main()
{
    int i;
    for (i = 1; i < 11; ++i)
    {
        printf("%d ", i);
    }
    return 0;
}
```

### Infinite Loop

- A loop becomes an infinite loop if a condition never becomes false.
- The for loop is traditionally used for this purpose.
- Three expressions are required to form the 'for' loop.
- But to provide two semicolons to validate the syntax of the for loop. This will work as an infinite for loop.
- When the conditional expression is absent, it is assumed to be true.

### Example

```
main ()
{
    for(;;)
    {
        printf("welcome to ");
    }
}
```

## Nested Loop

- Nested loop means a loop statement inside another loop statement. □ That is why nested loops are also called as “loop inside loop“.

### Syntax for Nested For loop:

```
for ( initialization; condition; increment )
{
    for ( initialization; condition; increment )
    {
        // statement of inside loop
    }
    // statement of outer loop
}
```

### Example

```
int main()
{
    int n; // variable declaration
    printf("Enter the value of n :");
    // Displaying the n tables.
    for(int i=1;i<=n;i++) // outer loop
    {
        for(int j=1;j<=10;j++) // inner loop
        {
            printf("%d\t",(i*j)); // printing the value.
        }
        printf("\n");
    }
}
```

### Output

Enter the value of n : 3

1 2 3 4 5 6 7 8 9 10 2 4 6 8 10 12 14 16 18 20 3 6 9 12 15 18 21 24 27  
30

## Unconditional Control Statements

- Control statements that do not need any condition to control the program execution flow.
- These control statements are called as unconditional control statements.
- C programming language provides three unconditional control statements.

1. *break*
2. *continue*
3. *goto*

### break Statement

- It is used to terminate the loop (or) exit from the block.
- The control jumps to next statement after the loop (or) block.
- break is used with for, while, do-while and switch statement.

- When break is used in nested loops then, only the innermost loop is terminated.

Syntax for break statement as :

**break;**

Example

```
int main( ){
    int i;
    for (i=1; i<=5; i++){
        printf ("%d", i);
        if (i==3)
            break;
    }
}
```

Syntax

```
{
    Stmt1;
    Stmt2;
    break;
    Stmt3;
    Stmt4;
}
```

continue Statement

- The continue statement is used to bring the program control to the beginning of the loop.
- The continue statement skips some lines of code inside the loop and continues with the next iteration.
- It is mainly used for a condition so that we can skip some code for a particular condition.

Syntax

```
//loop statements
continue;
//some lines of the code which is to be skipped
```

Syntax

```
{
    Stmt1;
    Stmt2;
    continue;
    Stmt3;
}
```

Example

```
int main(){
    int i=1; //initializing a local variable
    //starting a loop from 1 to 10
    for(i=1;i<=10;i++){
        if(i==5){ //if value of i is equal to 5, it will continue the loop
            continue;
        }
        printf("%d \n",i);
    } //end of for loop
    return 0;
}
```

goto Statement

- It is known as jump statement in C.
- goto is used to transfer the program control to a predefined label.
- The goto statment can be used to repeat some part of the code for a particular condition.
- Statement which is used to branch unconditionally within a program from one point to another.

## Syntax:

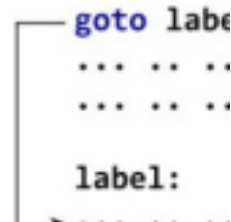
goto label;

-----

label:

statement - X;

/\* This the forward jump of goto statement \*/



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

The label is an identifier which is a valid variable name which is followed by a colon. When the goto statement is encountered, the control of the program jumps to label and starts executing the code.

Example - Program to calculate the sum and average of positive numbers. If the user enters a negative number, the sum and average are displayed.

```
int main() {
    const int maxInput = 100;
    int i;
    float number, average, sum = 0.0;

    for (i = 1; i <= maxInput; ++i) {
        printf("%d. Enter a number: ", i);
        scanf("%f", &number);

        // go to jump if the user enters a negative number
        if (number < 0.0) {
            goto jump;
        }
        sum += number;
    }
    jump:
    average = sum / (i - 1);
    printf("Sum = %.2f\n", sum);
    printf("Average = %.2f", average);

    return 0;
}
```

## Counter Controlled Loop

- It is also known as definite repetition loop.
- In this loop, the number of iterations is known before the loop begins to execute.
- The counter-controlled loop has the following components:
  - o a control variable.
  - o the increment (or decrement) value by which the control variable is modified at each iteration of the loop.
  - o loop terminating condition that checks if looping should continue.
- Since the counter controlled loop is controlled by a counter value.
- At each iteration counter value will increase or decrease with a definite value and condition will be checked.
- So the number of loop execution becomes definite.

### Example

```
int sum = 0;
int n = 1;
while (n <= 10){
    sum = sum + n*n;
    n = n+ 1;
}
```

### **Sentinel Controlled Loops**

- It is also called indefinite loop.
- In this, the number of iterations is not known in advance.
- In this, a special value called sentinel value which is used to change the loop control expression from true to false.
- The value of the control variable differs within a limitation and the execution can be terminated at any moment.
- The value of this variable changes inside the loop.
- The loop breaks when the value of the control variable matches the condition.

### Example

```
do{
    printf("Input a number");
    scanf("%d", &num);
}while(num > 0);
```

NO:	TOPICS	COUNTER CONTROLLED LOOP	SENTINEL CONTROLLED LOOP
1	Number of execution	Previously known number of executions take place	Unknown number of executions take place
2	Condition variable	Condition variable is known as counter variable. As because, it counts the total number of executions against the max number of executions	Condition variable is known as sentinel variable, which means a guard. This variable waits for a decision made inside the loop to let another cycle take place or break the loop
3	Value and limitation of variables.	The value of the variable and the limitation of the condition for the variable both are strict	The limitation for the condition variable is strict but value of the variable varies in this case.

**Exercise**

1. Write a C program to check whether the entered year is a leap year or not.
2. Write a C program to check whether the entered number is even or odd.
3. Write a C program to check whether the entered character is an alphabet, digit or a special character.
4. Write menu driven program in C to find the arithmetic operations.
5. Write a C program to perform string operations using switch – case.
6. Write a program in C to display the first N natural numbers.
7. Write a program in C to display the multiplication table of a given integer.
8. Write a program in C to display the pattern like right angle triangle with a number.
 

1  
 i. 1 2  
 ii. 1 2 3  
 iii. 1 2 3 4
9. Write a C program to find sum of all odd numbers between 1 to n.
10. Write a C program to find the factorial of a given number