Unit-5
Control Statements

Introduction

The statements which alter the flow of execution of the program are known as **control statements.**

Sometimes we have to do certain calculations/tasks depending on whether a condition or test is true or false. Similarly, it is necessary to perform repeated actions or skip some statements. For these operations, control statements are needed.

Types of Control Statements

There are two types of control statements:

- Decision Making(or branching) Statements
  - If statement
  - If…else statement
  - Nested if...else statement
  - Else if statement
  - Switch statement
- Loop statement or Iteration
  - For loop
  - While loop
  - Do…while loop

Decision making statements

- Used to making decisions based upon certain condition.
- Conditions are decide whether or not a statement should be executed.
- If the condition is determined to be true, a statement is executed and optionally other statements to be executed if the condition is determined to be false.

Types of decision making statements:

If statement

If a single statement or block of statements has to be executed only when the given condition is true, if- statement is used.

Syntax:

```
if (condition)
{
   //Statement(s);
}
```

The if statement first checks a condition and then, it executes the statements within its block if the condition is true.

E.g.

Program to test whether the given number is negative:

```
#include<stdio.h>
#include<conio.h>
```

```
main()
{
   int n;
   printf("Enter a number to be tested:");
   scanf("%d",&n);
   if(n<0)
      printf("The number is negative");
   getch();
   return 0;
}
```

If…else statement

The if-else statement is used when there are only two possible actions-one happens when a test condition is true, and other when it is false.

Syntax:

```
if(condition)
{
   //Statement block
}
else
{
   //Statement block
}
```

E.g.

Program to find whether a number is odd or even:

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a;
   printf("Enter a number :");
   scanf("%d",&a);
   if (a%2==0)
   {
      printf("%d is even",a);
   }
   else
   {
      printf("%d is odd",a);
   }
   getch();
}
```

Nested if...else statement

The nested if-else statement includes if-else statement inside if-else statement itself.

Syntax:

```
if(condition1)
{
```

```
  if(condition2)
  {
    statement/statements;
  }
  else
  {
    statement/statements;
  }
}
else
{
  if(condition3)
  {
    statement/statements;
  }
  else
  {
    statement/statements;
  }
}
```

E.g.

```
#include<stdio.h>
main()
{
  int num,rem;
  printf("Enter an integer Number:");
  scanf("%d",&num);
  if(num>=0)
  {
    rem=num%2;
    if(rem==0)
    {
       printf("%d is positive even number",num);
    }
    else
    {
       printf("%d is positive odd number",num);
    }
  }
  else
  {
    rem=num%2;
    if(rem==0)
    {
      printf("%d is negative even number",num);
    }
    else
    {
      printf("%d is negative odd number",num);
    }
```

```
  }
  return 0;
}
```

Else if statement (if-else-if ladder)

Else if is used when multipath decisions are required.

Syntax:

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

As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the else-if ladder is bypassed. If none of the conditions are true, then the final else statement will be executed.

E.g.

```
#include <stdio.h>
void main( )
{
   int a;
   printf("Enter a number...");
   scanf("%d", &a);
   if(a%5 == 0 && a%8 == 0)
   {
      printf("Divisible by both 5 and 8");
   }
   else if(a%8 == 0)
   {
      printf("Divisible by 8");
   }
   else if(a%5 == 0)
   {
      printf("Divisible by 5");
   }
   else
   {
      printf("Divisible by none");
   }
}
```

Q. Write a program to find whether given an integer is divisible by 3 and 5 but not by 10.

```
#include <stdio.h>
#include<conio.h>
main( )
{
   int n;
   printf("Enter an integer:");
   scanf("%d", &n);
   if(n%3 == 0 && n%5 == 0)
   {
      if(n%10!=0)
      {
        printf("%d is divisible by 3 & 5 and not by 10", n);
      }
      else
      {
         printf("%d is divisible by 3, 5 & 10", n);
      }
   }
}
```

Q. Write a program to read three numbers from user and determine the largest number among them.

```
#include <stdio.h>
#include<conio.h>
int main( )
{
   int n1, n2, n3;
   printf("Enter 3 numbers:");
   scanf("%d%d%d",&n1,&n2,&n3);
   if(n1>n2)
   {
      if(n1>n3)
         printf("largest=%d", n1);
      else
         printf("largest=%d", n3);
   }
   else
   {
      if(n2>n3)
         printf("largest=%d", n2);
      else
         printf("largest=%d", n3);
   }
   getch();
   return 0;
}
```

Switch case statement

When there are a number of options available and one of them is to be selected on the basis of some criteria, switch statement is used. Thus, a switch statement allows user to choose a statement among several alternatives. The switch statements compares a variable or expression with different constants (i.e. cases). If it is equal to case constant, a set of statements following the constants are executed. If there is no match, the default statements are executed.

Syntax:

```
switch (expression)
{
   case constant1:
     // statements
     break;

   case constant2:
     // statements
     break;
   .
   .
   .
   default:
     // default statements
}
```

The break is used to break out of the case statements. break is a keyword that breaks out of the code block. The break prevents the program from testing the next case statement also.

E.g.

```
#include <stdio.h>
int main() {
   int num = 8;
   switch (num) {
      case 7:
         printf("Value is 7");
         break;
      case 8:
         printf("Value is 8");
         break;
      case 9:
         printf("Value is 9");
         break;
      default:
         printf("Out of range");
         break;
   }
   return 0;
}
```

Q. Write a program to find whether a input char is vowel or constant or other characters using switch statement.

```
#include <stdio.h>
int main() {
  char c;
  printf("Enter a character:");
  scanf("%c", &c);
  if(c>='A'&&c<='Z'||c>='a'&&c<='z')
  {
    switch (c) {
       case'a':
       case'e':
       case'i':
       case'o':
       case'u':
       case'A':
       case'E':
       case'I':
       case'O':
       case'U':
          printf("It is vowel");
          break;
       default:
          printf("It is consonant");
    }
  }
  else
    printf("It is other character");
    return 0;
}
```

Loop Statements or Iteration

Loops are used when we want to execute a part of program or block of statement several times. So, a loop may be defined as a block of statements which are repeatedly executed for a certain number of times or until a particular condition is satisfied. There are three types of loop statements in C:

1. For
2. While
3. Do...while

For loop

The for loop is the most commonly used looping in C. When the number of repetitions is known in advance, the use of this loop will be more efficient.

For loop contains 3 parts Initialization, Condition and Increment or Decrements.

1. Initialization part: Where value is assigned to the variable.
2. Conditional or Logical Part: where the condition is checked until the expression is true.
3. Increment or decrement part: Which change index value of the for loop.

Syntax:

```
for(initilization; condition expression; increment/decrement)
{
   //Body of loop
}
```

E.g.

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

Q. Write a program to calculate sum of first n natural number.

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int n, i, sum=0;
   printf("\n Enter a number n:");
   scanf("%d", &n);
   for(i=1; i<=n; i++)
   {
      sum=sum+i;
   }
   printf("\nThe sum is %d", sum);
   getch();
   return 0;
}
```

While Loop

Syntax:

```
initialization;
while(condition_expression)
{
   //Body of loop
   //update statement(increment/decrement)
}
```

First the condition is evaluated; if it is true then the statements in the body of loop are executed. After the execution, again the condition is checked and if it is found to be true then again the statements in the body of loop are executed. This means that these statements are executed continuously till the condition is true and when it becomes false, the loop terminates and the control comes out of the loop.

E.g.

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

Q. Write a program to calculate factorial of a number using while loop.

```
#include<stdio.h>
#include<conio.h>
int main( )
{
  int n, i;
  long int fact;
  printf("Enter the integer:");
  scanf("%d", &n);
  fact=1, i=1;
  while(i<=n)
  {
     fact=fact*i;
     i++;
  }
  printf("Factorial of %d is %ld", n, fact);
  getch();
  return 0;
}
```

Do-while Loop

Syntax:

```
initialization;
do{
   //Body of loop
   //update statement
}while(condition_expression);
```

Here firstly the segments inside the loop body are executed and then the condition is evaluated. If the condition is true, then again the loop body is executed and this process continues until the condition becomes false.

E.g.

```
#include<stdio.h>
#include<conio.h>
```

```
void main()
{
  int i;
  i=1;
  do
  {
    printf("\n%d",i);
    i++;
  }while(i<5);
  getch();
}
```

Difference between while and do-while loop

1. The while loop is **pre-test loop**, where firstly the condition is checked and if the condition is true then only the statements of the while loop execute. The do-while loop is a **post-test loop**. In the do-while loop, the statements of the do-while loop are executed after that, the condition is evaluated, and if the condition is true then again the statements of the do-while loop are executed.

2. The condition of the while loop is at the top of the loop but the condition of the do-while loop is at the bottom of the loop.

3. While loop can't be terminated with a semicolon but the do-while loop should be terminated with a semicolon.

4. The statements of the do-while loop execute at least 1 time in every condition. In the while loop, the test expression evaluates false in first checking then the statements of the while loop is not executed. But the condition of the do-while loop is checked at the end of the loop, so it is guaranteed that the statements of the do-while loop execute at least once.

Q. Write a program to print the even number upto n using do-while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int n, i;
  printf("Enter a value of n:");
  scanf("%d",&n);
  i=1;
  do
  {
    if(i%2==0)
        printf("%d\t",i);
    i++;
  }while(i<=n);
  getch();
}
```

Q. Write a program to compute and print the sum of given numbers of squares. [$1^2+2^2+3^2+….+n^2$]

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int n, i, sum=0;
  printf("Enter a value of n:");
  scanf("%d",&n);
  for(i=1;i<=n;i++)
  {
     sum=sum+i*i;
  }
  printf("\nThe sum is:%d", sum);
  getch();
}
```

Q. Write a program using loop to compute a^b (a raised to power b) with a, b as input.

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int a, b;
  int result, i;
  printf("Enter the base:");
  scanf("%d", &a);
  printf("Enter the exponent:");
  scanf("%d", &b);
  if(b==0)
    result=1;
  else
  {
     result=1;
     for(i=1; i<=b;i++)
     {
        result=result*a;
     }
  }
  printf("%d raised to power %d=%d", a, b, result);
  getch();
}
```

Nesting of Loops

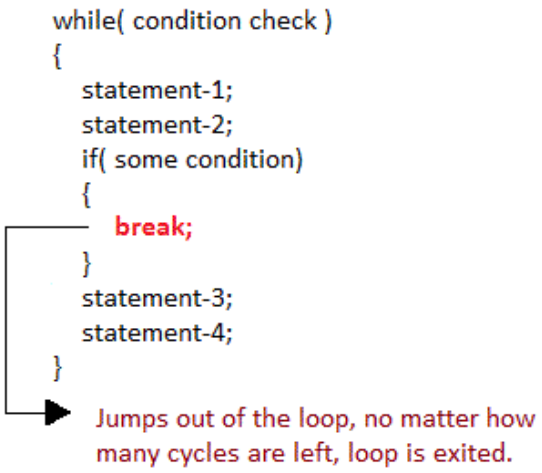
When a loop is written inside the body of another loop, then it is known as nesting of loops. Any type of loop can be nested inside any other type of loop. For example, a for loop may be nested inside another for loop or inside a while or do...while loop. Similarly, while and do while loops can be nested.

E.g.

```
#include<stdio.h>
void main( )
{
   int i, j;
   /* first for loop */
   for(i = 1; i < 5; i++)
   {
      printf("\n");
      /* second for loop inside the first */
      for(j = i; j > 0; j--)
      {
         printf("%d", j);
      }
   }
}
```

Break Statement

The break statement is used to terminate loops or exit from a switch. It can be used within a for, while, do –while, or switch statement. If a break statement is included in a while, do – while or for loop, then control will immediately be transferred out of the loop when the break statement is encountered.



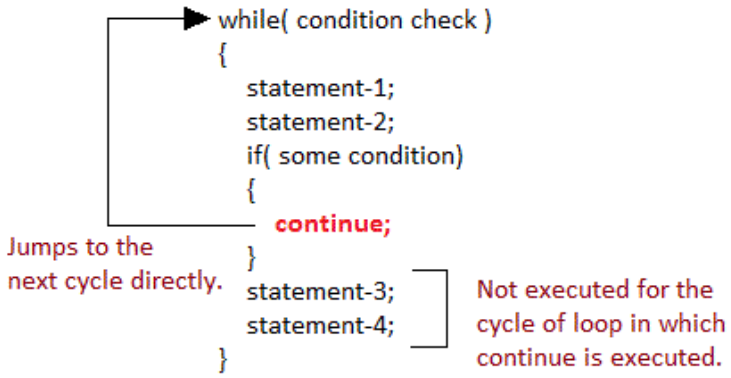
E.g.

```
#include <stdio.h>
int main()
{
  int i;
  double number, sum = 0.0;
  for (i = 1; i <= 10; ++i)
  {
    printf("Enter a n%d: ", i);
    scanf("%lf", &number);
```

```
    // if the user enters a negative number, break the loop
    if (number < 0.0)
    {
      break;
    }
    sum = sum + number;
  }
  printf("Sum = %lf", sum);
  return 0;
}
```

Continue statement

The continue statement is used to bypass the reminder of the current pass through a loop. The loop does not terminate when a continue statement is encountered. Rather, the remaining loop statements are skipped and the computation proceeds directly to the next pass through the loop.



E.g

Program to print sum of odd numbers between 0 and 10

```
#include <stdio.h>
int main ()
{
 int a,sum = 0;
 for (a = 0; a < 10; a++)
 {
   if ( a % 2 == 0 )
     continue;
  sum = sum + a;
  }
 printf("sum = %d",sum);
 return 0;
}
```

Goto Statement

The goto statement is used to alter the normal sequence of program execution by transferring control to some other part of the program.

Syntax:

goto label;

Where label is an identifier that is used to label the target statement to which control will be transferred. The target statement must be labeled, and the label must be followed by a colon. Thus, the target statement will appear as

label: statement

E.g.

```
#include<stdio.h>
#include<conio.h>
int main( )
{
  int n ;
  printf("Enter the number :") ;
  scanf ("%d", &n) ;
  if (n%2 == 0)
    goto even ;
  else
    goto odd;
  even :
     printf ("Number is even") ;
     goto end ;
  odd :
     printf ("Number is odd") ;
  end :
     printf ("\n") ;
  getch( );
  return 0;
}
```

Q. Write a program to determine whether a number is prime or not.

```
#include <stdio.h>
#include<conio.h>
int main( )
{
   int i, n;
   printf("Enter a number:");
   scanf("%d", &n);
   for(i=2; i<n; i++)
   {
      if(n%i==0)
      {
         printf("\nNot Prime!!");
         break;
      }
   }
   if(i==n)
   {
      printf("\nPrime Number!!");
   }
   getch();
   return 0;
}
```

Q. Write a program to give the following output.

```
1
2    3
4    5    6
7    8    9    10
11   12   13   14   15 …….
```

```
#include <stdio.h>
#include<conio.h>
int main( )
{
   int n, i, j, a;
   printf("How many rows:");
   scanf("%d", &n);
   a=1;
   for(i=1; i<=n; i++)
   {
      for(j=1; j<=i;j++)
      {
         printf("%d\t", a++);
      }
      printf("\n");
   }
   getch();
   return 0;
}
```

Q. Write a program to give following output:

```
*
*   *
*   *   *
*   *   *   *
*   *   *   *
```

```
#include <stdio.h>
#include<conio.h>
#define n 5
int main( )
{
   int i, j;
   for(i=1; i<=n; i++)
   {
      for(j=1; j<=i;j++)
      {
         printf("\t*");
      }
      printf("\n");
   }
   getch();
   return 0;
}
```

Q. Write a program to give following output:

```
1
1   2
1   2   3
1   2   3   4 …….upto n rows.
```

```
#include <stdio.h>
#include<conio.h>
int main( )
{
   int n, i, j;
   printf("How many rows:");
   scanf("%d", &n);
   for(i=1; i<=n; i++)
   {
      for(j=1; j<=i;j++)
      {
         printf("%d\t", j);
      }
      printf("\n");
   }
   getch();
   return 0;
}
```

Control Structures

Control Structure:

- Control structure specify the order in which the various instructions or statements in a program are to be executed by the computer.
- Control structure determines the “flow of control” in a program.

Types of Control Structure:

1. Sequence

- It is also know as a linear statement where control flows form top to bottom.

2. Decision(Selection)

- It also know as branching statement created using a switch, if, if…else, nested if….else, ladders of if…else etc.

3. Looping

- It is also know as looping created using for, while, do….while.

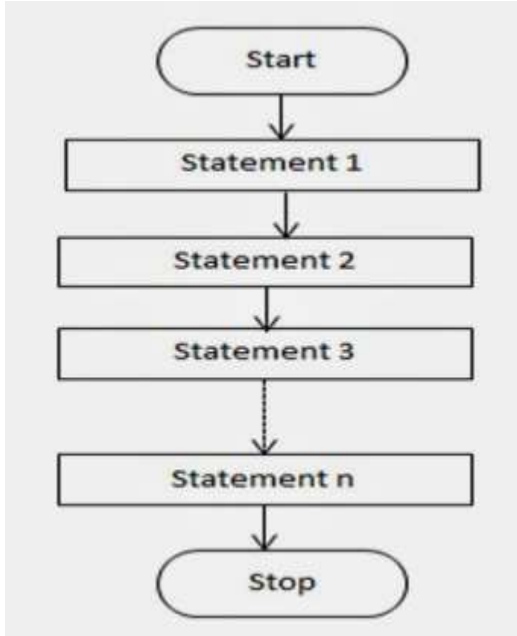
1. Sequence:

- In a sequential statement, program control flows from top to bottom.
- Program statements are executed one after another in a sequence manner.
- This is simple type of control structure.
- It does not contain any conditions and looping statements.

Syntax:

```
Statement_1;
Statement_2;
.
.
.
.
.
Statement_N;
```

Flowchart:



Example:

1. Program to convert kilometer into meter.
2. Program to find quotient and reminder.
3. Program to find sum of two input numbers.
4. Program to calculate area of a circle.
5. Program to convert Celsius temperature into Fahrenheit.
6. Program to calculate Simple Interest.
7. Program to find average of 3 input numbers.
8. Program to find third angle of a triangle by taking two angle as an input.

2. Selection/Branching/Decision:

- It allows us to control the flow of program's execution based upon condition.
- The conditional statement is mainly used for decision making.

Selection Statements are:

1. if statement
2. if else statement
3. nested if…else statement
4. if…else if…else statement(ladders of if…else)
5. Ternary operator
6. Switch statement

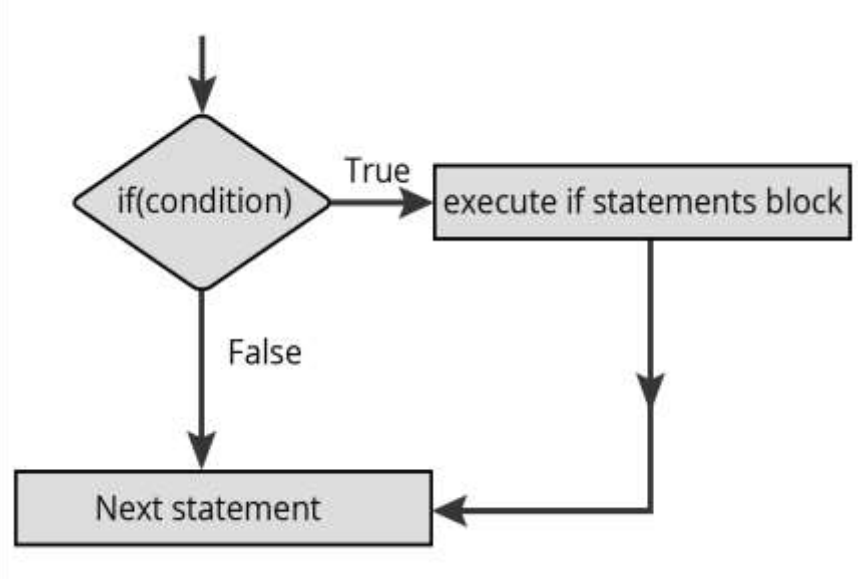
1. if statement:

- The if statement is used to express conditional expressions.
- If the given condition is true then it will execute the if statements otherwise it will execute the optional statements.

Syntax:

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

Flowchart:



Example:

1. Program to calculate discount based on given condition.
2. Program to compare two input numbers.
3. Program to check the given input number is divisible by 7 or not.
4. Program to check the result of student based on given input marks.
   [Pass Marks=40]

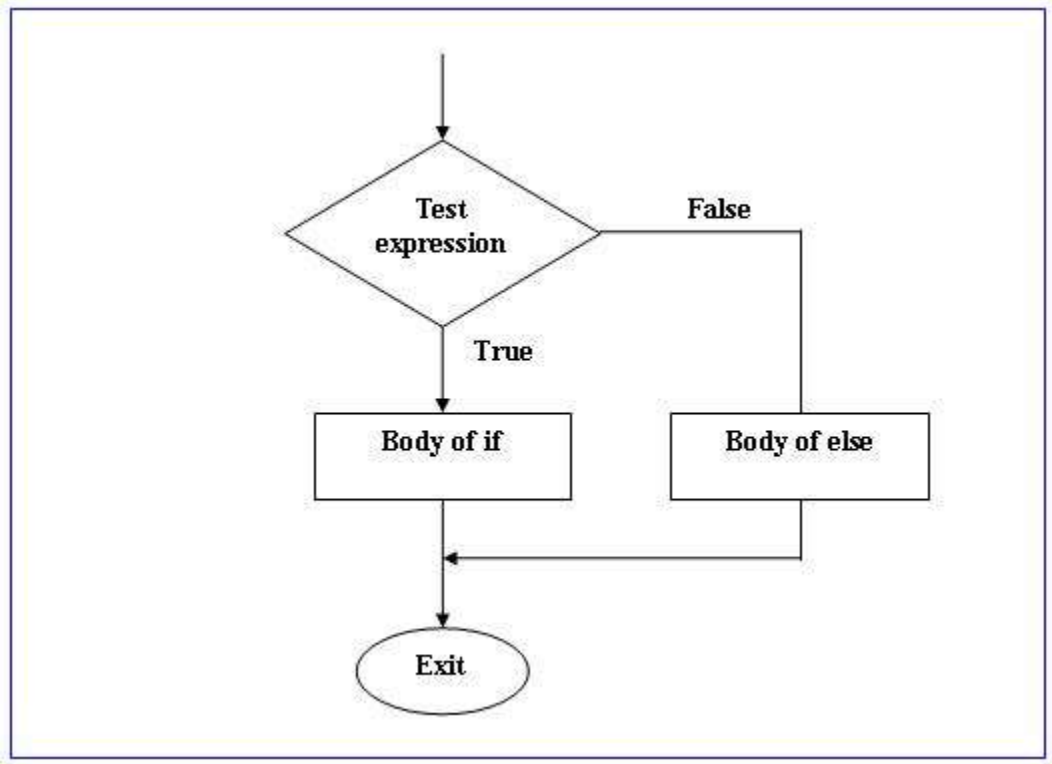
2. if…else statement:

In this statement, if the given condition is true or satisfied then if statement will execute otherwise else statement will be execute.

Syntax:

```
if(condition)
{
        statements_1;
}
else
{
        Statements_2;
}
```

Flowchart:



Example:

1. Program to check the given input number is odd or even.
2. Program to check the given input number is greater than 10 or not.
3. Program to check the given input number is divisible by 7 or 9 or not.
4. Program to check the given input number is divisible by 5 and 7 or not.
5. Program to input a mark in a subject of a student and check if the student is pass or not. (Pass Marks=32)
6. Program to check the given input number is positive or not.
7. Write a C program to check whether a year is leap year or not.
8. Write a C program to check whether a character is alphabet or not.
9. Program to check if the person is eligible for vote or not based on input age.
10. Program to find the largest number between two input numbers.

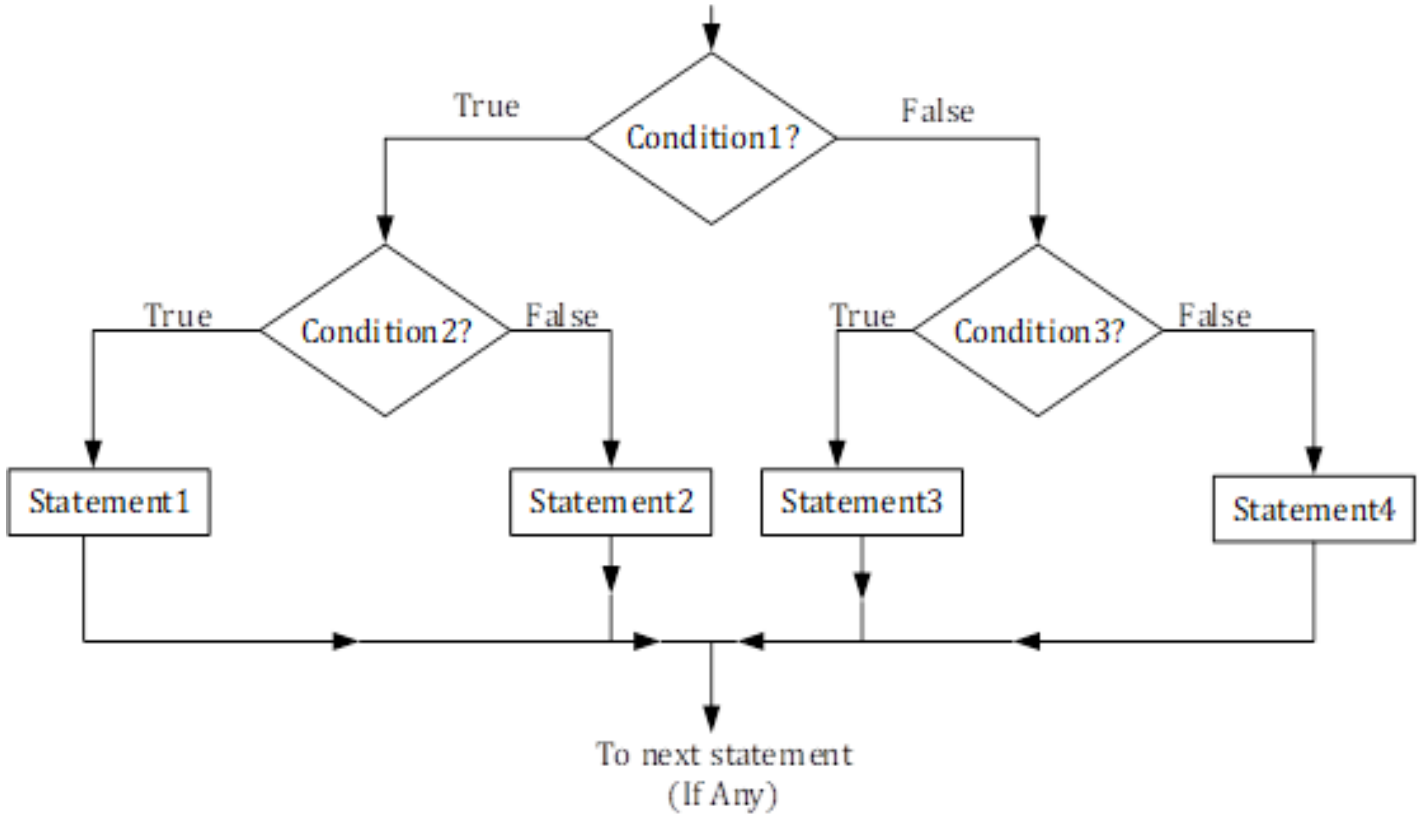
3. Nested if…else Statement:

The if…else statement inside another if…else statements then this is called nested if…else statements.

Syntax:

```
if(condition1)
{
        if(conditon2)
        statements_1;
        else
        statements_2;
}
else
{
        if(conditon3)
        statements_3;
        else
        statements_4;
}
```

Flowchart:



Example:

1. Program to input three different numbers and find the largest among 3 numbers.
2. Program to check the given input number is equal to 1 and less than 10 or greater than 10.

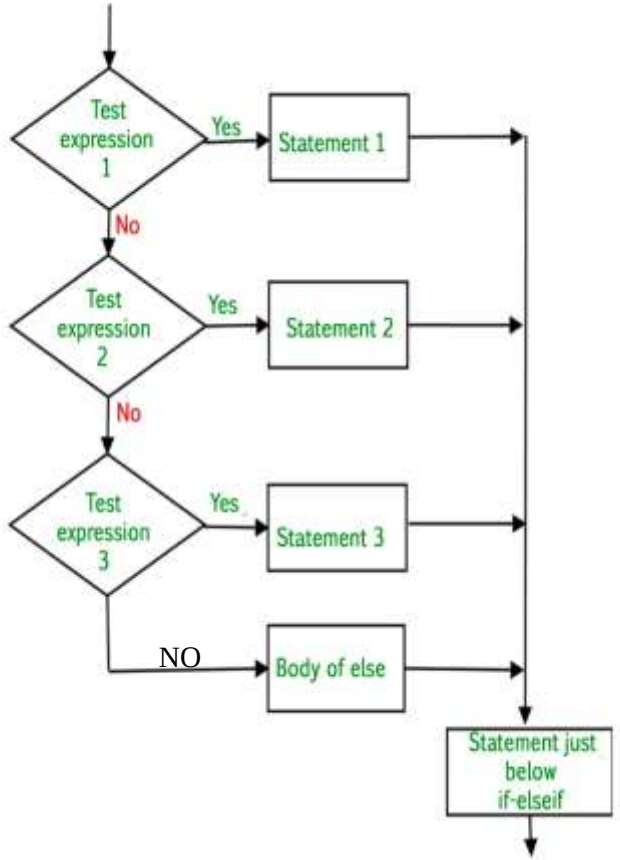
4. if…else if…else statement (ladders of if…else)

- The conditions are evaluated from the top to bottom.
- As long as a true condition is found, the statement associated with it is executed and the rest of the ladder is bypassed.
- If none of the conditions are true , the final else is executed.

Syntax:

```
if(condition1)
    statement1;
else if (condition2)
    statement2;
else if (condition3)
    statement3;
--------------
else
     statement;
```

Flowchart:



Examples:

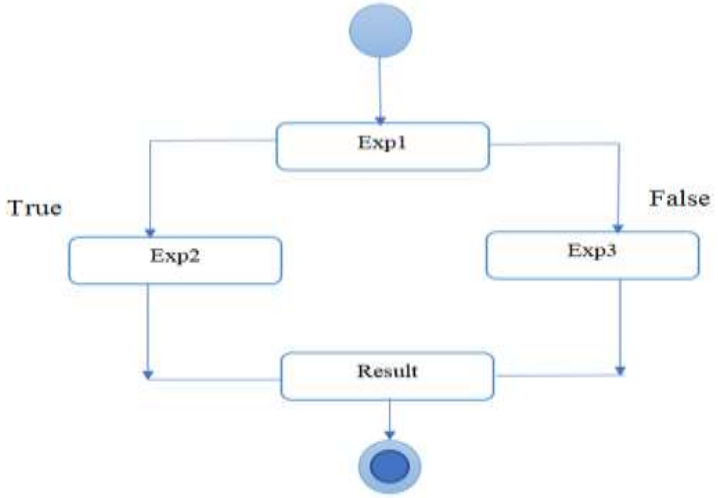
1. Program to check the given input number is positive or negative or equal to zero.
2. Program to find the largest number among four input numbers.
3. Program to check the division of students based on percentage.
4. Program that reads a character from the user and find the entered character is in lowercase, uppercase, digit, whitespace or unknown character using if…else if…else statement.

5. Ternary operator (Conditional Operator):

- Conditional operator (?:) stores a value depending upon a condition.
- This operator can also act as decision control statement like if statement.
- This operator requires **three operands**.

Syntax: expression1**?** expression2**:** expression3;

Flowchart:



Example:

1. Program to input age of a person and find out whether the person is eligible for driving or not.
2. Program to check the largest number between two input numbers.
3. Program to check whether a person is eligible to vote or not.
4. Program to check the largest number among three input numbers.

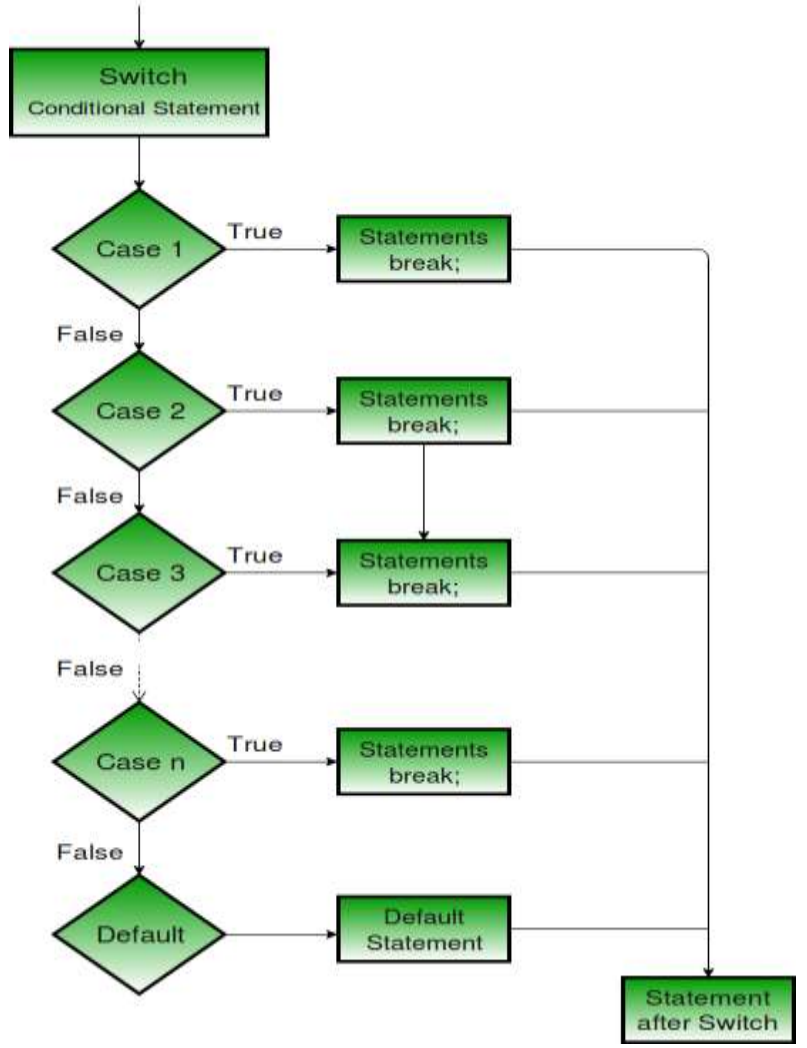
6. Switch-case statement:

- Switch statement in C tests the value of a variable and compares it with multiple cases.
- Once the case match is found, a block of statements associated with that particular case is executed.
- Each case in a block of a switch has a different name/number which is referred to as an identifier.
- **Switch expression must evaluate to an integer or character constant value.**
- Every case contains a break statement. If there is no break statement then subsequent expression will also get evaluated until it reach the next break statement.
- If any case is not matched then default statement will be execute.

Syntax of switch-case statement:

```
Switch(expression)
{
        case constant1:
                statement1;
                  break;
        case constant2:
                  statement2;
                  break;
        case constant3:
                  statement3;
                  break;
        ---------------------
        default:
                  statement;
```

Flowchart:



Example:

1. Program to display day using the switch statement depending upon the number entered i.e. input 1 for Sunday, 7 for Saturday.
2. Program to calculate sum, difference, product, division and reminder using switch case statement.
3. Write a C program to find area of circle, area of square, area of rectangle and simple interest using switch case statement.
4. Write a C program to find maximum between two numbers using switch case.
5. Write a C program to check whether a number is even or odd using switch case.

Looping

Looping:

- In simple, looping means repetition.
- A loop is a programming structure that repeats a sequence of instructions till a specific condition is true.
- A Loop executes the sequence of statements many times until the stated condition becomes false.
- The purpose of the **loop** is to repeat the same code a number of times.

Looping Statements are:

1. **for loop** statement
2. **while loop** statement
3. **do…while** statement
4. **Jump** statement

1. For Loop:

- The for loop is the most commonly used statement in C.
- For loop consists of three expressions.

Syntax:

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

Flowchart:

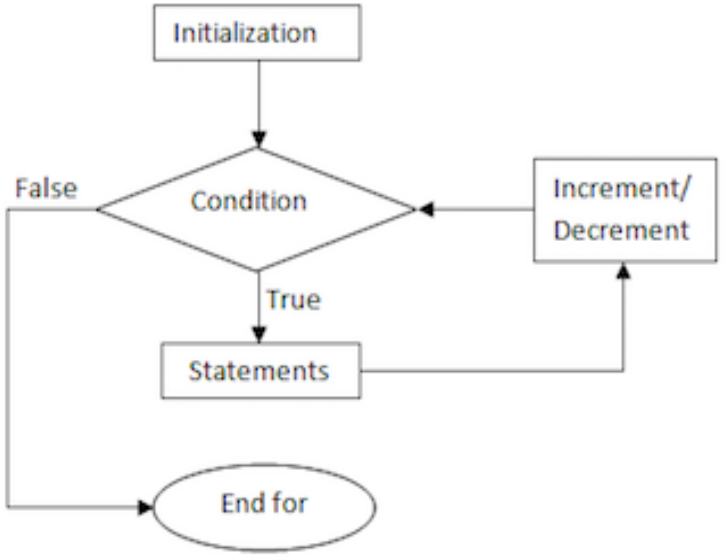


fig: Flowchart for for loop

Examples:

1. WAP to print “Hello World” 10 times using for loop.
2. WAP to print your name 20 times using for loop.
3. WAP to print natural numbers up to 10 using for loop.
4. WAP to print even numbers from 0 to 100 using for loop.
5. WAP to print odd numbers from 0 to 100 using for loop.
6. WAP to find sum of 10 natural numbers using for loop.
7. WAP to find sum of even numbers from 0 to 100.
8. WAP to find sum of odd numbers from 0 to 100.
9. WAP to check the given input number is prime or not.
10. WAP to find the factorial of given input number.
11. WAP to display the Fibonacci series using for loop.
12. WAP to find sum of the cubes of first ten numbers.

Nested for loop:

- Using for loop inside another for loop is called nested for loop.

Syntax:

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

Flowchart of nested for loop:

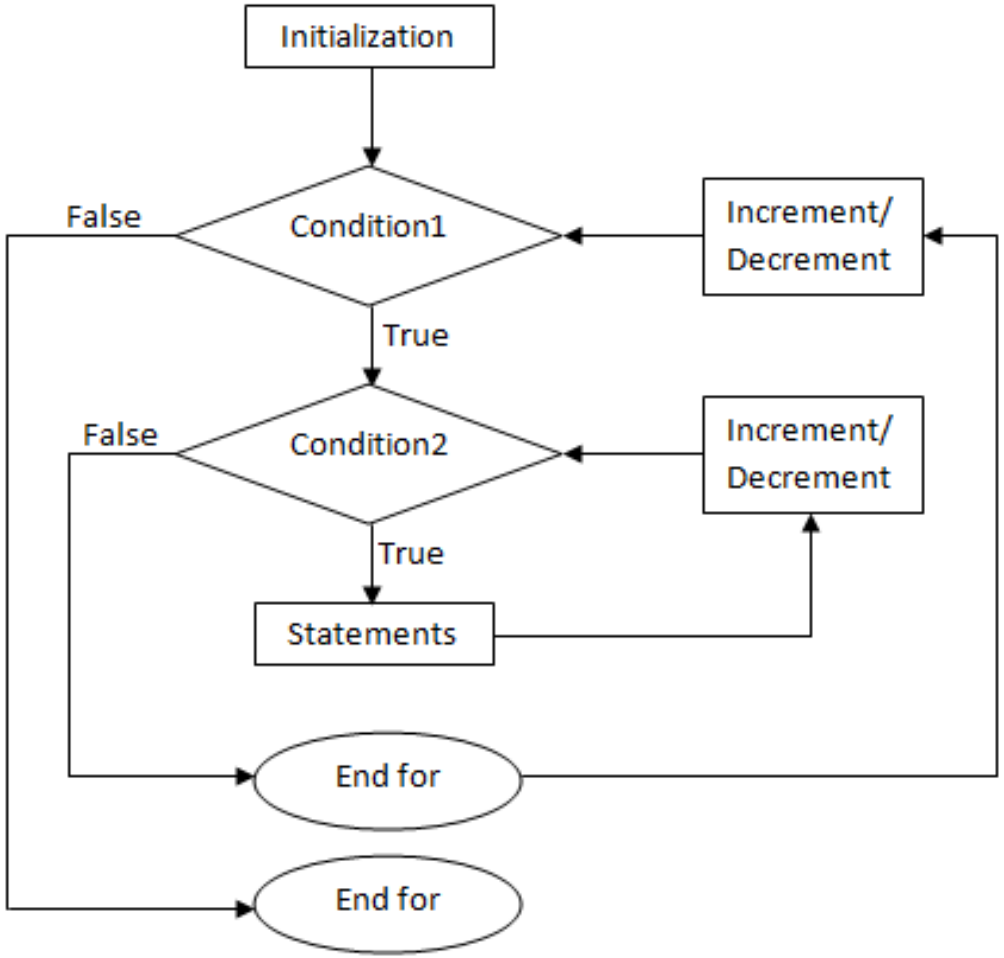


Fig: Flowchart for nested for loop

Examples:

1. Program to display the multiplication table of all the numbers from 1 to 5.
2. Program to print all the prime numbers from 0 to 100.

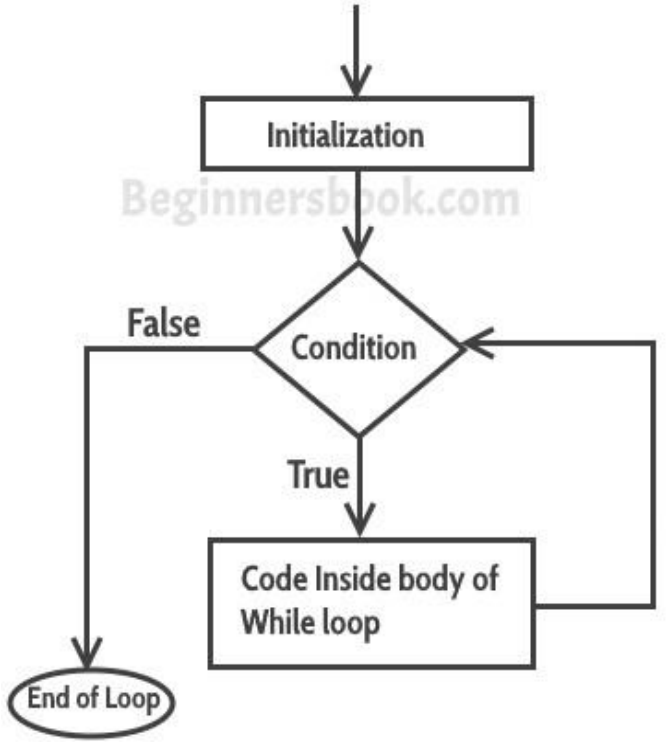
2. While loop:

- While loop first checks whether the initial condition is true or false and finding it to be true, it will enter the loop and executes the statement.
- While loop and for loop is an example of entry controlled loop.

Syntax:

```
initialization;
while(condition)
{
        statements;
        increment/decrement;
}
```

Flowchart:



Examples:

1. WAP to print “Kathmandu BernHardt” 10 times using while loop.
2. WAP to display all the numbers from 1 to 10.
3. WAP to find the sum of first ten numbers using while loop.
4. WAP to print all the even numbers from 0 to 100.
5. WAP to print all the odd numbers from 0 to 50 using while loop.
6. WAP to calculate the sum of even numbers from 0 to 100.
7. WAP to count the total number of digits in the input number.
8. WAP to find the reverse of given input number.
9. WAP to check the given input number is palindrome or not.
10. WAP to find the sum of individual digits from the given input numbers.
11. WAP to check whether the given input number is Armstrong or not.

(Example: 0, 1, 153, 370, 371,407, 1634, 8208, 9474, 54748, 92727, 548834, 1741725, 4210818 etc.)

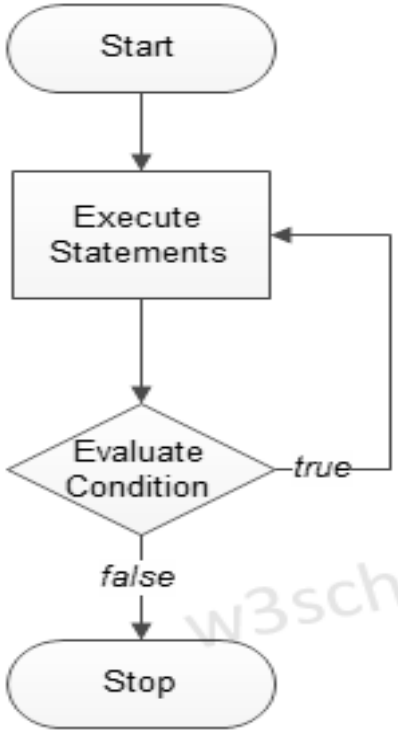
3. do…while loop:

- do while loop is a control flow statement that executes a block of code at least once, and then either repeatedly executes the block, or stops executing it, depending on a given Boolean condition at the end of the block.
- do while loop is also called **exit controlled loop**.

Syntax:

```
initialization;
do
{
        statements;
        increment/decrement;
}while(condition);
```

Flowchart:



Examples:

1. Write a C program to print “C programming” 5 times using do while loop.
2. WAP to find the sum of all the numbers from 1 to 10.
3. WAP to find the reverse of given input number using do while loop.
4. WAP to print all the odd numbers form 1 to 100.
5. WAP to calculate the sum of even numbers from 1 to 100.

4. Jump Statement:

The jump statements includes:

i. break statement
ii. continue statement
iii. goto statement

Difference between break and continue statement:

Break Statement	Continue Statement
1. The break statement is used to terminate the control from the switch...case structure as well in the loop.	1. The continue statement is used to bypass the execution of the further statements.
2. When the break statement encountered it terminates the execution of the entire loop or switch case.	2. When the continue statement is encountered it bypass single pass of the loop.
3. It does not bypass the current loop through it transfers the control out of the loop.	3. It is used to bypass the current pass through a loop.
4. The loop terminates when a break is encountered.	4. The loop doesn’t terminate when continue is encountered.
5. **Syntax**: break;	5. **Syntax**: continue;

Example of break statement:

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int i;
        for(i=1;i<=10;i++)
{
        if(i==5)
        break;
        printf("%d",i);
}
getch();
return 0;
}
```

Example of continue statement:

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int i;
        for(i=1;i<=10;i++)
{
        if(i==5)
        continue;
        printf("%d",i);
}
getch();
return 0;
}
```

goto statement:

- The goto statement is used to alter the normal sequence of program execution by transferring control to the some other part of the current function. In goto statement we use label.

Syntax: goto label;

Flowchart: