Control Statements

Introduction

The control statements defines the flow of the program according to the conditions. A control statement is very helpful tool for programmer to make more realist decision based program. There are various statements allowed you to control your program wisely.

1. Decision making statements
2. Iteration control statements
3. Selection control statements
4. Flow control statements

First 2 statements! Decision making, Iteration control statements work on True/False value means boolean values(we will learn about boolean value in higher languages). In C Language, All true values indicate by 1, However every non-zero or nonempty value is True. Every empty literal, collection, or Zero is a False value.

	Notation	Description	Example
TRUE	1	every non-zero or non-empty value/literal is a True value.	+ve numbers, -ve numbers, "Hello", "NULL", 'A', etc.
FALSE	0	zero or empty value/literal is a False value.	0, '\0', NULL, "", '' , etc.

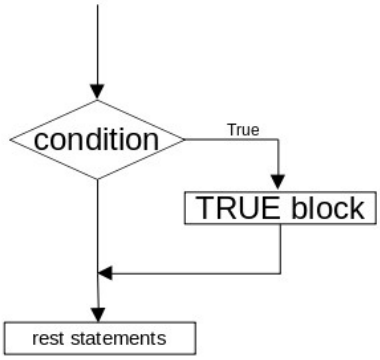
NOTE: Every True condition returns 1(As TRUE) & every false condition returns 0(As FALSE).

Decision making statements

These statements have multiple blocks, according to condition decides which block would be executed. Every starting & ending curly bracket covers a block. A block contains the number of statements. There are following decision making statements in C language.

1. if statement

This statement has only one block known as TRUE block. The TRUE block executes when the condition is True, otherwise program will continue its execution flow by ignoring the TRUE block.



flowchart :

If a condition is true then the TRUE block will execute and then rest statements of program will be executed. When the condition is false, rest statements executed normally.

CEC

Syntax:

```
if(condition)
{
      //TRUE block
}
```

if condition is true then TRUE block will be executed.

Program 1: Write a program to display an alert message when a user enters a -ve number or greater than 100.

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

     if(num>100 || num<0)
     {     //TRUE block
          printf("You entered wrong number !!!\n");
          printf("\a"); // '\a' for alert sound
     }
     printf("Number is : %d\n",num); //rest statements
getch();
}
```

OUTPUT 1:

```
Enter the number : 89
Number is : 89
```

OUTPUT 2:

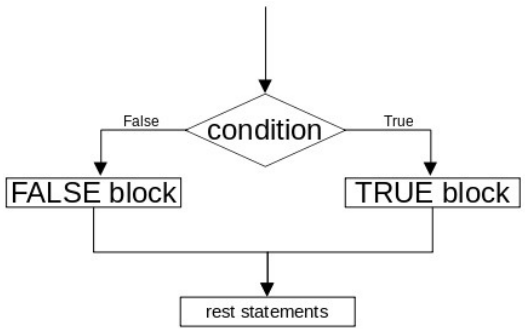
```
Enter the number : 101
You entered wrong number !!!
Number is : 101
```

2. if-else statement

This statement is used to make decisions when the condition is true as well as false. The True block of the statement will be executed when the condition is true otherwise the False block will be executed. However, the rest of the program will always execute either the condition is false or true.

CEC

flowchart:



If a condition is true then The TRUE block will be executed, else False block. The rest statements of the program will execute after the execution of TRUE/FALSE block.

Syntax:

```
if(condition)
{
      //TRUE block
}
else
{
      //FALSE block
}
```

Program 2: write a program to find whether an entered number is odd or even.

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

     if(num%2==0) // if the remainder is 0 , then number is even.
     {     //TRUE block
           printf("%d is the even number.\n",num);
     }
     else  // if the remainder is not zero, them number is odd.
     {     //FALSE block
           printf("%d is the odd number.\n",num);
     }
getch();
}
```

OUTPUT:

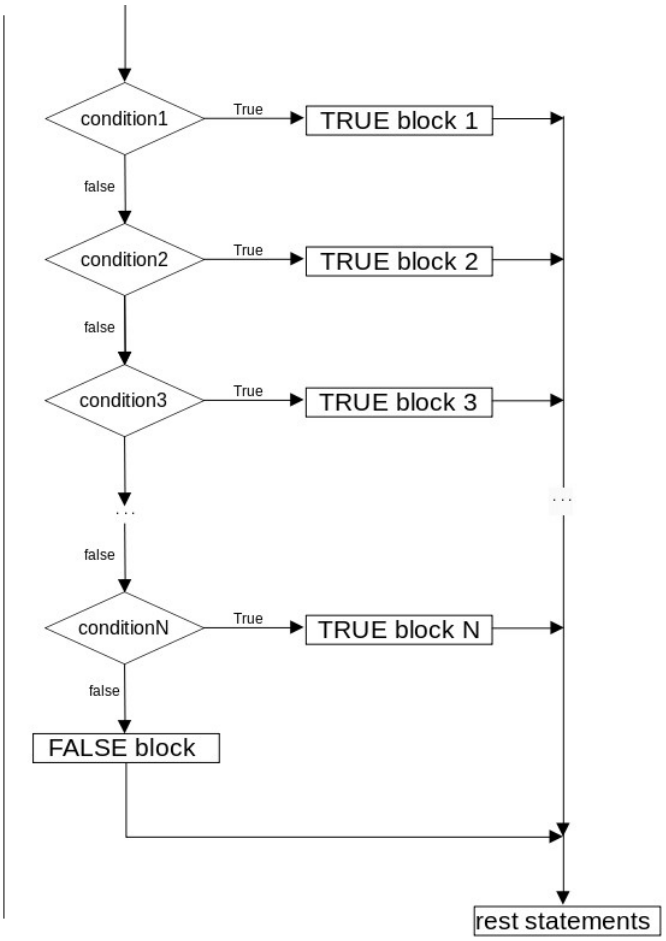
```
Enter the number : 92
92 is the even number.
```

3. else-if ladder

The else-if ladder works on multiple conditions as a single task. It is very helpful when a programmer wants to make a decision on multiple conditions or check multiple conditions in a single manner.

flowchart :

As per flowchart, There are serval conditions in the else-if ladder. If a condition fails, the next condition will be check unless it arrives at a true condition or the FALSE block. If a condition is true then according to that condition it's TRUE block will execute.



Syntax:

```
if(condition1)
{
      // TRUE block 1
}
else if(condition2)
{
      // TRUE block 2
}
else if(condition3)
{
      // TRUE block 3
}
...
else
{
      //FALSE block
}
```

Program 3: write a program to display the day according to the given day number.

```
#include<stdio.h>
#include<conio.h>
void main(){
     int day;
clrscr();
     printf("Enter the day number [ 1-7 ] : ");
     scanf("%d",&day);
```

CEC

```
    if(day==1) {
        printf("Monday\n");
    }
    else if(day==2){
        printf("Tuesday\n");
    }
    else if(day==3) {
        printf("Wednesday\n");
    }
    else if(day==4) {
        printf("Thursday\n");
    }
    else if(day==5) {
        printf("Friday\n");
    }
    else if(day==6) {
        printf("Saturday\n");
    }
    else if(day==7) {
        printf("Sunday\n");
    }
    else // in case user enter invalid day number.
    {
         printf("Enter a valid day number %c !!\n",2);
    }
getch();
}
```

OUTPUT 1:

```
Enter the day number [ 1-7 ] : 3
Wednesday
```

OUTPUT 2:

```
Enter the day number [ 1-7 ] : 10
Enter a valid day number ☻ !!
```

Nested decision making statements

Nested statements created when a statement define in another statement. Nested statements allowed you to make a priority of conditions. The Outer statements will execute first, then the inner statements would be executed if the Outer condition is satisfied. These nested statements are helpful when a programmer wants to perform a task only the first task executes successfully.

Syntax:

```
if(1st condition)
{
      //1st TRUE block
      if(2nd condition)
      {
            //2nd TRUE block
      }
      else
      {
            //2nd FALSE block
      }
}
else
{
      //1st FALSE block
}
```

Here, 2nd condition will be check only when 1st condition is true.

NOTE : you can use any decision-making statement as a nested statement according to your requirement like you can use else-if ladder as nested of if-else statement.

Program 4: Write a program to find the maximum number from 3 numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
     int a,b,c,max;
clrscr();
     printf("Enter 3 numbers : ");
     scanf("%d,%d,%d",&a,&b,&c);
     //outer if-else statement
     if(a>b)
     {     //executes when a is greater than b.
           if(a>c)  //inner if-else statement
           {
                max = a;
           }
           else
           {
                max = c;
           }
     }
```

```
      else{  //executes when b is greater than a.
            if(b>c)    //inner if-else statement
            {
                  max = b;
            }
            else
            {
                  max = c;
            }
      }
      printf("Maximum is %d\n",max);
getch();
}
```

OUTPUT:

```
Enter 3 numbers : 7,3,9
Maximum is 9
```

Selection Control statements

The selection control statement selects/executes the appropriate block according to the given expression.

1. Switch statement

The switch statement executes the block(statements) by matching the case. There are multiple cases available in switch but only one case matches to the given expression.

The duplicate cases are not allowed in the switch statement. The default block available if no case could match the expression.

The break statement is necessary at the end of case's block. If you don't provide a break statement at the end of case, next all blocks will be executes unless a break statement or end of switch arrives.

Syntax :

```
switch(Expression)
      {
         case 1:
               // block 1
                  break;
         case 2:
               // block 2
                  break;
            . . .

         case N:
               // block N
                  break;
         default:
               //default block
      }
```

Program 5: Write a program using the switch statement to display the day according to the given day number.

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

```
      switch(day)
      {
            case 1:  printf("Monday \n");    break;
            case 2:  printf("Tuesday \n");   break;
            case 3:  printf("Wednesday \n"); break;
            case 4:  printf("Thursday \n");  break;
            case 5:  printf("Friday \n");    break;
            case 6:  printf("Saturday \n");  break;
            case 7:  printf("Sunday \n");    break;
           default:  printf("please, Enter the valid number !!!");
      }
getch();
}
```

OUTPUT:

```
Enter the day number : 5
Friday
```

2. Nested switch statement

A switch statement within another switch statement is known as a nested switch statement. Make sure there are different expression as many as nested switches.

Syntax :

```
switch(Expression)
      {
          case 1:
                 // block 1
                 switch(Expression)
                 {
                     …
                 }
                     break;
          case 2:
                 // block 2
                 switch(Expression)
                 {
                     …
                 }
                     break;
             . . .

          case N:
                 // block N
                 switch(Expression)
                 {
                     …
                 }
                     break;
          default:
                 switch(Expression)
                 {
                     …
                 }
                 //default block
      }
```

NOTE : Not all blocks have to define a nested switch. The programmer defines nested switch as per requirement.

Program 6: write a program to make 15 groups of students for a school task. Students can enter a roll no & div to check which group they are.

```
#include<stdio.h>
#include<conio.h>
void main()
{
      char div;
      int rollno;
clrscr();
      printf("Enter the div-rollno like [ A-42 ]: ");
      scanf("%c-%d",&div,&rollno);      // '-' separates div & rollno
```

CEC

```
 if(rollno < 0  ||  rollno > 60) // validation for rollno.
 {
      printf("invalid rollno..");
      getch( );
      exit(0);    //exit the program.
 }

switch(div)
{
  case 'A' : // if div is 'A'
             //  5 groups for div A. So as per remainder, the group will be chosen
      switch(rollno%5)
      {
        case 0: printf("Your group is (A-1) Marvel sp.\n"); break;
        case 1: printf("Your group is (A-2) Wolverines.\n");  break;
        case 2: printf("Your group is (A-3) spy men.\n"); break;
        case 3: printf("Your group is (A-4) schemes.\n"); break;
        case 4: printf("Your group is (A-5) pogies.\n"); break;
      }
      break;

flowchart :  case 'B' : // if div is 'B'
              //  5 groups for div B. So as per remainder, the group will be chosen
      switch(rollno%5)
      {
        case 0: printf("Your group is (B-1) Dories.\n"); break;
        case 1: printf("Your group is (B-2) rings of hell.\n");  break;
        case 2: printf("Your group is (B-3) Sypher.\n"); break;
        case 3: printf("Your group is (B-4) gamers.\n"); break;
        case 4: printf("Your group is (B-5) paapi.\n"); break;
      }
      break;

  case 'C' : // if div is 'C'
              //  3 groups for div C. So as per remainder, the group will be chosen
      switch(rollno%3)
      {
        case 0: printf("Your group is (C-1) Lokians.\n"); break;
        case 1: printf("Your group is (C-2) Poly-sons.\n");  break;
        case 2: printf("Your group is (C-3) mentors.\n"); break;
      }
      break;

      case 'D' : // if div is 'D'
                 // 2 groups for div D. So as per remainder, the group will be chosen
       switch(rollno%2)
```

```
            {
           case 0: printf("Your group is (D-1) The watchers.\n");break;
           case 1: printf("Your group is (D-2) Winners.\n");  break;
           }
                 break;
           default:  // if a wrong div entered.
                 printf("Invalid division....");
      }
getch();
}
```

OUTPUT 1:

```
Enter the div-rollno like [ A-42 ]: B-23
Your group is (B-4) gamers.
```

OUTPUT 2:

```
Enter the div-rollno like [ A-42 ]: A-109
invalid rollno.._
```

3. Ternary operator

Also known as conditional (?:) operator. This operator is just like if-else statement but in one line. It contains 3 parts, first one is condition and another 2 are expressions as shown in the syntax.

Syntax:

```
(condition)? expression1: expression2;
```

if condition is true then expression1 will be executed, otherwise expression2 will be executed.

Program 7: Write a program to find whether number is odd or even using conditional operator.

```
#include<stdio.h>
#include<conio.h>
void main(){
      int num;
clrscr();
       printf("Enter the number : ");
       scanf("%d",&num);
       (num%2==0)? printf("%d is even.\n",num): printf("%d is odd.\n",num);
getch();
}
```

OUTPUT:

```
Enter the number : 31
31 is odd.
```

CEC

Iteration control statement

Iteration control statements are used to repeat the block(statements) several times or until the condition is true. If the condition is always true the loop will go into infinite iteration. The repeated execution of the block could be helpful for the programmer to perform the common task for serval times. So programmers don't need to write more code.

For example, a programmer wants to display 1000 lines of "Hello world" messages . If the programmer writes all lines manually, then LOC(lines of code) will be increase. However the programmer can use loops for it without increasing LOC compare to manually way.

There are 2 categories of loops(iteration control statements).

1. Entry loops

   The condition of these loops checks before the execution of the block. So if the condition is false in the first trial(iteration), there is no chance of execution of the block. **example : for loop & while loop**

2. Exit loops

   The condition of these loops checks after the execution of the block, so even condition is false in the first trial(iteration) the block will execute at least one time. **example : do while loop**

There are 3 types of loops in c language.

1. for loop
2. while loop
3. do-while loop

1. for loop

When the number of iterations are known, it is better to use for loop. for loop contains 3 expressions in syntax : Initialization, condition & increment/decrement. These expressions separated by semicolon (;).

Syntax:

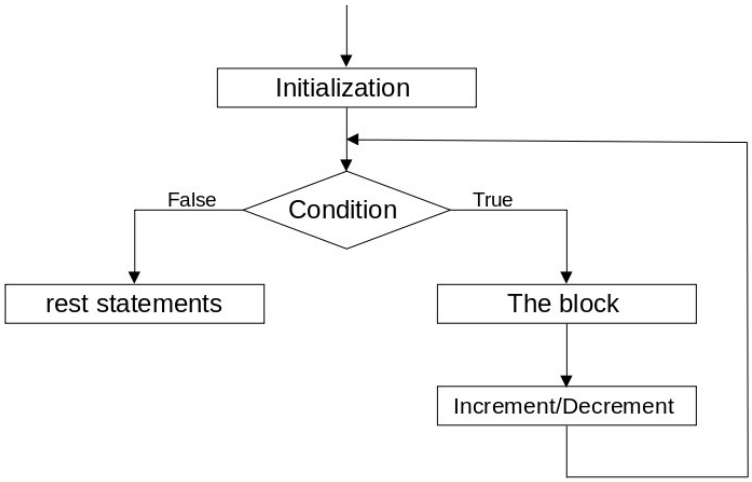
```
for(initialization; condition; increment/decrement)
{
      // the block to be executed.
}
```

NOTE : These 3 expressions are not mandatory. But if you fail to provide the right expression the loop will go into infinite iterations.

flowchart :

The compiler first executes the initialization expression, then it will go to check the condition if the condition is true then & then it will execute the block of for loop. After the execution of the block, the compiler executes the 3rd expression(increment/decrement) to update the variable, then it will go again to check the condition if true then execute the block. This process will repeat until the false condition comes (As per flowchart).

CEC



Program 8: Write a program to display 10 lines of "good morning" message using for loop.

```
#include<stdio.h>
#include<conio.h>
void main(
{
     int i;
clrscr();
      for(i=1; i<=10; i++)
      {
          textattr(13); //display message in color :)
          cprintf("Good morning %c !!\n\r",3);
      }
getch();
}
```

OUTPUT:

```
Good morning ♥ !!
Good morning ♥ !!
Good morning ♥ !!
Good morning ♥ !!
Good morning ♥ !!
Good morning ♥ !!
Good morning ♥ !!
Good morning ♥ !!
Good morning ♥ !!
Good morning ♥ !!
```

Program 9: Write a program to display all even numbers below 40 without using if-else.

```
#include<stdio.h>
#include<conio.h>
void main()
{
     int i;
clrscr();
      for(i=0; i<40; i+=2) // increment 'i' by 2
      {
          printf("%d, ",i);
      }
getch();
}
```

OUTPUT:

```
0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38,
```

Program 10: Write a program to display 10 numbers using for loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
     int i;
clrscr();
     i=1; //expression #1 : initialization
      for(;;) //without expressions
      {
          if(i>10)  // expression #2 : condition
               break;     //exit the loop
          printf("%d, ",i);

          i++; // expression #3 : increment
      }
getch();
}
```

OUTPUT:

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```

2. while loop

The while loop is used to repeat the block until the false condition comes. The number of iteration is not fixed in this loop unlike the for loop.
Usually, This loop goes into an infinite iteration because the increment/decrement expression depends upon the programmer. The programmer may forget about it, because it doesn't make a place in default syntax unlike for loop.

Syntax:

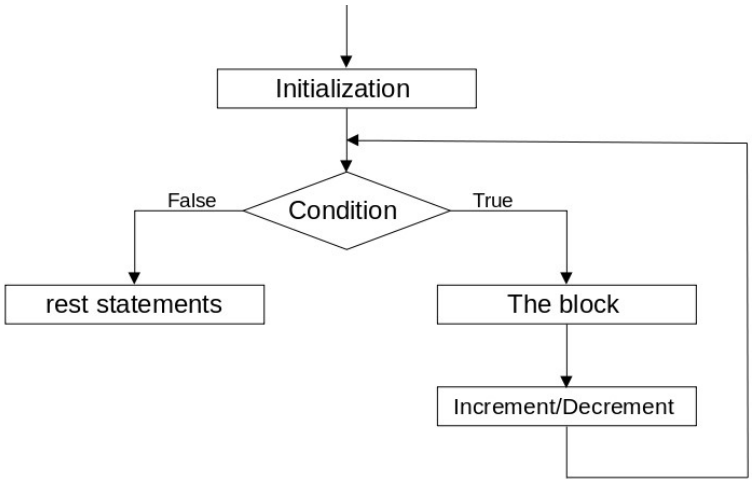
```
initialization;
while(condition)
{
      //the block to be executed.

      increment/decrement;
}
```

flowchart :

The flowchart looks like the flowchart of for loop. The execution of the loop is, First initialization expression executes however "initialization" is not the default part of the loop, the programmer can write this expression at any place before the loop.

After the initialization expression, the condition will be check. if the condition is true the block executes then increment/decrement execute. the compiler will check the again condition & if it is true then the block and increment/decrement executes. This process repeats until the false condition comes.



Program 11: write a program to display all divisible numbers by 5 below 100.

```
#include<stdio.h>
#include<conio.h>
void main()
```

CEC

```
{
     int i;
clrscr();
     i=0;       // initialization
          while(i<100)  // condition
          {
               printf("%d, ",i);
               i+=5; //increment by 5
          }
getch();
}
```

OUTPUT:

```
0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, _
```

Program 12: write a program to reverse the given number.

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

       while(num>0)
       {
          rem = num%10;        // to get last digit
          rev = rev*10+rem;    // append the rem into rev
          num = num/10;        // to erase the last digit / decrement
       }

     printf("  reverse number : %d\n",rev);
getch();
}
```

OUTPUT:

```
Enter the number : 1234
  reverse number : 4321
```

Explanation:

default value of **rev** is **0**, because when you gonna use to store a result into a variable, which variable is the operand of the operation (used in the expression). you must initialize that variable first.

let **num = 1234, rev = 0.**

rem(num%10)	rev(rev*10+rem)	num(num/10)
4 // 1234%10	4 // 0*10+4	123 //1234/10
3 // 123%10	43 // 4*10+3	12 // 123/10
2 // 12%10	432 // 43*10+2	1 // 12/10
1 // 1%10	**4321** // 423*10+1	**0** // 1/10

When the value of num becomes **zero**, the loop will terminate itself. So latest value of **rev is 4321**, which is the reverse of **1234.**

Program 13: write a program to check whether an entered number is an Armstrong number or not.

What is an Armstrong number?

if the sum of the **n^{th} power** of its digits is equal to the original number then the number is an Armstrong number. where n is a total number of digits are available in the number.

for example :

153 = $1^3+5^3+3^3$ // 3^{rd} power because total digits are 3.
= 1+125+27
= **153**

lets, write a program.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
     int rem,num,arm=0,temp,digits=0;
clrscr();
     printf("Enter the number : ");
     scanf("%d",&num);

     temp = num;
     // first, we need to calculate no of digits presents in the number
     while(temp>0)
     {
          digits++;
          temp/=10; //temp = temp/10;
     }
```

```
        temp = num;
         while(num>0)
         {
            rem = num%10;                    // to get last digit
            arm = arm+pow(rem,digits);    // adding rem^digits to arm
            num = num/10;    // to erase the last digit / decrement
         }

      if(temp == arm)
      {
           printf("%d is an Armstrong number.\n",temp);
      }
      else
      {
           printf("%d is not an Armstrong number.\n",temp);
      }
getch( );
}
```

OUTPUT 1:

```
Enter the number : 153
153 is an Armstrong number.
```

OUTPUT 2:

```
Enter the number : 721
721 is not an Armstrong number.
```

Explanation:

let **arm = 0**, **num = 153** so **digits = 3**

rem(num%10)	arm(arm+rem^{digits})	num(num/10)
3 // 153%10	27 // 0+3^3	15 //153/10
5 // 15%10	152 // 27+5^3	1 // 15/10
1 // 1%10	**153** // 152+1^3	**0** // 1/10

Here, we use temp as recovery of **num**, num will modify by num/10 expression. so for comparing the **arm,** we need the original number.

math.h allows us to use pow() function.

Syntax:

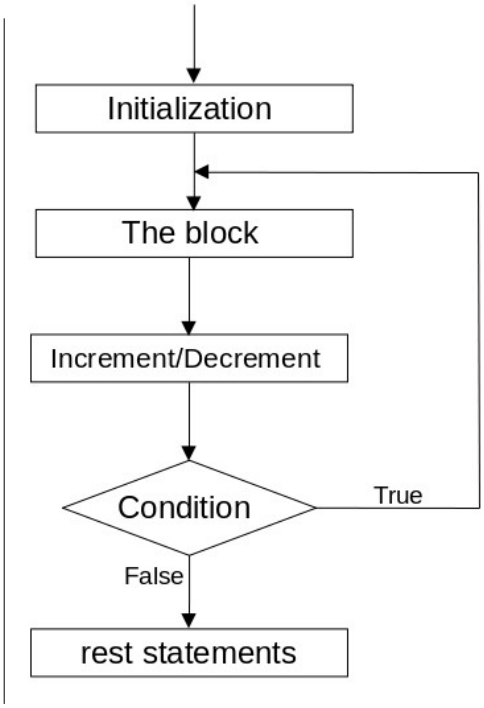
pow(x,y); // x^y

3. do-while loop

This is an exit loop because the condition will be checked after the execution of the block. If the condition is true then the loop will execute again, Like the while loop the initialization expression is independent part of the syntax, means programmer can define anywhere before the loop.

Syntax:

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



flowchart :

The initialization expression is independent part of the syntax. this expression execute only once. After the initialization expression the block will execute. In this loop increment/decrement comes before the condition. if the condition is true the block will execute again. So even condition is false in 1st iteration, the loop will execute at least one time. that's benefit of exit loop.

Program 14: write a program to display table of given number using do while loop.

```
#include<stdio.h>
#include<conio.h>
void main()
{
      int num;
      int i=1; //initialization
clrscr();
       printf("Enter the number : ");
       scanf("%d",&num);
       do{
          printf("%2d x %2d = %d\n",num,i,num*i); // %2d for 2 digit formatting
          i++;                //increment
       }while(i<=10);         //condition
getch();
}
```

OUTPUT:

Program 15: write a program make a calculator using switch statement, so that user can input data as an operation like 7+2 .Ask user for input till user want to perform operations.

```
#include<stdio.h>
#include<conio.h>
void main()
{
      int a,b;
      char choice,ch;
clrscr();
      do{
            printf("\nEnter the operation : ");
            scanf("%d%c%d",&a,&ch,&b);

            switch(ch)
            {
              case '+': printf("%d + %d = %d \n",a,b,a+b); break;
              case '-': printf("%d - %d = %d \n",a,b,a-b); break;
              case '*': printf("%d * %d = %d \n",a,b,a*b); break;
              case '/': printf("%d / %d = %d \n",a,b,a/b); break;
              case '%': printf("%d %% %d = %d \n",a,b,a%b); break;
              default : printf("invalid operator. %c\n",1);
            }
      flushall(); // to flush the buffer of previous inputs
          printf("do you want to repeat[y/n] : "); // asking user to execute again.
          scanf("%c",&choice);

      }while(choice=='y' || choice=='Y');
      printf("exit....");
getch();
}
```

OUTPUT:

```
Enter the operation : 7+2
7 + 2 = 9
do you want to repeat[y/n] : y

Enter the operation : 5%2
5 % 2 = 1
do you want to repeat[y/n] : y

Enter the operation : 7=23
invalid operator.
do you want to repeat[y/n] : n
exit...._
```

Nested loops

A loop in another loop is called nested loop. You can create many nested loops as you want to create. if outer loop iterates **n** times & inner loop iterators **m** times then the block of inner loop will execute **m*n** times. For creating nested statements, they don't need to be same as outer/inner.

Program 16: write a program to display following pattern.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
     int i,j,n;
clrscr();
     printf("Enter the number : ");
     scanf("%d",&n);

     for(i=0;i<n;i++)          // for rows, outer loop
     {
          for(j=0;j<n;j++)     //for columns, inner loop
          {
               printf(" * ");
          }
          printf("\n"); // new line after all columns filled, for creating new row.
     }
```

CEC

```
getch();
}
```

OUTPUT :

Flow control statements

The flow control statement defines & controls the flow of execution of the program. The control statement can skip, repeat or break the block.

1. break statement
2. continue statement
3. goto statement
4. exit statement

(return statement is also a flow control statement, but we will discuss it later.)

1. break statement

The break statement helps to exit a block of the loop/switch. This statement skip the all iteration of loop then start the execution of the outer statements (rest statements).

Program 17: write a program to demonstrates use of break statement.

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

       for(; i<=10; i++)
       {
           if(i==6)
           {
               break; // break statement
           }
          printf("line %d.\n",i);
       }
       printf("\nhello world"); //rest statements
getch();
}
```

OUTPUT :

2. continue statement

The continue statement is used to skip the rest of the statements of the block & resume remaining iterations. Whenever continue statement executes, rest of the statements of block will skip. The continue statement may execute many time as per condition unlike the break statement.

Program 18: write a program to demonstrates use of continue statement.

```
#include<stdio.h>
#include<conio.h>
void main()
{
     int i;
clrscr();
      for(i=1;i<=10;i++)
      {
         if(i==6)
          {
               continue; //continue statement
          }
         printf("line %d.\n",i);
      }
      printf("\nHello World"); //rest statements
getch();
}  // NOTE : line 6 is skipped.
```

OUTPUT:

3. goto statement

goto statement is used to transfer the control of the execution at a specific place in program. The specific place is noted by a **label**. The label could be any name as an identifier. There are 2 types of goto statement.

1. forward goto statement
2. backward goto statement

1. forward goto statement

The forward goto statement is helpful to skip ahead statements. This statement may no need of condition.

Syntax:

```
...
      goto LABEL;
...
...
      LABEL:
...
```

Program 19: write a program to demonstrates use of forward goto statement.

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
           printf("Line 1\n");
           printf("Line 2\n");
    goto code;                 //transfer the control to label 'code'
           printf("Line 3\n");
           printf("Line 4\n");
           printf("Line 5\n");
   code:
           printf("Line 6\n");
           printf("Line 7\n");
getch();
}
```

OUTPUT:

```
Line 1
Line 2
Line 6
Line 7
```

2. **backward goto statement.**

This statement is used to repeat the statements. This statement must use with a condition, otherwise the infinite iterations will be create.

Syntax:

```
      ...
            LABEL:
      ...
      ...
            goto LABEL;
```

CEC

Program 20: write a program to demonstrates use of forward goto statement.

```
#include<stdio.h>
#include<conio.h>
void main()
{
     char ch;
clrscr();

     printf("Line 1\n");
     printf("Line 2\n");
    code:
     printf("\tLine 3\n");
     printf("\tLine 4\n");
     printf("\tLine 5\n");
    flushall();
          printf("wanna display again ? [y/n] : ");
          scanf("%c",&ch);
                if(ch=='y' || ch=='Y')
                    goto code;

     printf("Line 6\n");
     printf("Line 7\n");
getch();
}
```

OUTPUT:

```
Line 1
Line 2
        Line 3
        Line 4
        Line 5
wanna display again ? [y/n] : y
        Line 3
        Line 4
        Line 5
wanna display again ? [y/n] : y
        Line 3
        Line 4
        Line 5
wanna display again ? [y/n] : n
Line 6
Line 7
```

4. exit statement

The exit statement is used to exit the program. Sometimes is necessary to exit the program or let the program crash. we can pass an integer argument in it as a number. That could be any number however 0 & 1 are best for it. 1 means forcefully exit sign & 0 means normally exit from the program.

Program 21: write a program to display the given number. if the given number is -ve then program should be close after exit message display.

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

     if(num<0)
     {
          printf("%d is -ve..\nextit...\n",num);
          getch();
          exit(10);
     }
     printf("Entered number is %d\n",num);
getch();
}
```

OUTPUT 1:

```
Enter the number : 23
Entered number is 23
```

OUTPUT 2:

```
Enter the number : -21
-21 is -ve..
exit...
```