Unit-5

Control Statements

Introduction

In C programs, statements are executed sequentially in the order in which they appear in the program, But sometimes we may want to use a condition for executing only a part of a program. Also many situations arise where we may want to execute some statements several times. Control statements enable us to specify the order in which the various instructions in the program are to be executed. This determines the flow of control. Control statements define how the control is transferred to other parts of the program. C language supports four types of control statements, which are given below.

1. if……else
2. goto
3. switch
4. loop
    a. while
    b. do…….while
    c. for

Compound Statement or Block

A compound statement or a block is a group of statements enclosed within a pair to curly braces { }. The statements inside the block are executed sequentially. The general form is-

```
{
  statement1;
  statement2;
  ……
  ……
}
```

If Statement

The if statement is used to express conditional expressions. The general form of if statement is;

For only one statement we don't need braces.

```
if(condition)
statement;
```

For more than one statements we need braces;

if(condition)

{

statement1;

statement2;

……..

statementN;

}

Example

/* To find out commission*/

```
#include<stdio.h>
void main()
{
 float sales, commission;
 printf("Enter total sales made:);
 scanf("%f" &sales);
 if(sales >= 8000)
       commission = 0.08 * sales;
 printf("commission is: %f" commission);
 }
```

If …… else statement

This is a bi-directional conditional control statement. This statement is used to test a condition and take one of the two possible actions. If the condition is true then a single statement or a block of statements is executed (one part of the program), otherwise another single statement or a block of statements is executed (other part of the program). Recall that in C, any nonzero value is regarded as true while zero is regarded as false.

Syntax :

For single line

if(condition)

statement1;

else

statement2;

For multiline

if(condition)

{

statement;

…….;

}

else

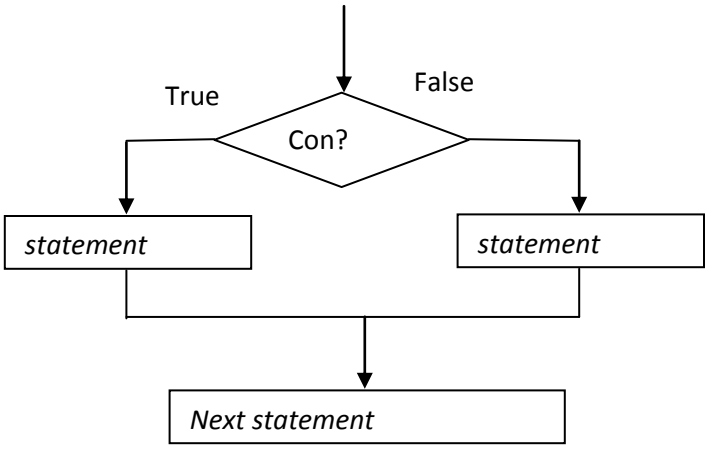
{

statement;

……;

}

Flowchart



Write a program to find whether a given number is even or odd.

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

Nesting if……else statement

If we declare another if…else statement in the block of if or the else block. This is known as nesting if….else statements.

Syntax:

```
if(condition)
 {
    if(condition)
        statement;
         else
       statement;
 }
else
   {
    if(condition)
        Statement;
     else
      statement;
}
```

Write a program to find largest among any three given numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c,large;
clrscr();
printf("Enter three numbers\n");
scanf("%d%d%d",&a,&b,&c);
if(a>b)
{
    if(a>c)
    large=a;
    else
    large=c;
}
else
{
    if(b>c)
```

```
    large=b;
    else
    large=c;
}
printf("Largest
number=%d",large);
getch();
}
```

else…if ladder

This is the 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 frequently used in programs and is also known as else if ladder.

Syntax:

if(condition 1)

statement 1;

else if(condition 2)

statement 2;

else if(condition 3)

statement 3;

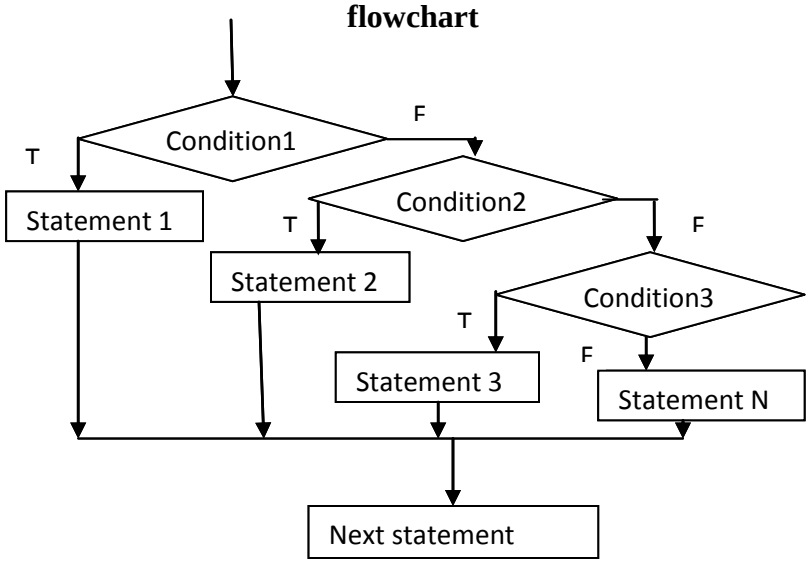
:

:

else

statement N;

flowchart



Write a program to read the marks of four subjects then find total, percentage and division according to given condition.

Percentage	**Division**
p ≥ 80	Distinction
80 > p ≥ 70	First division
70 > p ≥ 50	Second division
50 >p ≥ 40	Third division
Otherwise	Fail.

Assume each subject carrying 100 full marks and student must secure greater or equal to 40 in each subject for division.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int s1,s2,s3,s4,total;
float per;
clrscr();
printf("Enter marks in 4 different subjects\n");
scanf("%d%d%d%d",&s1,&s2,&s3,&s4);
total=s1+s2+s3+s4;
printf("\nTotal marks:%d",total);
if(s1>=40&&s2>=40&&s3>=40&&s4>=40)
{
printf("\nResult:PASS");
per=total/4.0;
printf("\nPercentage:%f",per);
if(per>=80)
printf("\nDivision:DISTINCTION");
else if(per>=70)
printf("\nDivision:FIRST");
else if(per>=50)
printf("\nDivision:SECOND");
else
printf("\nDivision:THIRD");
}
else
printf("\nResult:FAIL");
getch();
}
```

goto

This is an unconditional control statement that transfer the flow of control to another part of the program .

The goto statements can be as-

goto label;

……..

……

label:

statements;

……

…….

Write a program to read a positive number from user.

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

```
printf("Number is negative so
re-enter a number\n");
goto again;
}
getch();
}
```

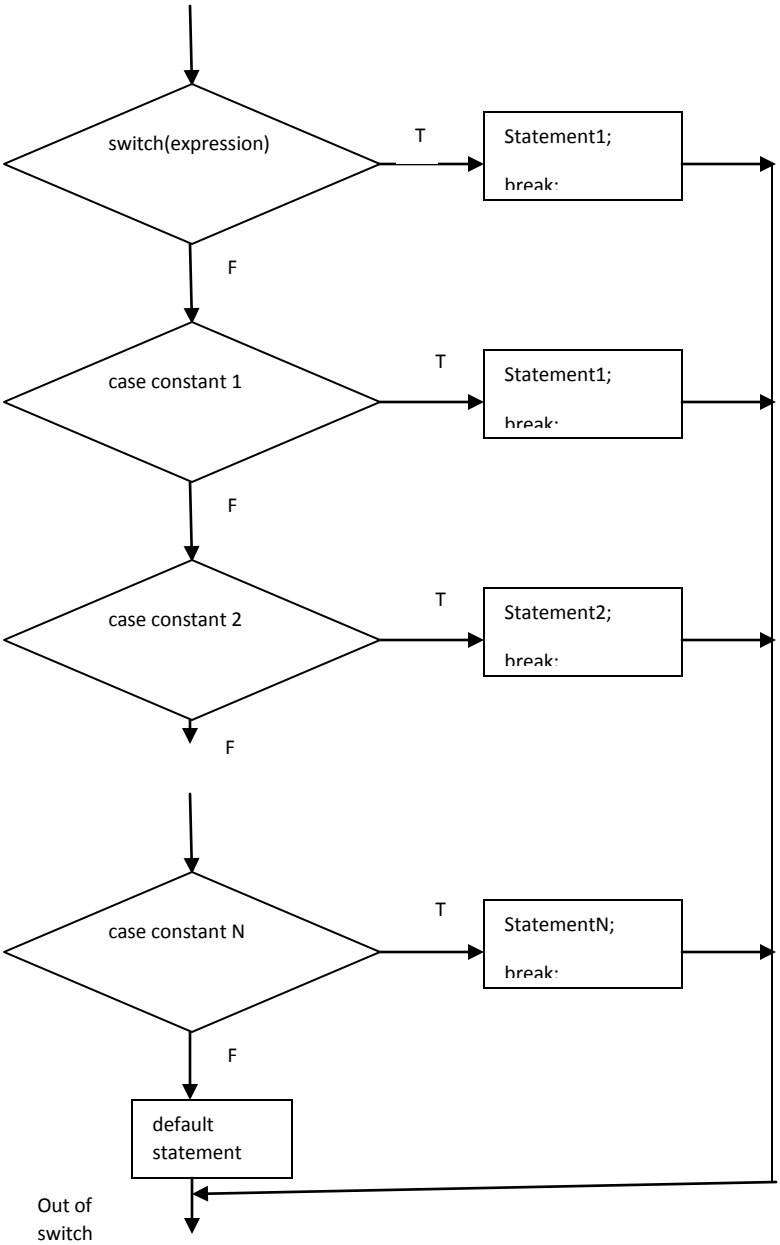
Switch

This is a multi-directional condition control statements. Sometimes there is a need in program to make choice among number of alternatives. For making this choice, we use the switch statements. This can be written as:-

Syntax:

```
switch (expression)
  {
  case constant1:
            statement 1;
            break;
  case constant2:
            statement 2;
            break;
  case constant3:
            statement 3;
           break;
  default:
            statement;
  }
```

Flowchart:



Write a program to print the day of the week according to the number entered 1 for Sunday ……..7 for Saturday.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int num;
clrscr();
printf("Enter a number\n");
scanf("%d",&num);
switch(num)
{
case 1:
printf("Sunday");
break;
case 2:
printf("Monday");
break;
case 3:
printf("Tuesday");
break;
case 4:
printf("Wednesday");
break;
case 5:
printf("Thusday");
break;
case 6:
printf("Friday");
break;
case 7:
printf("Saturday");
break;
default:
printf("Wrong input");
}
getch();
}
```

Write a program to perform arithmetic calculation based on the operators entered by user.

+ for addition, - for subtraction, * for multiplication, / for division and % for remainder.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
char op;
clrscr();
printf("Enter operator\n");
scanf("%c",&op);
printf("Enter two numbers\n");
scanf("%d%d",&a,&b);
switch(op)
{
case '+':
printf("Sum =%d",a+b);
break;
case '-':
printf("Difference =%d",a-b);
break;
case '*':
printf("Product =%d",a*b);
break;
case '/':
printf("Quotient =%d",a/b);
break;
case '%':
printf("Remainder =%d",a%b);
break;
default:
printf("Wrong operator");
}
getch();
}
```

looping

Repetition/looping means executing the same section of code more than once until the given condition is true. A section of code may either be executed a fixed number of times, or while condition is true. C provides three looping statements:

1. for loop
2. while loop

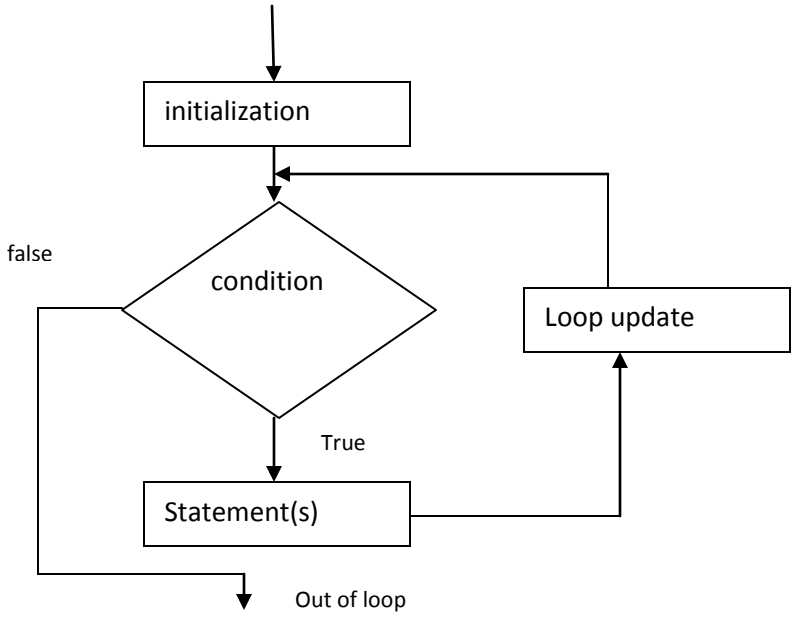
3. do while loop

For loop

For statement makes programming more convenient to count iterations of a loop and works well where the number of iterations of the loop is known before the loop entered. The syntax is as follows:

```
for (initialization; test condition; loop update)
        {
                Statement(s);
        }
```

Flowchart



The main purposed is to repeat statement while condition remains true, like the while loop. But in addition, for provides places of specify an initialization instruction and an increment or decrement of the control variable instruction. So this loop is specially designed to perform a repetitive action with counter.

For loop has the four parts and it works as:

1. Initialization is executed. Generally it is an initial value setting for a counter variable. This is executed only once.
2. Condition is checked, if it is true the loop continues, otherwise the loop finishes and statement is skipped.
3. Statements(s) are /are executed. As usual, it can be either a single instruction or a block of instructions enclosed within curly brackets {}.

4. Finally, whatever is specified in the increment or decrement of the control variable field is executed and the loop gets back to step 2.

Exercises

1. Write a program to print 1 to 10.
2. Write a program to print your name for 20 times.
3. Write a program to display factorial of any given number.
4. Write a program to display multiplication table for any given number.
5. Write a program to display following Fibonacci series upto n terms.
   0 1 1 2 3 ……..n terms
6. Write a program to check whether a given number is prime or composite.
7. Write a program to find the sum of odd numbers upto n.
8. Write a program to find the sum of natural numbers upto n.
9. Write a program to display the sum of squares of natural numbers upto n.
10. Write a program to find the sum of cubes of first 10 numbers.
11. Write a program to evaluate the sum of n terms of the following series.
    1/1 + 1/3 + 1/5 + 1/7 + ……..

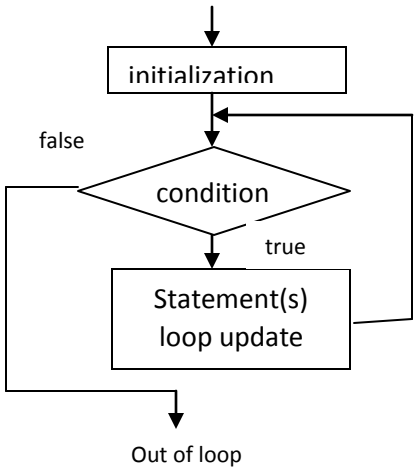
While loop

A while loop statement in C programming language repeatedly executes a target statement as long as a given condition is true.

The syntax of a **while** loop in C programming language is

```
while(condition)
{
   statement(s);
}
```

Flowchart



Here, **statement(s)** may be a single statement or a block of statements.. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

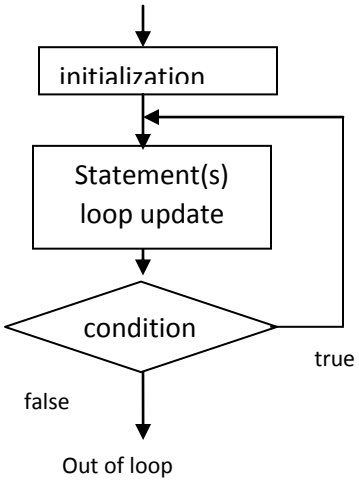
do while

In C, do...while loop is very similar to while loop. Only difference between these two loops is that, in while loops, test expression is checked at first but, in do...while loop code is executed at first then the condition is checked. So, the code are executed at least once in do...while loops.

The syntax of a **do while** loop in C programming language is

```
do
{
   statement(s);
}while(condition);
```

flowchart



Difference between while and do while statement

While	**do……while**
1. It is entry control loop.	1. It is an exit control loop.
2. Test condition is evaluated before the loop is executed	2. Test condition is evaluated after the loop is executed.

3. It use keyword *while.*	3. It uses keywords *do* and *while.*
4. Loop is not terminated with semicolon.	4. Loop is terminated with semicolon.
5. It doesn't execute the body of loop until and unless the condition is true.	5. Body of the loop is always executed at least once since the test condition is not checked until end of the loop.

6. Example
 	
`#include<stdio.h>`
`void main()`
`{`
`int i;`
`i=1;`
`while(i<=5)`
`{`
`printf("%d\n",i);`
`}`
	6. Example
	
	`#include<stdio.h>`
	`void main()`
	`{`
	`int i;`
	`i=1;`
	`do`
	`{`
	`printf("%d\n",i);`
	`} while(i<=5);`

Exercise

1. Write a program to find the sum of digits of any given number.
2. Write a program to reverse any given number.
3. Write a program to check whether a given number is palindrome or not.

break statement

break statement is used inside loops and switch statements. Sometimes it becomes necessary to come out of loop before its termination. In such a situation, break statement is used to terminate the loop. When break statement is encountered, loop is terminated and the control is transferred to the statement immediately after the loop.

continue statement

continue statement is used when we want to go to the next iteration of the loop after skipping some statements of the loop. It is generally used with a condition. When continue statement is encountered all the remaining statements after continue in the current iteration are not executed and loop continues with the next iteration.

Difference between break and continue statement

break statement	**continue statement**
1. break statement is used to terminate the control from the switch case as well as in the loop.	1. continue statement is used to by-pass 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 by-passes single pass of the loop.
3. It uses keyword *break.*	3. It uses keyword *continue.*
4. Syntax: break;	4. Syntax: continue;

5. Example:
`#include<stdio.h>`
`#include<conio.h>`
`void main()`
`{`
`int i;`
`clrscr();`
`for(i=1;i<=10;i++)`
`{`
`if(i==5)`
`break;`
`printf("%d\n",i);`
`}`
`getch();`
`}`
 	
`outputs:`
`1`
`2`
`3`
`4`
	5. Example:
	`#include<stdio.h>`
	`#include<conio.h>`
	`void main()`
	`{`
	`int i;`
	`clrscr();`
	`for(i=1;i<=10;i++)`
	`{`
	`if(i==5)`
	`continue;`
	`printf("%d\n",i);`
	`}`
	`getch();`
	`}`
	
	`outputs:`
	`1`
	`2`
	`3`
	`4`
	`6`
	`7`
	`8`
	`9`
	`10`

exit()

exit() is used to terminate the whole program. In other words it returns control to the operating system.

Nested loop

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 loop.

Exercise

1. Write a program to display prime numbers from 100 to 500.
2. Write a program to display multiplication table from 1 to 20.
3. Write a program to display factorial of first 10 integer numbers.