

# GBEST 204 Programming in C

## Module 1

### **Part – III**

### **Control flow statements**

# Control statements - Syllabus

- if, if-else, nested if,
- switch,
- while,
- do-while,
- for,
- break and continue
- nested loops

# Control flow statements

- Conditional branching
  - Decision making or selection statements
  - example - `if`, `if...else`, nested `if`, `if.....else if`  
`switch.....case`
- Unconditional branching
  - `goto`, `break`, `continue`, `exit`
- Loop control structures
  - `while`, `for`, `do.....while`

# Decision making statements

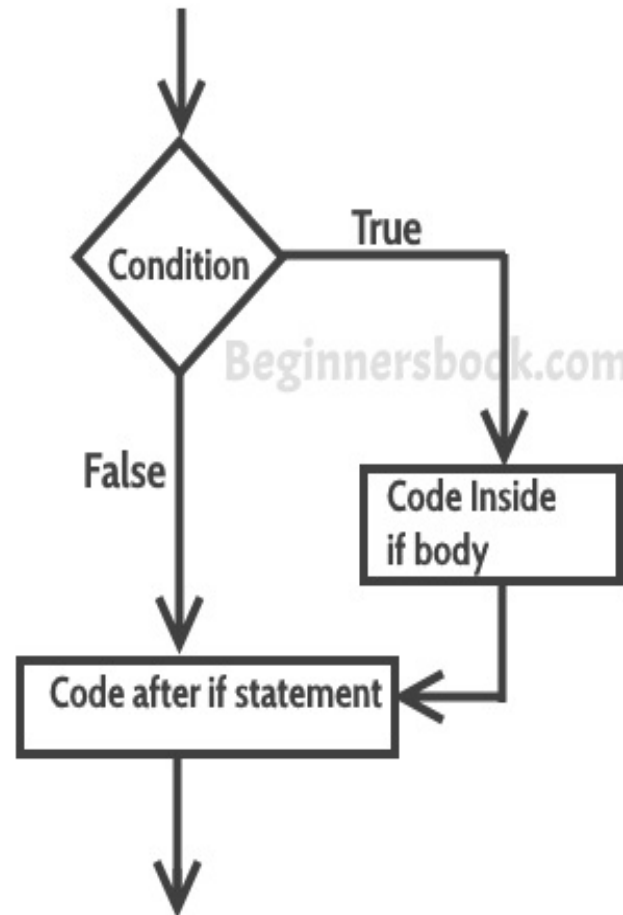
- Helps the programmer to **transfer** the control from one part to other parts of the program.
- It helps to determine the **flow** of the program
- In some cases it is essential
  - ☐ To alter the flow of program
  - ☐ To test the logical condition
  - ☐ To control the flow, as per selection

# Conditional branching - if

## Syntax

```
if(condition)
{
    statement_1;
    statement_2;
}
```

## Flow chart



# Programming - if

```
#include<stdio.h>
int main()
{
    int age;
    printf("Enter the age : ");
    scanf("%d", &age);
    if (age >= 20)
        printf("Eligible to vote");
    return 0;
}
```

Output 1:  
Enter the age : 10

Output 2:  
Enter the age : 22  
Eligible to vote

# Programming - if

Write a program to find if  $a$  is less than  $b$

```
#include<stdio.h>
int main()
{
    int a,b;
    printf("Enter any two numbers : ");
    scanf("%d %d",&a, &b);
    if (a<b)
        printf("a is less than b\n");
    return 0;
}
```

Output 1:

Enter any two numbers : 2

5

a is less than b

Output 2:

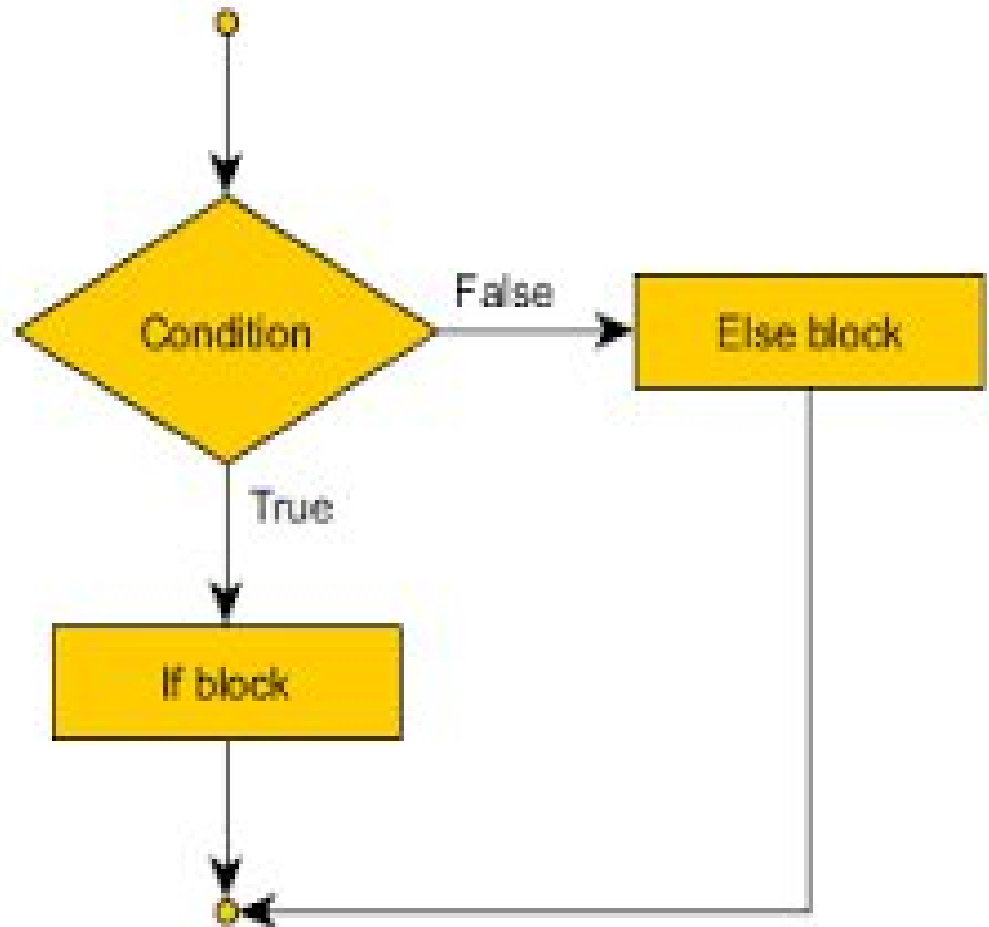
Enter any two numbers : 6 4

# Program – if....else

## Syntax

```
if(condition)
{
    statement_1;
}
else
{
    statement_2;
}
```

## Flow chart





# Programming – if...else

Write a program to find if a number is even

```
#include<stdio.h>
int main()
{
    int num;
    printf("Enter any positive number: ");
    scanf("%d", &num);
    if (num%2==0)
        printf("The given number %d is EVEN\n", num);
    else
        printf("The given number %d is ODD\n", num);
    return 0;
}
```

Output 1:

Enter any positive number : 26  
The given number 26 is EVEN

Output 2:

Enter any positive number : 95  
The given number 95 is ODD

# Try

- WAP to find the roots of quadratic equation

# Answer – roots of quadratic eqn

```
#include<math.h>
#include<stdio.h>
int main()
{
float a,b,c,d,r1,r2;
printf("Enter any three numbers: ");
scanf("%f %f %f",&a,&b,&c);
d=b*b-(4*a*c);
if(d<0)
    printf("roots are imaginary\n");
else
{
    r1 =(-b+sqrt(d))/(2*a);
    r2 =(-b-sqrt(d))/(2*a);
    printf("The roots are : %f %f\n",r1,r2);
}
return 0;
}
```

Output 1:

Enter any three numbers : 1 5 4  
The roots are : -1.000000 -4.000000

Output 2:

Enter any three numbers : 1 4 6  
The roots are imaginary

# Programming – if....else (Try)

If hours worked is less than or equal to 40, regular pay is calculated by multiplying hours worked by rate of pay and overtime pay is 0.

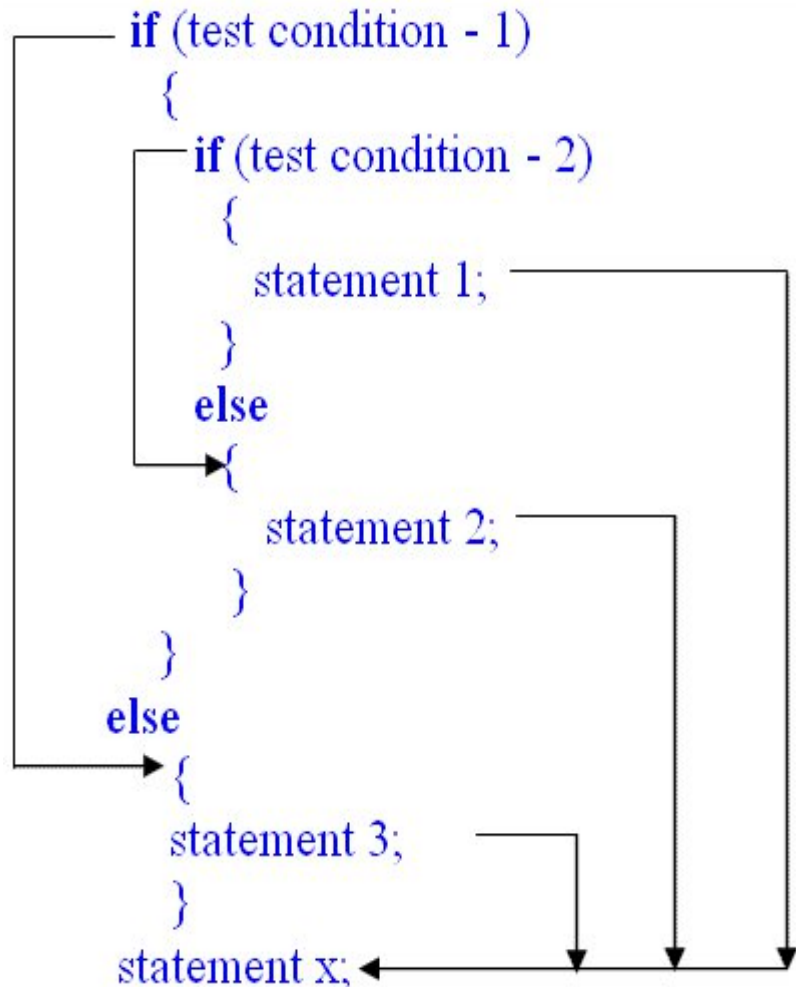
If hours worked is greater than 40, regular pay is calculated by multiplying 40 by the rate of pay and overtime pay is calculated by multiplying the hours *in excess of* 40 by the rate of pay by 1.5.

Gross pay is calculated by adding regular pay and overtime pay.

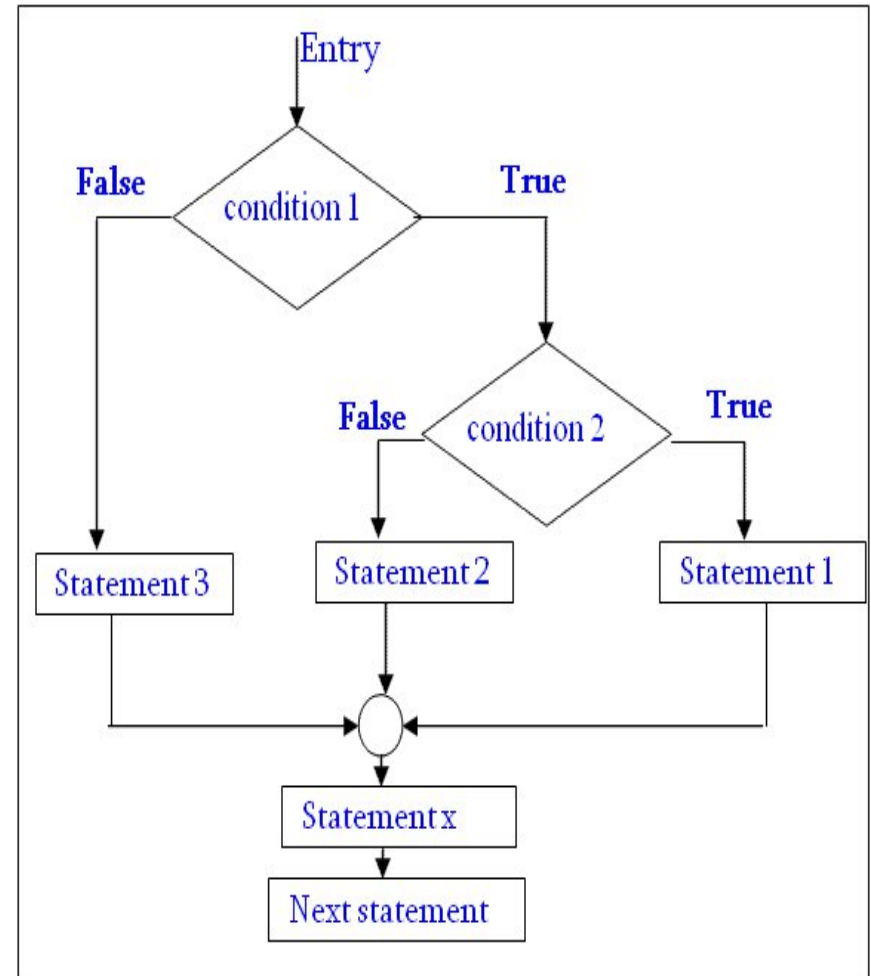
Write a program to **prompt for hours worked and rate of pay** and to print regular pay, overtime pay, and gross pay.

# Conditional branching– Nested if....else

## Syntax



## Flowchart



# Conditional branching– Nested if....else

The mark obtained by a student is given as input. The student gets a division as per the following rules:

Mark above or equal to 60 - First division ;

Mark between 50 and 59 - Second division ;

Mark between 40 and 49 - Third division ;

Mark less than 40 - Fail.

Write a C program to calculate the division obtained by the student.

# Program - Nested if....else

```
#include<stdio.h>
int main( )
{
    int mark;
    printf("Enter the mark:");
    scanf( "%d", &mark);
    if(mark >= 60)
        printf("First
division\n");
```

Output 1:  
Enter the mark : 56  
Second division

```
    else
    {
        if(mark >= 50)
            printf("Second division\n");
        else
        {
            if(mark >= 40)
                printf("Third division");
            else
                printf("Fail\n" ) ;
        }
    }
    return 0;
}
```

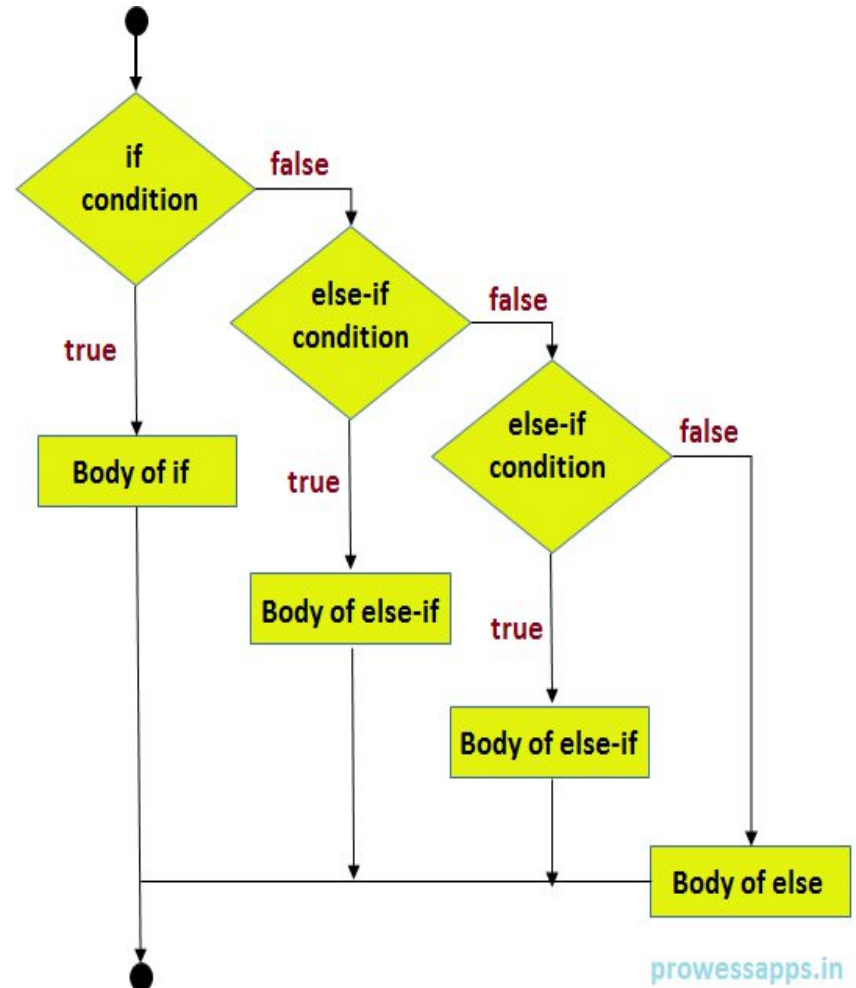
Output 2:  
Enter the mark : 87  
First division

# if.....elseif (if Ladder statement)

## Syntax

```
if(condition1)
{
    code to be executed if condition1 is true
}
else if(condition2)
{
    code to be executed if condition2 is true
}
else if(condition3)
{
    code to be executed if condition3 is true
}
else
{
    code to be executed if all the conditions
        are false
}
```

## Flowchart





# Conditional branching– if....else if

The mark obtained by a student is given as input. The student gets a division as per the following rules:

Mark above or equal to 60 - First division ; Mark between 50 and 59 - Second division ; Mark between 40 and 49 - Third division ; Mark less than 40 - Fail . Write a program to calculate the division obtained by the student.

```
#include<stdio.h>
int main( )
{
    int mark;
    printf("Enter the mark :");
    scanf("%d", &mark);
    if(mark >= 60)
        printf("First division\n");
    else if(mark >= 50)
        printf("Second division\n");
    else if(mark >= 40)
        printf("Third division\n");
    else
        printf("Fail\n" ) ;
    return 0;
}
```

Output 1:

Enter the mark : 56  
Second division

Output 2:

Enter the mark : 87  
First division

# Programs of “if...else” (Try)

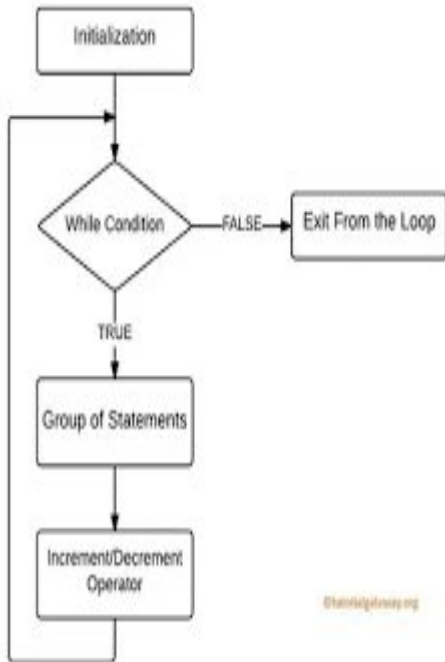
If cost price and selling price of an item is input through the keyboard, WAP to determine whether the seller has made profit or incurred loss.

Also determine how much profit he made or loss be incurred.

# Loop Statements - Syntax

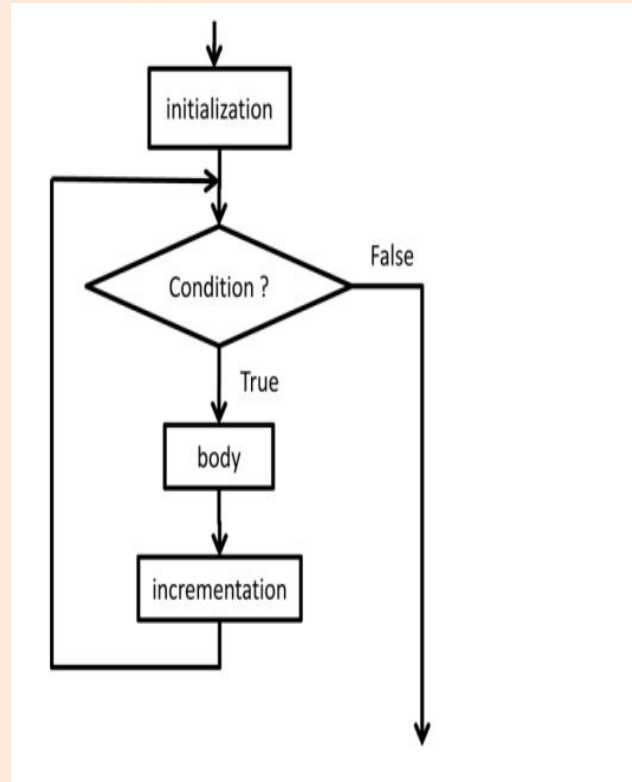
while

```
initialize;
while(condition)
{
    statement;
    increment or decrement;
}
```



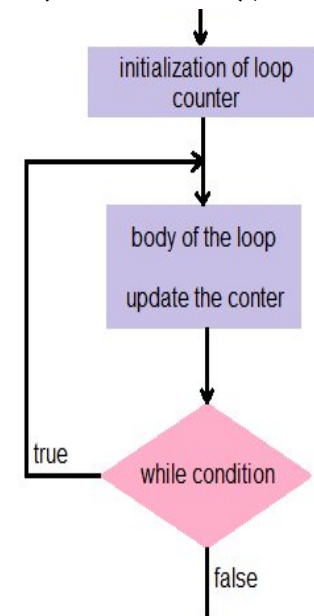
for

```
for (initialize; condition; inc or dec)
{
    statement;
}
```



## do..while

```
initialize;
do
{
statement;
increment or decrement;
}
while(condition);
```



# Loop Statements - while

Write a program to print first N natural numbers

```
#include<stdio.h>
int main()
{
    int i,num;
    printf("Enter any positive number :");
    scanf("%d",&num);
    i=1;
    while(i<=num)
    {
        printf("%d\t",i);
        i=i+1;
    }
    return 0;
}
```

Output 1:  
Enter any positive number : 6  
1 2 3 4 5 6

# Loop Statements - for

Write a program to print first N natural numbers

```
#include<stdio.h>
int main()
{
    int i,num;
    printf("Enter any positive number:");
    scanf("%d",&num);
    for(i=1;i<=num;i++)
        printf("%d\n",i);
    return 0;
}
```



Note the  
semicolon ;

Output 1:  
Enter any positive number : 5  
1  
2  
3  
4  
5

# Loop Statements – do...while

Write a program to print first N natural numbers

```
#include<stdio.h>
int main()
{
    int i,num;
    printf("Enter any positive number :");
    scanf("%d",&num);
    i=1;
    do
    {
        printf("%d\t",i);
        i=i+1;
    } while (i<=num) ;
    return 0;
}
```

Output 1:

Enter any positive number : 4  
1 2 3 4

# WAP to print the digits of a number

```
#include<stdio.h>
int main()
{
    int num, remainder, sum;
    printf("Enter any positive number : ");
    scanf("%d",&num);
    printf("The digits of the number %d are : \n",num);
    while(num>0)
    {
        remainder = num % 10;
        printf("%d \t", remainder);
        num = num/10;
    }
    return 0;
}
```

Output:

Enter any positive number : 1068

The digits of the number 1068 are :

8 6 0 1

# Programs of loop

## WAP to print the sum of the digits of a number

```
#include<stdio.h>
int main()
{
    int num, remainder, sum;
    printf("Enter any positive number : ");
    scanf("%d", &num);
    sum = 0;
    while(num>0)
    {
        remainder = num % 10;
        num = num /10;
        sum = sum + remainder;
    }
    printf("\n The sum of the digits of  number is %d\n", sum);
    return 0;
}
```

Output:

Enter any positive number : 567

The sum of the digits of number is : 18



# Write a program to reverse a number

```
#include <stdio.h>
int main()
{
    int n, reverse, remainder;
    reverse = 0;
    printf("Enter an integer: ");
    scanf("%d", &n);
    while (n > 0)
    {
        remainder = n % 10;
        reverse = reverse * 10 + remainder;
        n /= 10;
    }
    printf("The Reversed number is: %d", reverse);
    return 0;
}
```

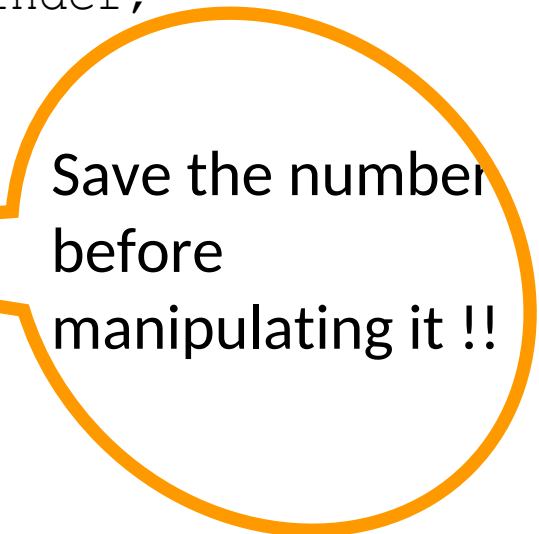
Output:

Enter an integer: 4567

The Reversed number is : 7654

# Check if number is a palindrome

```
#include <stdio.h>
int main()
{
    int num, Original_Num, reverse, remainder;
    reverse = 0;
    printf("Enter an integer: ");
    scanf("%d", &num);
    Original_Num = num;
    while (num > 0)
    {
        remainder = num % 10;
        reverse = reverse * 10 + remainder;
        num = num/10;
    }
    printf("The Reversed number = %d\n", reverse);
    if(Original_Num == reverse)
        printf("The given number %d is a PALINDROME \n", Original_Num);
    else
        printf("The given number %d is NOT A PALINDROME\n",Original_Num);
    return 0;
}
```



Save the number before manipulating it !!

# Output of palindrome

Enter an integer: **4567**

The Reversed number = 7654

The given number 4567 is **NOT A PALINDROME**

Enter an integer: **1234321**

The Reversed number = 1234321

The given number 1234321 is a **PALINDROME**

# Program – prime number

- Check to see if a number has more than 2 factors, other than 1, and the same number.
- If there are no other factors other than 1 and the number itself, then it can be termed as a prime number, otherwise, it is a composite number.
- WAP to print prime numbers between 200 and 300

# Answer - Prime number

```
#include<stdio.h>
#include<math.h>
int main()
{
    int i, number, squareroot;
    int flag = 1;
    printf("Enter any positive number :");
    scanf("%d", &number);
    squareroot = sqrt(number);
    for(i = 2; i < squareroot; i++)
    {
        if((number % i) == 0)
        {
            flag = 0;
        }
    }
    if (flag == 1)
        printf("%d is prime number.", number);
    else
        printf("%d is NOT a prime number.", number);
    return 0;
}
```

Output 1:

Enter any positive number :62  
62 is NOT a prime number.

Output 2:

Enter any positive number :79  
79 is a prime number.

# Prime numbers (1 - 500)

2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	71
73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173
179	181	191	193	197	199	211	223	227	229
233	239	241	251	257	263	269	271	277	281
283	293	307	311	313	317	331	337	347	349
353	359	367	373	379	383	389	397	401	409
419	421	431	433	439	443	449	457	461	463
467	479	487	491	499					

# switch case

- The switch statement is used when the selection is based on the value of single variable or simple expression called controlling expression.
- The value of the expression may be **int** or **char**; neither `string` nor `double`
- **default** is optional. Good programming practice - declare a default
- The switch case is often used in **menu** selection

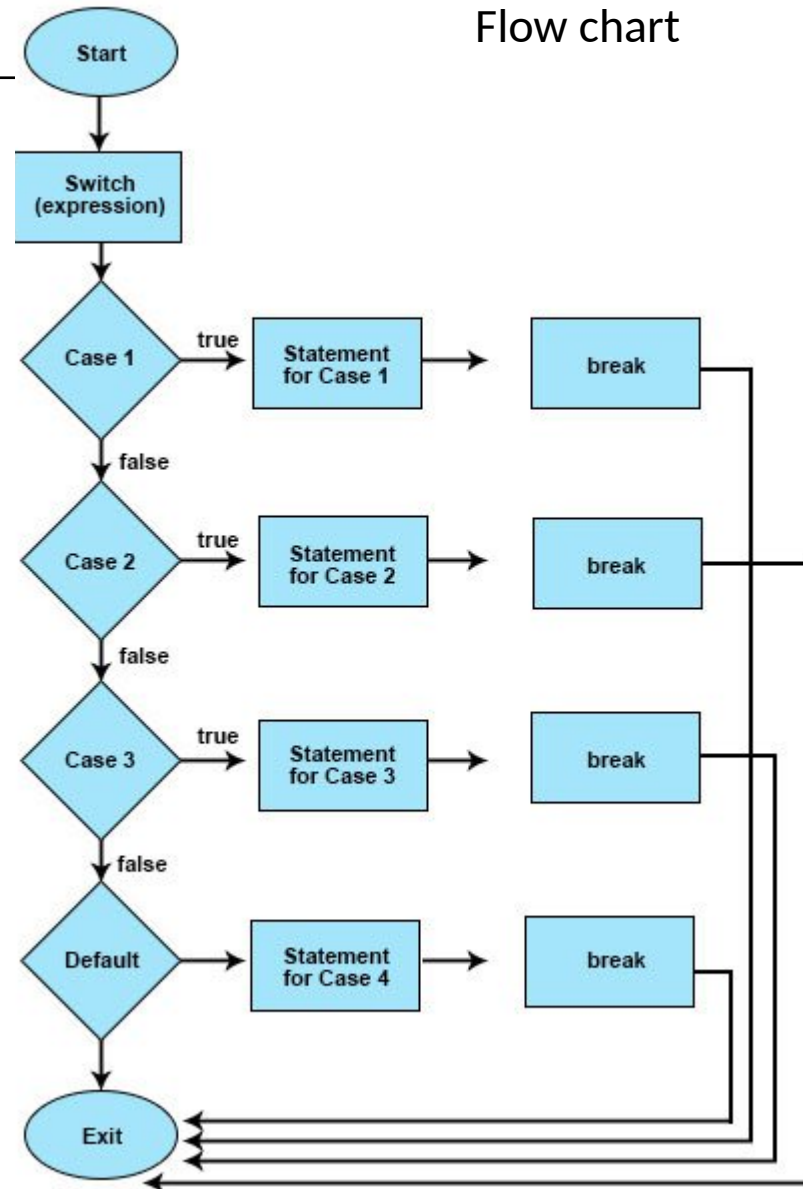
# Unconditional branching - switch

Syntax

```
switch(expression)
{
    case value_1 :
        do this;
        break;
    case value_2 :
        do this;
        break;
    case value_3 :
        do this;
        break;
    default :
        do this;
}
```

Note: case ends with colon

Flow chart





# Unconditional branching - Switch

WAP to create a calculator which can do addition, subtraction, multiplication and division.

```
#include<stdio.h>
int main()
{
    int a, b, c;
    printf("Enter two nos. ");
    scanf("%d %d", &a, &b);
    printf("1: \t Add\n
           2: \t Subtract\n
           3: \t Multiply\n
           4: \t Divide\n
           \n Enter your choice: \t");
    scanf("%d", &c);
```

```
    switch(c)
    {
        case 1:
            printf("%d\n", a+b);
            break;
        case 2:
            printf("%d\n", a-b);
            break;
        case 3:
            printf("%d\n", a*b);
            break;
        case 4:
            printf("%f\n", (float) a/b);
            break;
        default:
            printf("\nInvalid Input\n");
    }
    return 0;
}
```

# Output

Enter two nos. 56

4

1: Add

2: Subtract

3: Multiply

4: Divide

Enter your choice: 4

14.000000

```
#include<stdio.h>
int main ()
{
int num1, num2;
float result;
char ch;
printf("Enter first number : ");
scanf("%d", &num1);
printf("Enter second number: ");
scanf("%d", &num2);
printf("Choose operator to
perform operations : ");
scanf(" %c", &ch);
result = 0;
switch(ch)
{
case '+':
    result = num1 + num2;
    break;
```

```
case '-':
    result = num1 - num2;
    break;
case '*':
    result = num1 * num2;
    break;
case '/':
    result = (float)num1/num2;
    break;
default:
    printf("Invalid Operation\n");
}
printf("Result: %d %c %d = %f
\n", num1, ch, num2, result);
return 0;
}
```

# Output

Enter first number : 65

Enter second number: 3

Choose operator to perform operations : /

Result:  $65 / 3 = 21.666666$

# Unconditional branching - goto

- The programs become unreliable, unreadable, and hard to debug. They obscure the flow of control.
- The usage of the **goto** should be avoided as it usually violets the normal flow of execution.

Syntax 1	Syntax 2
goto <i>label</i> ; statement_1; statement_2; <i>label</i> : statement_3;	<i>label</i> : statement_3; statement_1; statement_2; goto <i>label</i> ;

# Program using goto

## WAP to print the multiplication table of 10

```
#include <stdio.h>
int main()
{
    int num,i;
    num = 10;
    i=1;
table:
    printf("%d x %2d = %3d\n",num,i,num*i);
    i++;
    if(i<=10)
        goto table;
}
```

Output

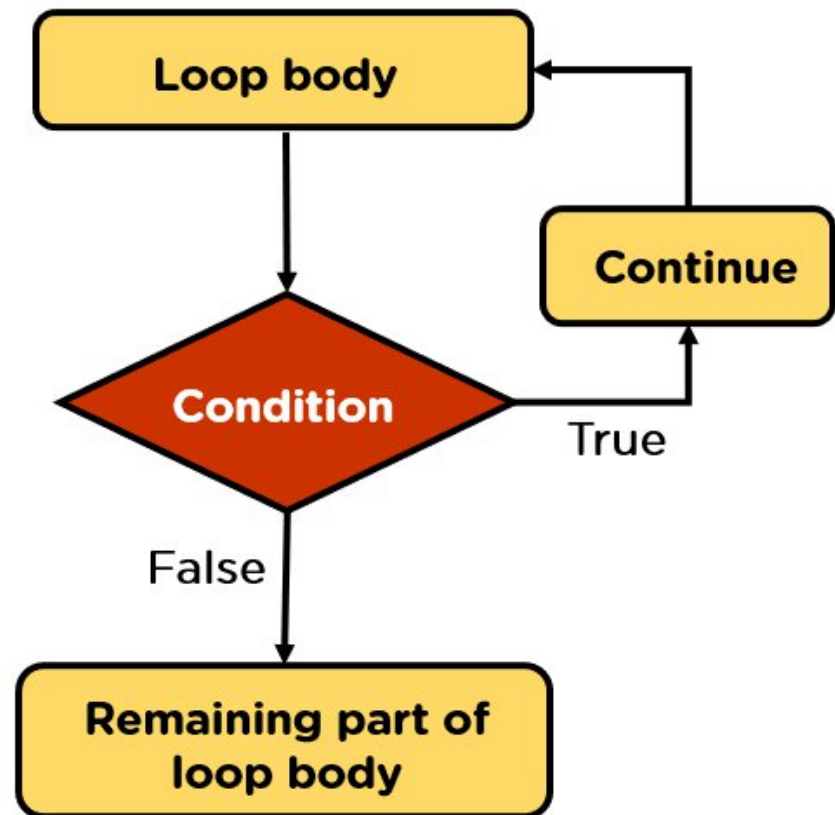
10 x 1 =	10
10 x 2 =	20
10 x 3 =	30
10 x 4 =	40
10 x 5 =	50
10 x 6 =	60
10 x 7 =	70
10 x 8 =	80
10 x 9 =	90
10 x 10 =	100

# Unconditional branching - continue

Syntax

`continue;`

Flow chart



# Program using 'continue'

WAP to print odd numbers between 10 and 30

```
#include<stdio.h>
int main()
{
    int i;
    for(i=10;i<30;i++)
    {
        if(i%2==0)
        {
            continue;
        }
        printf("%d\t", i);
    }
    return 0;
}
```

True

False

Output:

11 13 15 17 19 21 23 25 27 29

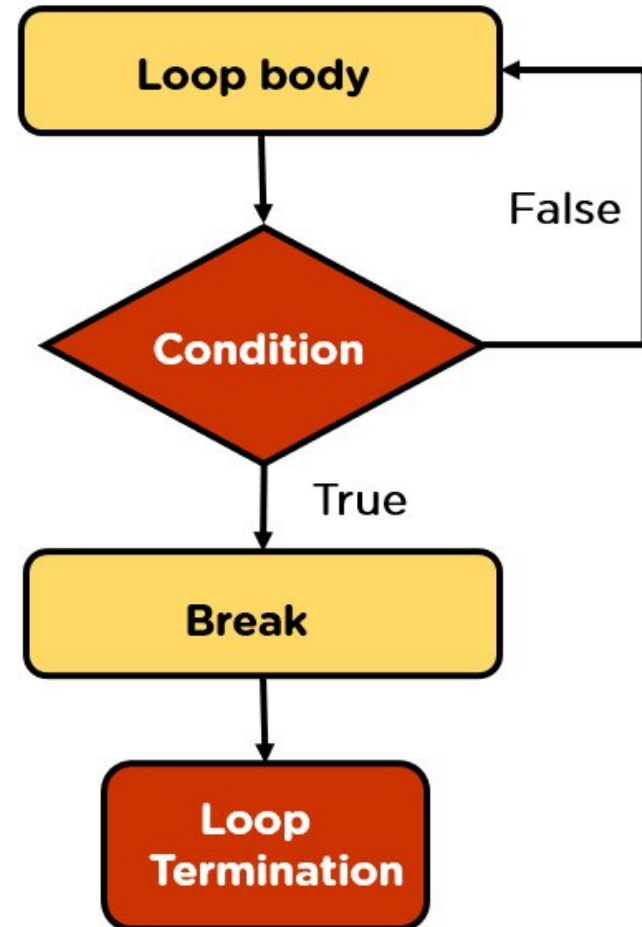


# Unconditional branching - Break

Syntax

`break;`

Flow chart



# Programs using 'break'

WAP to print the entered values of 'x', that does not exceed 100. However break out if negative number is detected.

```
#include<stdio.h>
int main()
{
    int x;
    printf("\n Enter any number : \t");
    do
    {
        scanf("%d",&x);
        if(x<0)
        {
            printf("Error - negative value for x\n");
            break;
        }
    } while(x<=100);
    return 0;
}
```

True

False

Output :

Loop will continue to execute as long as the current value of x does not exceed 100. However the computation will break if negative values are entered.

# Output - break

Enter any number :	Enter any number :
74 34 56 100 95 200	5 0 67 -89 Error - negative value for x
Remark: The entered value is greater than 100	Remark: The entered value is negative

# Difference between break and continue

## –compare previous example

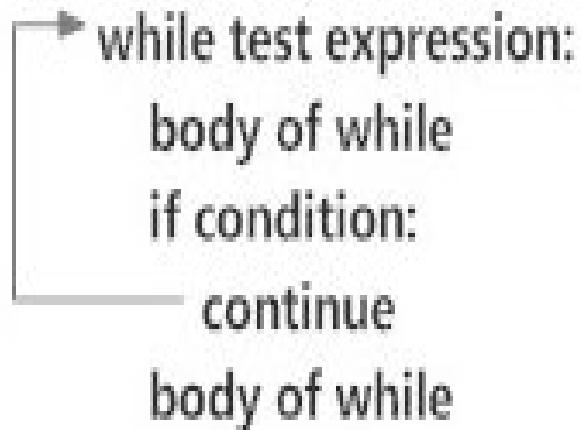
```
#include<stdio.h>
int main()
{
    int x;
    printf("\n Enter any number: \n");
    do
    {
        scanf ("%d", &x) ;
        if (x<0)
        {
            printf("\n Error: negative value for x\n");
            continue;
        }
    }while (x<=100) ;
    return 0;
}
```

### Output:

The processing of current value will be bypassed if the value of x is negative. Execution of the loop will then continue with the next pass.

# Difference between break and continue

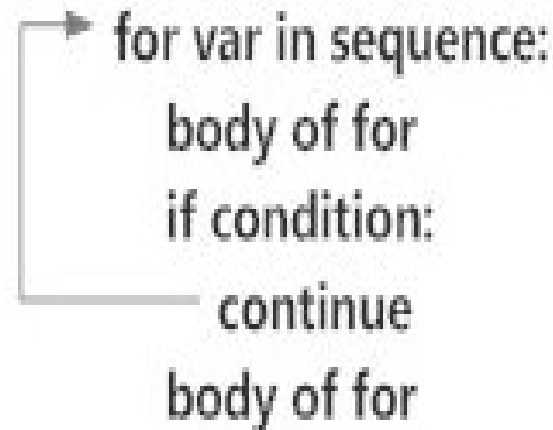
- Break causes a loop to end
- Continue stops the current iteration and begin the next iteration



```
graph TD; A[while test expression:] --> B[body of while]; B --> C[if condition:]; C --> D[continue]; D --> A;
```

while test expression:  
body of while  
if condition:  
continue  
body of while

statement(s)



```
graph TD; A[for var in sequence:] --> B[body of for]; B --> C[if condition:]; C --> D[continue]; D --> A;
```

for var in sequence:  
body of for  
if condition:  
continue  
body of for

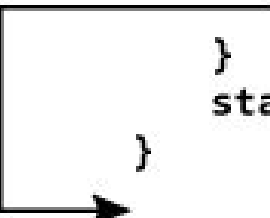
statement(s)

# Output -continue


Output 1	Output 2
Enter any number :	Enter any number :
74 34 56 100 95 200	5 0 -89 Error - negative value for x 43 7 : .
Remark: The entered value is greater than 100	Remark: Continues till values $> 100$ is entered

# Difference between while, do-while and for

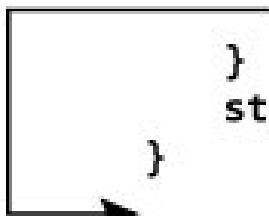
```
while (test expression) {  
    statement/s  
    if (test expression) {  
        break;  
    }  
    statement/s  
}
```



```
do {  
    statement/s  
    if (test expression) {  
        break;  
    }  
    statement/s  
}  
while (test expression);
```



```
for (initial expression; test expression; update expression) {  
    statement/s  
    if (test expression) {  
        break;  
    }  
    statements/  
}
```



# exit

The C library function **exit(int status)** terminates the calling process immediately.

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    printf("Start of the program....\n");
    printf("Exiting the program....\n");
    exit(0);
    printf("End of the program....\n");
    return(0);
}
```

Output:

```
Start of the program....
Exiting the program....
```



# Try

- WAP to find factorial of a number
- WAP to find if a number is strong number

A strong number is a number in which the sum of the factorial of the digits is equal to the number itself.

$$\begin{aligned}\text{Eg:- } 145 &= 1! + 4! + 5! = 1 + 24 + 120 \\ &= 145\end{aligned}$$