

# **Problem Solving and Programing**

User input: “Hello !” // include library of standard input and output commands

```
#include<iostream.h>
```

```
using namespace std;
```

```
int main()
```

```
{ // Begin main function
```

```
    string name; // create variable called name
```

```
    cout << "What is your name?";
```

```
    cin >> name; // get name from user
```

```
    cout << "Hello "; // output "Hello "
```

```
    cout << name << "!\n"; // output "<name >"
```

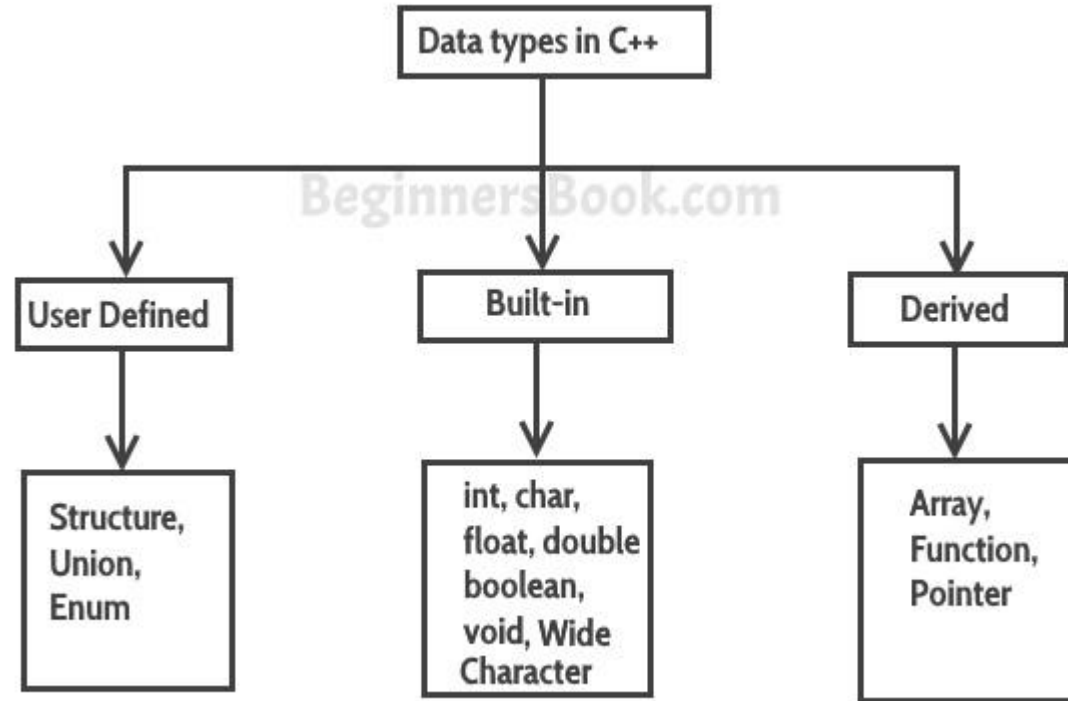
```
    return 0; // end program
```

```
} // End main function
```

What is your name? Sree

Hello Sree

# Data types in c++



# Pre-defined Data Types

Type	Description	Length
char	Single letter character entries only	8 bits
int (short)	Integer values only	16 bits
long	Accepts higher range of integer values than the int data type	32 bits
float	Supports decimal point notation	32 bits
double	Supports decimal point notation with greater precision than Float data type	64 bits
double float	Similar to a float, with greater precision to the right of the decimal point	128 bits
void	Does not return any value.	N/A

# Declaring variables

data\_type      variable name

[ = initial value ]

```
int num ;
```

```
int num1 = 3;
```

```
char alpha = 'G' ;
```

```
float float_no ;
```

```
float floa = 33.33;
```

```
double d_no ;
```

```
double doub = 1234.121212121 ;
```

optional



# Character strings

```
char str_string[6] ;
```

```
char str_string[30]= “ Nice Day “ ;
```

Variable name should start with letter(a-zA-Z) or underscore (\_)

## Valid

Age

\_age

Age

## Invalid

1age

In variable name, no special characters allowed other than underscore (\_).

## Valid

\_age

age\_

## Invalid

age\_\*

+age

Variables are case sensitive.

age and Age are different, since variable names are case sensitive.

Variable name can be constructed with digits and letters.

Age1

Age2

Variable name should not be a C/c++ keyword

Int    main

# Change the limits of a data type

Use modifiers

signed

unsigned

short

long

```
long int var1 ;
```

```
Unsigned short int var2 ;
```

`long int a;`       $\longrightarrow$  occupies 4 bytes of memory space



`2 + 2`       $\longrightarrow$  4 bytes

`long double b;`       $\longrightarrow$  occupies 10 bytes of memory space



`2 + 8`       $\longrightarrow$  10 byte



C Data types / storage Size	Range
char / 1	−127 to 127
int / 2	−32,767 to 32,767
float / 4	1E−37 to 1E+37 with six digits of precision
double / 8	1E−37 to 1E+37 with ten digits of precision
long double / 10	1E−37 to 1E+37 with ten digits of precision
long int / 4	−2,147,483,647 to 2,147,483,647
short int / 2	−32,767 to 32,767
unsigned short int / 2	0 to 65,535
signed short int / 2	−32,767 to 32,767
long long int / 8	−(2 <sup>power</sup> (63) − 1) to 2 <sup>(power)</sup> 63 − 1
signed long int / 4	−2,147,483,647 to 2,147,483,647
unsigned long int / 4	0 to 4,294,967,295
unsigned long long int / 8	2 <sup>(power)</sup> 64 − 1

Data Type (Keywords)	Description	Size	Typical Range
<i>int</i>	Integer.	4 bytes	-2147483648 to 2147483647
<i>signed int</i>	Signed integer. Values may be negative, positive, or zero.	4 bytes	-2147483648 to 2147483647
<i>unsigned int</i>	Unsigned integer. Values are always positive or zero. Never negative.	4 bytes	0 to 4294967295
<i>short</i>	Short integer.	2 bytes	-32768 to 32767
<i>signed short</i>	Signed short integer. Values may be negative, positive, or zero.	2 bytes	-32768 to 32767
<i>unsigned short</i>	Unsigned short integer. Values are always positive or zero. Never negative.	2 bytes	0 to 65535
<i>long</i>	Long integer.	4 bytes	-2147483648 to 2147483647
<i>signed long</i>	Signed long integer. Values may be negative, positive, or zero.	4 bytes	-2147483648 to 2147483647
<i>unsigned long</i>	Unsigned long integer. Values are always positive or zero. Never negative.	4 bytes	0 to 4294967295

## Access qualifiers

- Const
- Volatile
- Mutable

constant declaration

keyword          const

```
const float pi = 3.146 ;  
const char three = '3';  
const int number = 5 ;  
const double d_number = 8738478.9898 ;
```

Escape sequence	Description
<code>\n</code>	Newline. Position the screen cursor to the beginning of the next line.
<code>\t</code>	Horizontal tab. Move the screen cursor to the next tab stop.
<code>\r</code>	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
<code>\a</code>	Alert. Sound the system bell.
<code>\\</code>	Backslash. Used to print a backslash character.
<code>\'</code>	Single quote. Use to print a single quote character.
<code>\"</code>	Double quote. Used to print a double quote character.

# Types of Operators in C++

- Basic Arithmetic Operators
- Assignment Operators
- Auto-increment and Auto-decrement Operators
- Logical Operators
- Comparison (relational) operators
- Bitwise Operators
- Ternary Operator

# Basic Arithmetic Operators

Basic arithmetic operators are: +, -, \*, /, %

```
#include <iostream>
using namespace std;
int main()
{
    int num1 = 240;
    int num2 = 40;
    cout<<"num1 + num2: "<<(num1 + num2)<<endl;
    cout<<"num1 - num2: "<<(num1 - num2)<<endl;
    cout<<"num1 * num2: "<<(num1 * num2)<<endl;
    cout<<"num1 / num2: "<<(num1 / num2)<<endl;
    cout<<"num1 % num2: "<<(num1 % num2)<<endl;
    return 0;
}
```

## Output:

```
num1 + num2: 280
num1 - num2: 200
num1 * num2: 9600
num1 / num2: 6
num1 % num2: 0
```

# Assignment Operators

Assignments operators in C++ are: =, +=, -=, \*=, /=, % =

```
#include <iostream>
using namespace std;
int main()
{
    int num1 = 240;
    int num2 = 40;
    num2 = num1;
    cout<<"= Output: "<<num2<<endl;
    num2 += num1;
    cout<<"+= Output: "<<num2<<endl;
    num2 -= num1;
    cout<<"-= Output: "<<num2<<endl;
    num2 *= num1;
    cout<<"*= Output: "<<num2<<endl;
    num2 /= num1;
    cout<<"/= Output: "<<num2<<endl;
    num2 %= num1;
    cout<<"%= Output: "<<num2<<endl;
    return 0;
}
```

## Output:

= Output: 240

+= Output: 480

-= Output: 240

\*= Output: 57600

/= Output: 240

%= Output: 0

# Auto-increment and Auto-decrement Operators

```
#include <iostream>
using namespace std;
int main()
{
    int num1 = 240;
    int num2 = 40;
    num1++;
    num2--;
    cout<<"num1++ is: "<<num1<<endl;
    cout<<"num2-- is: "<<num2;
    return 0;
}
```

## Output:

num1++ is: 241  
num2-- is: 39



# Logical Operators

- Logical Operators are used with binary variables. They are mainly used in conditional statements and loops for evaluating a condition.
- Logical operators in C++ are: `&&`, `||`,

```
#include <iostream>
using namespace std;
int main()
{
    bool b1 = true;
    bool b2 = false;
    cout<<"b1 && b2: "<<(b1&&b2)<<endl;
    cout<<"b1 || b2: "<<(b1||b2)<<endl;
    cout<<"!(b1 && b2): "<<!(b1&&b2);
    return 0;
}
```

## Output:

b1 && b2: 0

b1 || b2: 1

!(b1 && b2): 1

# Relational operators

We have six relational operators in C++: ==, !=, >, <, >=, <=

## Bitwise Operators

There are six bitwise Operators: &, |, ^, ~, <<, >>

## Ternary Operator

variable num1 = (expression) ? value if true : value if false

```
int main()
{
int a = 9; int b = 4;
cout<<" a > b\n", a > b;
cout<<"a >= b\n", a >= b;
cout<<"a <= b\n", a <= b;
cout<<"a < b \n", a < b;
cout<<"a == b \n", a == b;
cout<<"a != b \n", a != b;
}
```

a > b	1
a >= b	1
a <= b	0
a < b	0
a == b	0
a != b	1

## Bitwise Operators in C/C++

In C, following 6 operators are bitwise operators (work at bit-level)

- **& (bitwise AND)** Takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
- **| (bitwise OR)** Takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 any of the two bits is 1.
- **^ (bitwise XOR)** Takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.
- **<< (left shift)** Takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.
- **>> (right shift)** Takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.
- **~ (bitwise NOT)** Takes one number and inverts all bits of it

```
/* C Program to demonstrate use of bitwise operators */
```

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

```
int main()
```

```
{
```

```
    unsigned char a = 5, b = 9; // a = 5(00000101), b = 9(00001001)
```

```
    printf("a = %d, b = %d\n", a, b);
```

```
    printf("a&b = %d\n", a&b); // The result is 00000001
```

```
    printf("a|b = %d\n", a|b); // The result is 00001101
```

```
    printf("a^b = %d\n", a^b); // The result is 00001100
```

```
    printf("~a = %d\n", a = ~a); // The result is 11111010
```

```
    printf("b<<1 = %d\n", b<<1); // The result is 00010010
```

```
    printf("b>>1 = %d\n", b>>1); // The result is 00000100
```

```
    return 0;
```

```
}
```

a = 5, b = 9

a&b = 1

a|b = 13

a^b = 12

~a = 250

b = 4

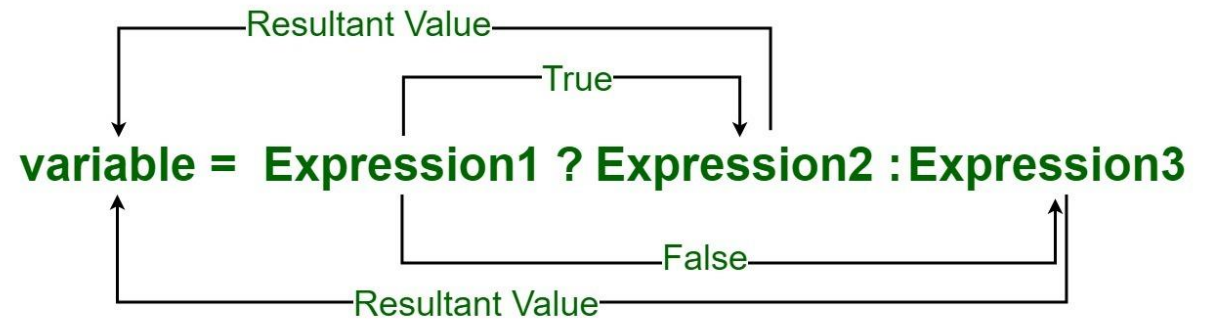
```
#include <iostream>
using namespace std;
int main()
{
int num1, num2;
num1 = 99;
num2 = (num1 == 10) ? 100: 200;
cout<<"num2: "<<num2<<endl;
num2 = (num1 == 99) ? 100: 200;
cout<<"num2: "<<num2;
return 0;
}
```

## Output:

num2: 200

num2: 100

### Conditional or Ternary Operator (?:) in C/C++



```
int a = 10, b = 20, c;  
if (a < b)  
{  
    c = a;  
}
```

```
else  
{  
    c = b;  
}
```

```
cout<< c;
```

10

```
int a = 10, b = 20, c;  
c = (a < b) ? a : b;  
cout<< c;
```

10

```
int a = 1, b = 2, ans;  
if (a == 1)  
{  
    if (b == 2)  
    {  
        ans = 3;  
    }  
    else  
    {  
        ans = 5;  
    }  
}  
else  
{  
    ans = 0;  
}  
cout<< ans;
```

**3**

```
int a = 1, b = 2, ans;  
ans = (a == 1 ? (b == 2 ? 3 : 5) : 0);  
cout<< ans;
```

**3**



# C++ Modifier

C++ allows the **char**, **int**, and **double** data types to have modifiers preceding them

The data type modifiers are listed here

- signed
- unsigned
- long
- short

```
#include <iostream>
using namespace std;
/* This program shows the difference between * signed and unsigned integers. */
int main()
{
    short int i; // a signed short integer
    short unsigned int j; // an unsigned short integer
    j = 50000;
    i = j;
    cout << i << " " << j; return 0;
}
```

-15536  
50000

```
#include <iostream>
using namespace std;
int main()
{
    int a = 10, b = 3;
    float ans = (a/b);
    cout<<fixed<<setprecision(3);
    cout << (a/b) << endl;
    cout << ans << endl;
    return 0;
}
```

# Type Conversion

A type cast is basically a conversion from one type to another. There are two types of type conversion:

**Implicit Type Conversion** Also known as ‘automatic type conversion’. Done by the compiler on its own, without any external trigger from the user.

- Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
- All the data types of the variables are upgraded to the data type of the variable with largest data type.

```
bool -> char -> short int -> int ->  
  
unsigned int -> long -> unsigned ->  
  
long long -> float -> double -> long double
```

It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

// An example of implicit conversion

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int x = 10; // integer x
```

```
    char y = 'a'; // character c
```

```
    // y implicitly converted to int. ASCII
```

```
    // value of 'a' is 97
```

```
    x = x + y;
```

```
    // x is implicitly converted to float
```

```
    float z = x + 1.0;
```

```
    cout << "x = " << x << endl
```

```
        << "y = " << y << endl
```

```
        << "z = " << z << endl;
```

```
    return 0;
```

```
}
```

x = 107

y = a

z = 108

**Explicit Type Conversion:** This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.

**Converting by assignment:** This is done by explicitly defining the required type in front of the expression in parenthesis. This can be also considered as forceful casting.

(type) expression	<pre>// C++ program to demonstrate // explicit type casting #include &lt;iostream&gt; using namespace std; int main() {     double x = 1.2;      // Explicit conversion from double to int     int sum = (int)x + 1;      cout &lt;&lt; "Sum = " &lt;&lt; sum;      return 0; }</pre>	Sum = 2
-------------------	---	---------

**Conversion using Cast operator:** A Cast operator is an **unary operator** which forces one data type to be converted into another data type.

- 1.C++ supports four types of casting: [Static Cast](#)
- 2.Dynamic Cast
- 3.[Const Cast](#)
- 4.[Reinterpret Cast](#)

```
#include <iostream>
using namespace std;
int main()
{
    float f = 3.5;

    // using cast operator
    int b = static_cast<int>(f);

    cout << b;
}
```

## Number System and Base Conversions

A number N in base or radix b can be written as:

$$(N)_b = d_{n-1} d_{n-2} \text{ --- } d_1 d_0 . d_{-1} d_{-2} \text{ --- } d_{-m}$$

In the above,  $d_{n-1}$  to  $d_0$  is integer part, then follows a radix point, and then  $d_{-1}$  to  $d_{-m}$  is fractional part.

Base	Representation
2	Binary
8	Octal
10	Decimal
16	Hexadecimal

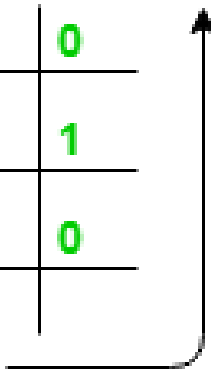


## 1. Decimal to Binary

$(10.25)_{10}$

Integer part :

2	10	0
2	5	1
2	2	0
	1	



$$(10)_{10} = (1010)_2$$

Fractional part

:

$$0.25 \times 2 = 0.50$$

$$0.50 \times 2 = 1.00$$



$$(0.25)_{10} = (0.01)_2$$

**Note:** Keep multiplying the fractional part with 2 until decimal part 0.00 is obtained.

$$(0.25)_{10} = (0.01)_2$$

**Answer:**  $(10.25)_{10} = (1010.01)_2$

# Decimal to Octal

		Remainder	
8	473		
8	59	1	MSD ↑ LSD
8	7	3	
	0	7	

Reading the remainders from bottom to top,  
 $473_{10} = 731_8$

# Decimal to Hexadecimal

		Remainder	
16	423		
16	26	7	↑
16	1	A	
	0	1	

Reading the remainders from bottom to top we get,  
 $423_{10} = 1A7_{16}$

Decimal Number	4-bit Binary Number	Hexadecimal Number
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

## 2. Binary to Decimal

$(1010.01)_2$

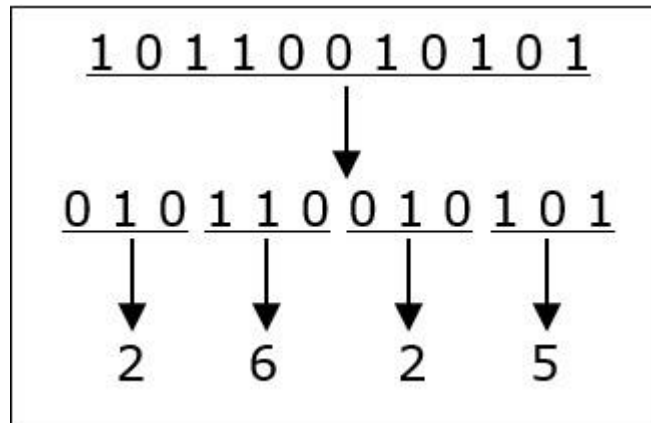
$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = 8 + 0 + 2 + 0 + 0 + 0.25 = 10.25$$

$11101_2 = 29_{10}$

### Binary to Octal and Vice Versa

To convert a binary number to octal number, these steps are followed –

- Starting from the least significant bit, make groups of three bits.
- If there are one or two bits less in making the groups, 0s can be added after the most significant bit
- Convert each group into its equivalent octal number



$$10110010101_2 = 2625_8$$

## Binary to Hexadecimal

Then, look up each group in a table:

Binary:	0000	0001	0010	0011	0100	0101	0110	0111
Hexadecimal:	0	1	2	3	4	5	6	7
Binary:	1000	1001	1010	1011	1100	1101	1110	1111
Hexadecimal:	8	9	A	B	C	D	E	F

Binary = 1110 0101  
Hexadecimal = E 5 = E5 hex

## 3. Decimal to Octal

$(10.25)_{10}$   
 $(10)_{10} = (12)_8$   
Fractional part:  
 $0.25 \times 8 = 2.00$

**Note:** Keep multiplying the fractional part with 8 until decimal part .00 is obtained.

$(.25)_{10} = (.2)_8$   
**Answer:**  $(10.25)_{10} = (12.2)_8$

4. Octal to Decimal

$(12.2)_8$   
 $1 \times 8^1 + 2 \times 8^0 + 2 \times 8^{-1} = 8+2+0.25 = 10.25$   
 $(12.2)_8 = (10.25)_{10}$

Octal to Binary

Octal:	0	1	2	3	4	5	6	7
Binary:	000	001	010	011	100	101	110	111

Octal =

3

4

5

Binary =

011

100

101

= 011100101 binary

Octal to Hexadecimal

Octal = 3 4 5  
Binary = 011 100 101 = 011100101 binary

Drop any leading zeros or pad with leading zeros to get groups of four binary digits (bits):  
Binary 011100101 = 1110 0101

Binary:	0000	0001	0010	0011	0100	0101	0110	0111
	0	1	2	3	4	5	6	7
Hexadecimal:	1000	1001	1010	1011	1100	1101	1110	1111
	8	9	A	B	C	D	E	F

Binary = 1110 0101  
Hexadecimal = E 5 = E5 hex

Hexadecimal to Binary

Hexadecimal:	0	1	2	3	4	5	6	7
Binary:	0000	0001	0010	0011	0100	0101	0110	0111
Hexadecimal:	8	9	A	B	C	D	E	F
Binary:	1000	1001	1010	1011	1100	1101	1110	1111

Hexadecimal =      A                      2                      D                      E

Binary =            1010                      0010                      1101                      1110

1010001011011110 binary

# Hexadecimal to Octal

Hexadecimal =     A                             2                             D                             E  
Binary =            1010                    0010                    1101            1110     = 1010001011011110 binary

Add leading zeros or remove leading zeros to group into sets of three binary digits.

Binary: 1010001011011110 = 001 010 001 011 011 110

Binary:	000	001	010	011	100	101	110	111
Octal:	0	1	2	3	4	5	6	7

Binary = 001       010       001       011       011       110

Octal = 1        2        1        3        3        6  
       = 121336 octal



## Hexadecimal to Decimal

Hexadecimal:	0	1	2	3	4	5	6	7
Decimal:	0	1	2	3	4	5	6	7
Hexadecimal:	8	9	A	B	C	D	E	F
Decimal:	8	9	10	11	12	13	14	15

A2DE hexadecimal:

$$= ((A) * 16^3) + (2 * 16^2) + ((D) * 16^1) + ((E) * 16^0)$$

$$= (10 * 16^3) + (2 * 16^2) + (13 * 16^1) + (14 * 16^0)$$

$$= (10 * 4096) + (2 * 256) + (13 * 16) + (14 * 1)$$

$$= 40960 + 512 + 208 + 14$$

$$= 41694 \text{ decima}$$

# Evaluation of Expressions

$$X = a / b - c + d * e - a * c$$

$$a = 4, b = c = 2, d = e = 3$$

Interpretation 1:

$$((4/2)-2)+(3*3)-(4*2)=0 + 8+9=1$$

Interpretation 2:


$$(4/(2-2+3))*(3-4)*2=(4/3)*(-1)*2=-2.66666\dots$$

How to generate the machine instructions corresponding to a given expression?

precedence rule + associative rule

Token	Operator	Precedence <sup>1</sup>	Associativity
( ) [ ] -> .	function call array element struct or union member	17	left-to-right
-- ++	increment, decrement <sup>2</sup>	16	left-to-right
-- ++ ! - - + & * sizeof	decrement, increment <sup>3</sup> logical not one's complement unary minus or plus address or indirection size (in bytes)	15	right-to-left
(type)	type cast	14	right-to-left
* / %	mutiplicative	13	Left-to-right

+ -	binary add or subtract	12	left-to-right
<< >>	shift	11	left-to-right
> >= < <=	relational	10	left-to-right
== !=	equality	9	left-to-right
&	bitwise and	8	left-to-right
^	bitwise exclusive or	7	left-to-right
	bitwise or	6	left-to-right
&&	logical and	5	left-to-right
ξξ	logical or	4	left-to-right

?:	conditional	3	right-to-left
= += -= /= *= %= <<= >>= &= ^= 	assignment	2	right-to-left
,	comma	1	left-to-right

# Order of Operation

- Parentheses  $() \{ \} [ ]$
- Exponents (right to left)
- Multiplication and division (left to right)
- Addition and subtraction (left to right)

Infix Expression	Prefix Expression	Postfix Expression
$A + B * C + D$	$++A * B C D$	$A B C * + D +$
$(A + B) * (C + D)$	$* + A B + C D$	$A B + C D + *$
$A * B + C * D$	$+ * A B * C D$	$A B * C D * +$
$A + B + C + D$	$+++A B C D$	$A B + C + D +$

# Statements and flow control

a) if statement (Block of stmnts are executed if condition is true)

## Sorting Two Numbers

b) nested if statement

c) if-else statement

d) if-else-if statement

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

```
cout << "Enter two integers: ";
```

```
int Value1;
```

```
int Value2;
```

```
cin >> Value1 >> Value2;
```

```
if (Value1 > Value2)
```

```
{
```

```
    int RememberValue1 = Value1;
```

```
    Value1 = Value2;
```

```
    Value2 = RememberValue1;
```

```
}
```

```
cout << "The input in sorted order: " << Value1 << " " << Value2  
<< endl;
```



```
#include <iostream.h>
#include <conio.h>
void main(void)
{
    char grade ;
    int score ;

    cout << "Enter the score: " ;
    cin >> score ;
    cout << endl ;

    if(score >= 70)
    {
        grade = 'A' ;
        cout << "Excellent. Grade: " << grade
    }
}
```

**Enter the score: 70**  
**Excellent. Grade: A**

- Syntax

if (*Expression*)

*Action*<sub>1</sub>

else

*Action*<sub>2</sub>

- If *Expression* is true then execute *Action*<sub>1</sub> otherwise execute *Action*<sub>2</sub>

if (v == 0) {

    cout << "v is 0";

}

else {

    cout << "v is not 0";

}

## Finding the Max

```
cout << "Enter two integers: ";  
int Value1;  
int Value2;  
cin >> Value1 >> Value2;  
int Max;  
if (Value1 < Value2) {  
    Max = Value2;  
}  
else {  
    Max = Value1;  
}  
cout << "Maximum of inputs is: " << Max << endl;
```

## An If-Else-If Statement

```
if ( nbr < 0 ){  
    cout << nbr << " is negative" << endl;  
}  
else if ( nbr > 0 ) {  
    cout << nbr << " is positive" << endl;  
}  
else {  
    cout << nbr << " is zero" << endl;  
}
```

```
if(score < 35)
{
    grade = 'F' ;
    cout << "Failed. Grade: " << grade ;
}

else

    if(score < 70)
    {
        grade = 'B' ;
        cout << "Average. Grade: " << grade ;
    }
else
{
    grade = 'A' ;
    cout << "Excellent. Grade: " << grade ;
}
```

**Enter the score: 34**

**Failed. Grade: F**

**Enter the score: 56**

**Average. Grade: B**

**Enter the score: 75**

**Excellent. Grade: A**

## Nested 'if' statement

```
if (expression 1)
{
    if (expression 2)
    {
        Processing statement 1;
        Processing statement 2;
    }
}
```

```
#include <iostream>
using namespace std;
int main()
{
    int num=90;
    if( num < 100 )
    {
        cout<<"number is less than 100"<<endl;
    }
    if(num > 50)
    {
        cout<<"number is greater than 50";
    }
    return 0;
}
```

## Output:

number is less than 100  
number is greater than 50

# Decision Making

## •Jump Statements:

1.break

2.continue

3.goto

If-else

Switch

if

if-else

if-else if

Nested if

```
if( condition )
{
    //true
}
```

```
if( condition )
{
    //true
}
else
{
    //false
}
```

```
if( condition 1 )
{
    //true
}
else if( condition 2 )
{
    //true
}
else
{
}
```

```
if( condition 1 )
{
    if(condition)
    {
    }
    else
    {
    }
}
else
{
    if(condition)
    {
    }
    else
    {
    }
}
```

```
switch( expression )
{
    case 1:
        break;
    case 2:
        break;
    case 3:
        break;
    default;
}
```



## Test expression is true

```
int test = 5;
```

```
if (test < 10)  
{  
    // codes  
}
```

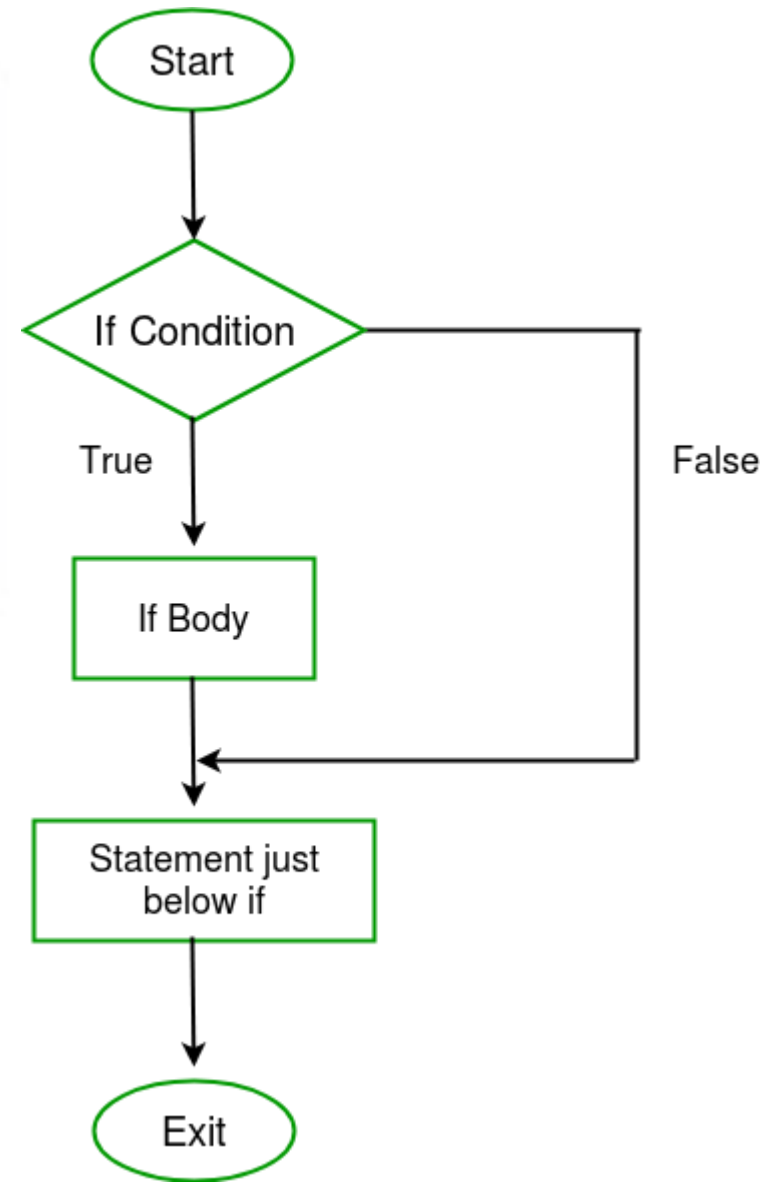
```
// codes after if
```

## Test expression is false

```
int test = 5;
```

```
if (test > 10)  
{  
    // codes  
}
```

```
// codes after if
```



- // Program to print positive number entered by the user
- // If user enters negative number, it is skipped

```
#include <iostream>
using namespace std;
int main()
{
    int number;
    cout << "Enter an integer: ";
    cin >> number;
    // checks if the number is positive
    if ( number > 0)
    {
        cout << "You entered a positive integer: " << number << endl;
    }
    cout << "This statement is always executed.";
    return 0;
}
```

Enter an integer: 5

You entered a positive number: 5


This statement is always executed.

# How if...else statement works

Test expression is true

```
int test = 5;

if (test < 10)
{
    // codes
}
else
{
    // codes
}
// codes after if...else
```



Test expression is false

```
int test = 5;

if (test > 10)
{
    // codes
}
else
{
    // codes
}
// codes after if...else
```

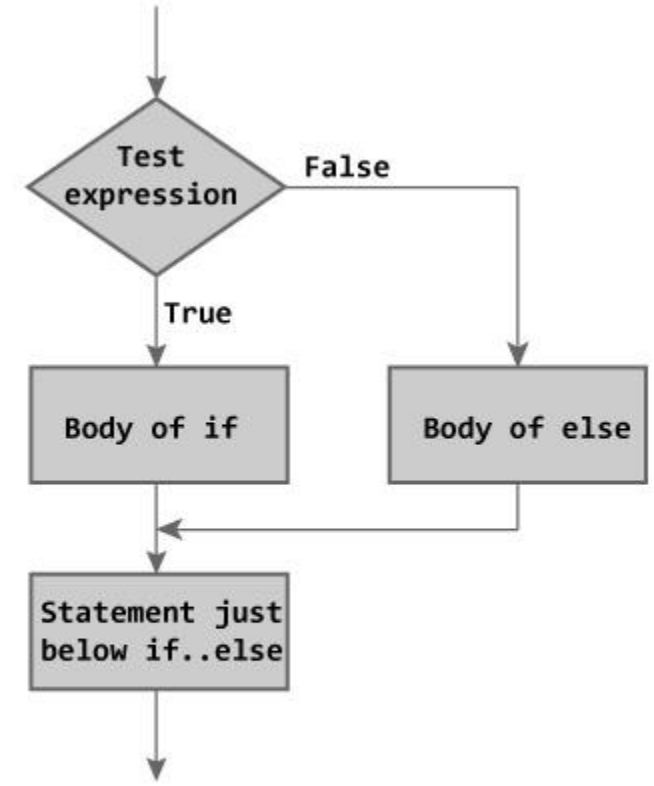



Figure: Flowchart of if...else Statement

- // Program to check whether an integer is positive or negative
- // This program considers 0 as positive number

```
#include <iostream>
using namespace std;
int main()
{
    int number;
    cout << "Enter an integer: ";
    cin >> number;
    if ( number >= 0)
    {
        cout << "You entered a positive integer: " << number << endl;
    }
    else
    {
        cout << "You entered a negative integer: " << number << endl;
    }
    cout << "This line is always printed.";
    return 0;
}
```

Enter an integer: -4

You entered a negative integer: -4.

This line is always printed.

## Syntax of Nested if...else

```
if (testExpression1)
{
    // statements to be executed if testExpression1 is true
}
else if(testExpression2)
{
    // statements to be executed if testExpression1 is false and testExpression2 is true
}
else if (testExpression 3)
{
    // statements to be executed if testExpression1 and testExpression2 is false and testExpression3 is true
}
.
.
else
{
    // statements to be executed if all test expressions are false
}
```

A program to determine whether a character is in lower-case or upper case

```
#include<iostream>
using namespace std;
int main ()
{
    char ch;
    cout<<"Enter an alphabet:";
    cin>>ch;
    if( (ch>='A') && (ch<='Z'))
        cout <<"The alphabet is in upper case";
    else
        if ( (ch>=' a') && (ch<=' z' ) )
            cout<<"The alphabet is in lower case";
        else
            cout<<"It is not an alphabet";
    return 0;
}
```

Your local library needs your help! Given the expected and actual return dates for a library book, create a program that calculates the fine (if any). The fee structure is as follows:

- 1.If the book is returned on or before the expected return date, no fine will be charged (i.e.: **fine=0**).
- 2.If the book is returned after the expected return day but still within the same calendar month and year as the expected return date, . **fine =15\* Number of days late**
- 3.If the book is returned after the expected return month but still within the same calendar year as the expected return date, the .. **fine =500\* Number of months late**
- 4.If the book is returned after the calendar year in which it was expected, there is a fixed fine of **1000Rs**.

Actual:  $D_a=9$ ,  $M_a=9$ ,  $Y_a=2019$

Because  $Y_e=Y_a$ , we know it is less than a year late.

Expected:  $D_e=6$ ,  $M_e=6$ ,  $Y_e=2019$

Because  $M_e=M_a$ , we know it's less than a month late.

A: 9- 6 -2019

E: 6 -6 -2019

15 X (9-6)=45

```
int main()
```

```
{
```

```
    int d1; int m1; int y1;
```

```
    cin>>d1>>m1>>y1;
```

```
    int d2; int m2; int y2;
```

```
    cin>>d2>>m2>>y2;
```

```
    True { if(y2==y1)
```

```
    {
```

```
        True { if(m2==m1)
```

```
        {
```

```
            True { if(d1<d2)
```

```
            {
```

```
                cout<<"0";
```

```
            else
```

```
                cout<<15*(d1-d2);
```

```
            }
```

```
        else
```

```
        {
```

```
            True { if(m1<m2)
```

```
            {
```

```
                cout<<"0";
```

```
            else
```

```
                cout<<500*(m1-m2);
```

```
            }
```

```
        }
```

False

```
    else
```

```
    {
```

True

```
        { if(y1<y2)
```

```
            cout<<"0";
```

```
        else
```

```
            cout<<10000*(y1-y2);
```

```
        }
```

```
    return 0;
```

```
    }
```

False



# A Switch Statement

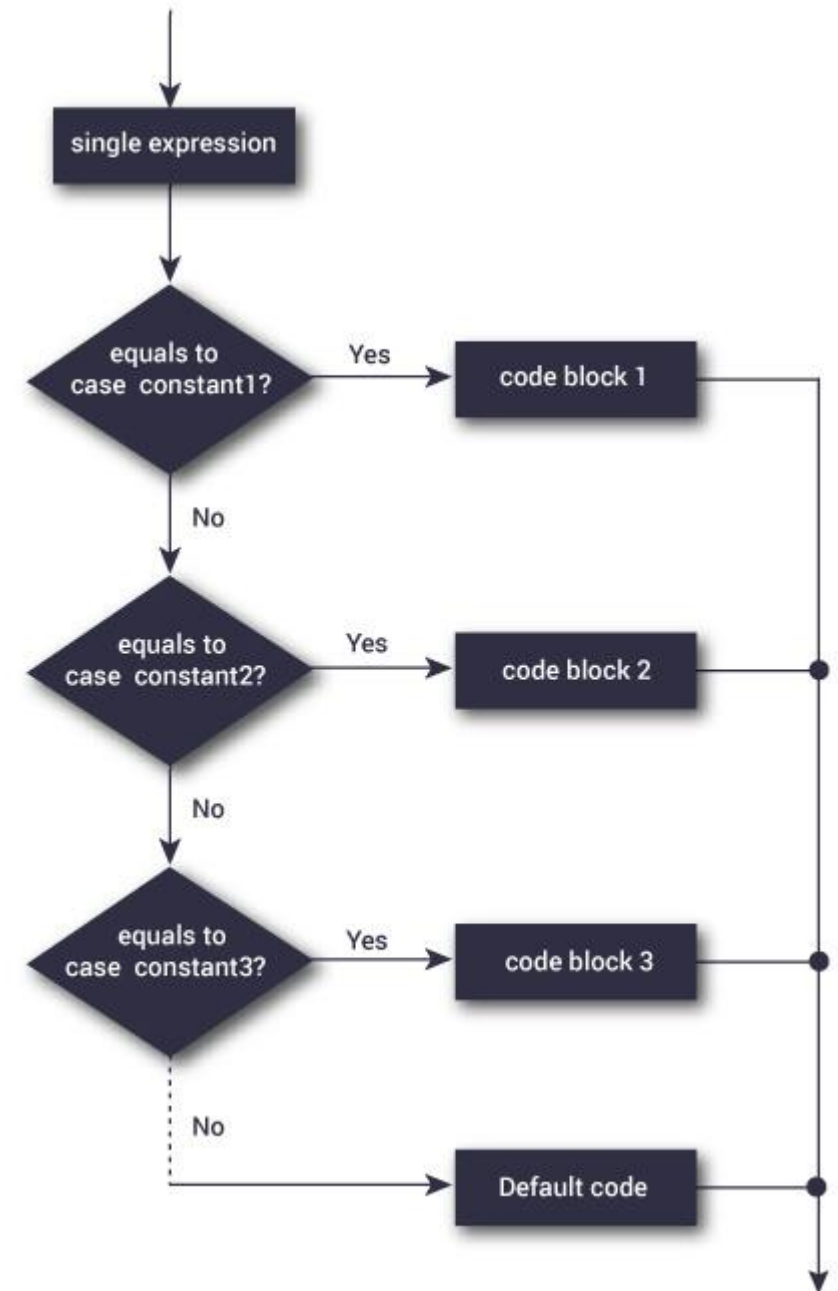
Switch case statement is used when we have multiple conditions and we need to perform different action based on the condition. When we have multiple conditions and we need to execute a block of statements when a particular condition is satisfied. In such case either we can use lengthy if..else-if statement or switch case. The problem with lengthy if..else-if is that it becomes complex when we have several conditions. The switch case is a clean and efficient method of handling such scenarios.

```
switch(expression)
{
case constant-expression :
    statement(s);
break; /* optional */
case constant-expression :
    statement(s);
break; /* optional */
/* you can have any number of case statements */
default : /* Optional */
    statement(s);
}
```

1. The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type.
2. You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
3. The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
4. When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
5. When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
6. A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No **break** is needed in the default case.

# A Switch Statement

```
switch (ch) {  
    case 'a': case 'A':  
    case 'e': case 'E':  
    case 'i': case 'I':  
    case 'o': case 'O':  
    case 'u': case 'U':  
        cout << ch << " is a vowel" << endl;  
        break;  
    default:  
        cout << ch << " is not a vowel" << endl;  
}
```



```
#include <iostream>
using namespace std;
int main(){
    int num=1;
    switch(num) {
        case 1:
            cout<<"Case1: Value is: "<<num<<endl;
        case 2:
            cout<<"Case2: Value is: "<<num<<endl;
        case 3:
            cout<<"Case3: Value is: "<<num<<endl;
        default:
            cout<<"Default: Value is: "<<num<<endl;
    }
    return 0;
}
```

Output: Case1: Value is: 1  
Case2: Value is:1  
Case2: Value is:1  
Default : Value is:1

```
#include <iostream>
using namespace std;
int main(){
    int i=5;
    switch(i) {
        case 1: cout<<"Case1 "<<endl;
        case 2: cout<<"Case2 "<<endl;
        case 3: cout<<"Case3 "<<endl;
        case 4: cout<<"Case4 "<<endl;
        default: cout<<"Default "<<endl;
    }
    return 0;
}
```

Output: Default

```
#include <iostream>
using namespace std;
int main(){
    int i=2;
    switch(i) {
        case 1:
            cout<<"Case1 "<<endl;
            break;
        case 2:
            cout<<"Case2 "<<endl;
            break;
        case 3:
            cout<<"Case3 "<<endl;
            break;
        case 4:
            cout<<"Case4 "<<endl;
            break;
        default:
            cout<<"Default "<<endl;
    }
    return 0;
}
```

Output: Case2

```
#include <iostream>
using namespace std;
int main(){
    char ch='b';
    switch(ch) {
        case 'd': cout<<"Case1 ";
        break;
        case 'b': cout<<"Case2 ";
        break;
        case 'x': cout<<"Case3 ";
        break;
        case 'y': cout<<"Case4 ";
        break;
        default: cout<<"Default ";
    }
    return 0;
}
```

Output: Case2

```
cout << "Enter simple expression: ";  
int Left;  
int Right;  
char Operator;  
cin >> Left >> Operator >> Right;  
cout << Left << " " << Operator << " " << Right << " = ";  
switch (Operator)  
{  
    case '+' : cout << Left + Right << endl; break;  
    case '-' : cout << Left - Right << endl; break;  
    case '*' : cout << Left * Right << endl; break;  
    case '/' : cout << Left / Right << endl; break;  
    default: cout << "Illegal operation" << endl;  
}
```

# Recall Purpose of Loops/Repetition

- To apply the same steps again and again to a block of statements.
- Recall a block of statement is one or more statement, block usually defined by braces { ... } with syntactically correct statements inside.



# Most Common Uses of Loops

- For counting
- For accumulating, i.e. summing
- For searching
- For sorting
- For displaying tables
- For data entry – from files and users
- For menu processing
- For list processing

# Types of loops

➤ while

➤ for

➤ do..while

# C++ Loop Structures

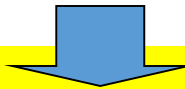
- **Pre-test (the test is made before entering the loop)**
  - **while loops**
    - general purpose
  - **for loops**
    - When you know how many times (**fixed condition**)
    - When you process arrays (more in later lectures)
- **Post-test (the test is done at the end of the loop)**
  - **do ... while loops**
    - When you do not know how many times, but you know you need at least one pass.

# The for Statement Syntax

start condition

while condition

change expression



**Example:**

```
for (count=1;
```

```
count < 7;
```

```
count++)
```

```
{
```

```
    cout << count << endl;
```

```
}
```

```
//next C++ statements;
```

# Example of Repetition

```
int  n;

for ( int i = 1 ; i <= n ; i++ )
{
    cout << i << “ Potato” << endl;
}
```

num

?

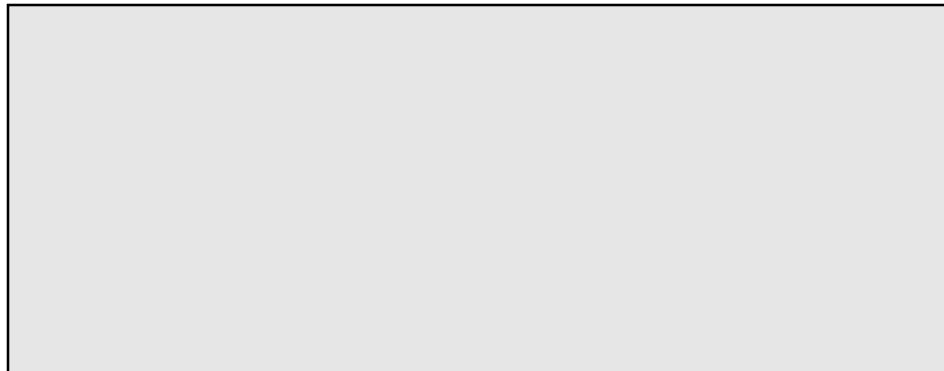
## Example of Repetition

```
int n;
```

```
for ( int i = 1 ; i <= n ; i++ )
```

```
    cout << i << " Potato" << endl;
```

**OUTPUT**



num

1

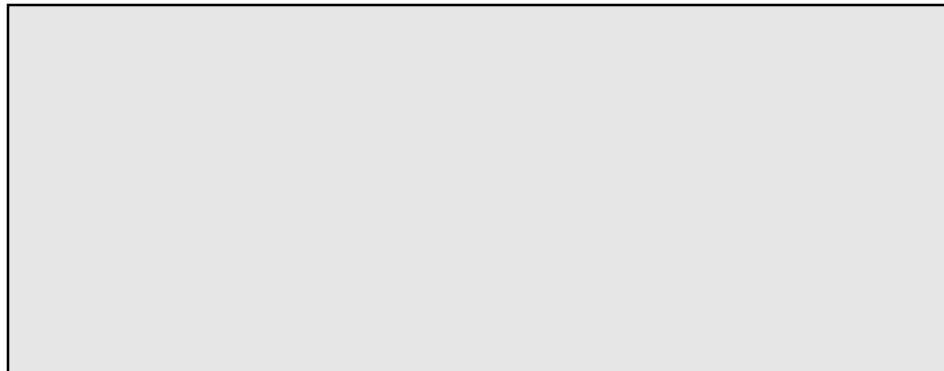
## Example of Repetition

```
int num;
```

```
for ( num = 1 ; num <= 3 ; num++ )
```

```
    cout << num << "Potato" << endl;
```

**OUTPUT**



num

1

## Example of Repetition

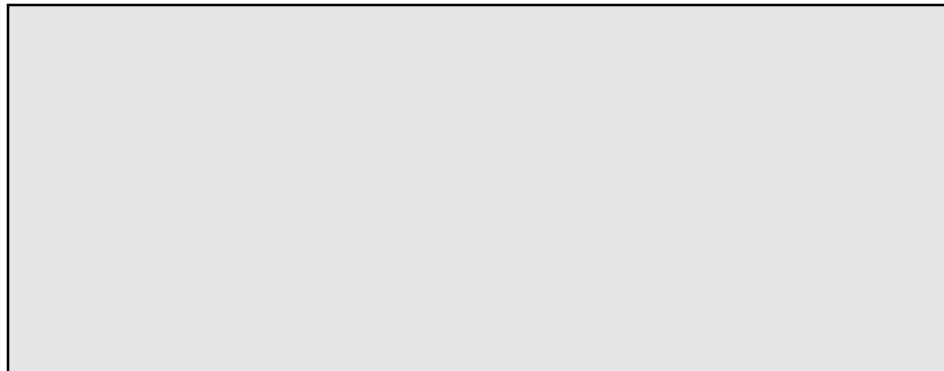
```
int num;
```

true

```
for ( num = 1 ; num <= 3 ; num++ )
```

```
    cout << num << "Potato" << endl;
```

### OUTPUT





num

1

## Example of Repetition

```
int num;
```

```
for ( num = 1 ; num <= 3 ; num++ )
```

```
cout << num << "Potato" << endl;
```

## OUTPUT

1Potato

num

2

## Example of Repetition

```
int num;
```

```
for ( num = 1 ; num <= 3 ; num++ )
```

```
    cout << num << "Potato" << endl;
```

## OUTPUT

1Potato

num

2

## Example of Repetition

```
int num;
```

true

```
for ( num = 1 ; num <= 3 ; num++ )
```

```
    cout << num << "Potato" << endl;
```

## OUTPUT

1Potato

num

2

## Example of Repetition

```
int num;
```

```
for ( num = 1 ; num <= 3 ; num++ )
```

```
cout << num << "Potato" << endl;
```

## OUTPUT

1Potato

2Potato

num

3

## Example of Repetition

```
int num;
```

```
for ( num = 1 ; num <= 3 ; num++ )
```

```
    cout << num << "Potato" << endl;
```

## OUTPUT

1Potato

2Potato

num

3

## Example of Repetition

```
int num;
```

true

```
for ( num = 1 ; num <= 3 ; num++ )
```

```
    cout << num << "Potato" << endl;
```

## OUTPUT

1Potato

2Potato

num

3

## Example of Repetition

```
int num;
```

```
for ( num = 1 ; num <= 3 ; num++ )
```

```
cout << num << "Potato" << endl;
```

## OUTPUT

1Potato

2Potato

3Potato

num

4

## Example of Repetition

```
int num;
```

```
for ( num = 1 ; num <= 3 ; num++ )
```

```
    cout << num << "Potato" << endl;
```

## OUTPUT

1Potato

2Potato

3Potato



num

4

## Example of Repetition

```
int num;
```

false

```
for ( num = 1 ; num <= 3 ; num++ )
```

```
    cout << num << "Potato" << endl;
```

## OUTPUT

1Potato

2Potato

3Potato

num

4

## Example of Repetition

```
int num;
```

false

```
for ( num = 1 ; num <= 3 ; num++ )
```

```
    cout << num << "Potato" << endl;
```

**When the loop control condition is evaluated and has value false, the loop is said to be “satisfied” and control passes to the statement following the for statement.**

## A Simple Example

Create a table with a for loop **NUMBER SQUARE CUBE**

```
int num;  
cout << "NUMBER\tSQUARE\tCUBE\n";  
cout << "-----\t-----\t----\n";  
  
for (num = 1; num < 11; num++) {  
    cout << num << "\t";  
    cout << num * num << "\t";  
    cout << num * num * num << "\n";  
}  
//see useofFunction2.cpp
```

NUMBER	SQUARE	CUBE
-----	-----	-----
1	1	1
2	4	8
.	.	.
.	.	.
10	100	1000

// C++ Program to compute factorial of a number

// Factorial of  $n = 1*2*3...*n$

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int number, i = 1, factorial = 1;
```

```
    cout << "Enter a positive integer: ";
```

```
    cin >> number;
```

```
    while ( i <= number)
```

```
    {
```

```
        factorial *= i; //factorial = factorial * i;
```

```
        ++i;
```

```
    }
```

```
    cout<<"Factorial of "<< number <<" = "<<
```

```
    factorial;
```

```
    return 0;
```

```
}
```

# Fibonacci Series up to n number of terms

The Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21

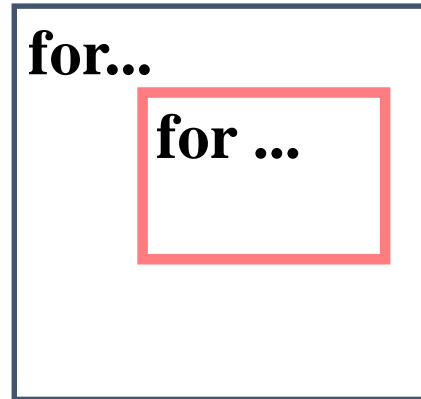
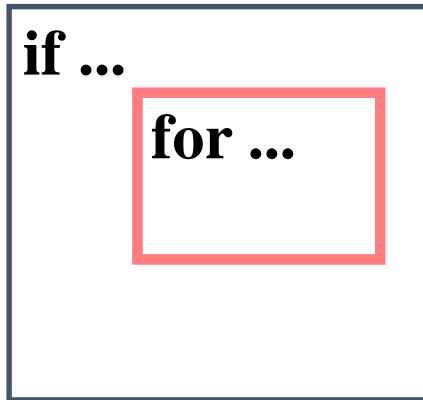
Enter the number of terms: 10

Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,

```
int main()
{
    int n, t1 = 0, t2 = 1, nextTerm = 0;
    cout << "Enter the number of terms: ";
    cin >> n;
    cout << "Fibonacci Series: ";
    for (int i = 1; i <= n; ++i)
    {
        // Prints the first two terms.
        if(i == 1)
        {
            cout << " " << t1; continue;
        }
        if(i == 2)
        {
            cout << t2 << " "; continue;
        }
        nextTerm = t1 + t2;
        t1 = t2;
        t2 = nextTerm;
        cout << nextTerm << " ";
    }
    return 0;
}
```

# Nested Loops

Recall when a control structure is contained within another control structure, the inner one is said to be *nested*.



## Nested Loops - Ex.1 for within for

### Example

```
int row,col;
for (row = 0; row < 5; row++)
{
    cout << "\n" << row; //throws a new line
    for (col = 1; col <= 3; col++)
    {
        cout << "\t" << col;
    }
    cout << "\t**"; //use of tab
}
```

What is the output from this loop?

```
int count;  
for (count = 0; count < 10; count++) ;  
{  
    cout << "*" ;  
}
```



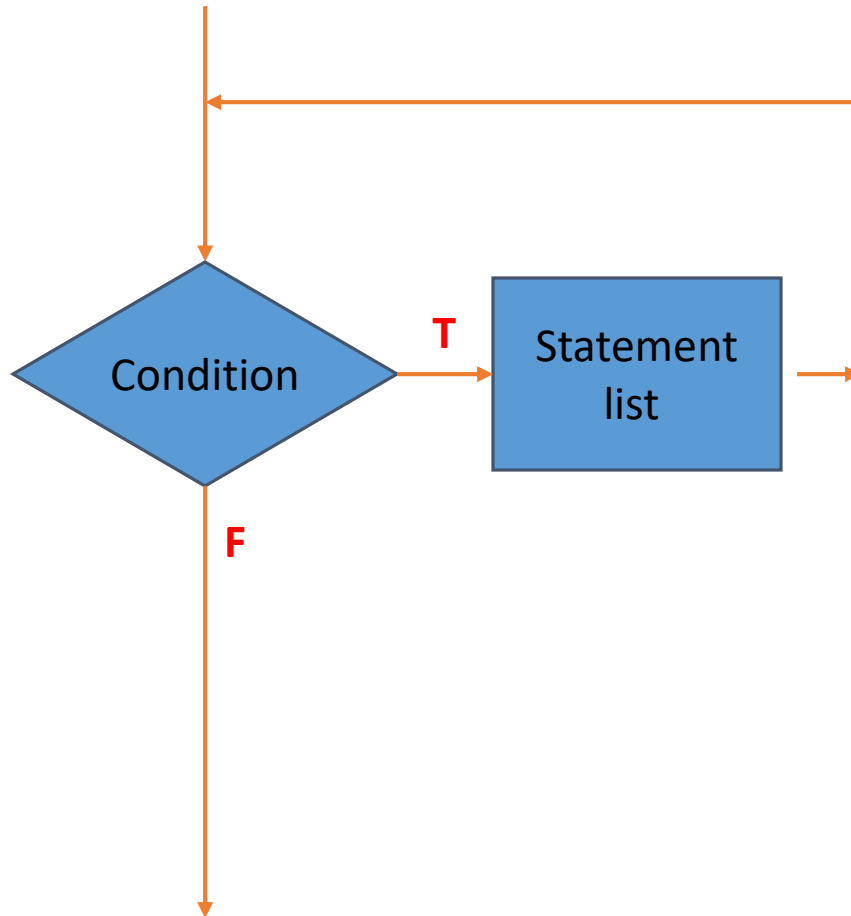
## Common errors in constructing for Statements

- `for (j = 0, j < n, j = j + 3)`
- `// commas used when semicolons needed`
- `for (j = 0; j < n)`
- `// three parts needed`
- `for (j = 0; j >= 0; j++)`
- `?????what is wrong here ?????`
- `for (j=0, j=10; j++);`

# The for Statement

- Used as a counting loop
- Used when we can work out in advance the number of iterations, i.e. the number of times that we want to loop around.
- Semicolons separate the items in the for loop block
  - There is no semi colon at the end of the for loop definition at the beginning of the statement

# while



```
while (Condition)
{
    Statement list
}
```

## Example 1: **while**

```
string ans = "n";
```

**(ans != "Y" || ans != "y")**

```
while (ans != "Y" && ans != "y")
```

```
{  
    cout << "Would you marry me?";  
    cin >> ans;  
}
```

```
cout << "Great!!";
```

**Can I put ; here?**

**No!!**

**Should I put ; here?**

**Up to you!!**

## Example 2: **while**

Will there be any problem?

What if one inputs -1?

```
int no_times;
```

```
cout << "How many times do you want to say?";  
cin  >> no_times;
```

```
while (no_times != 0)  
{  
    cout << "Hello!" << endl;  
    no_times--;  
}
```

```
while (no_times > 0)
```

```
cout << "End!!" << endl;
```

### Example 3: **while**

```
int a,b,sum;
```

```
cout << "This program will return the ";  
cout << "summation of integers from a to b.\n\n";  
cout << "Input two integers a and b:";  
cin  >> a >> b;
```

```
sum = 0;
```

Don't forget to set  
**sum = 0;**

```
while (a <= b)
```

```
{
```

```
    sum += a;
```

```
    a++;
```

```
}
```

sum = sum + a;

```
cout << "The sum is " << sum << endl;
```

## Program to Generate Fibonacci Sequence Up to a Certain Number

```
int main()
{
    int t1 = 0, t2 = 1, nextTerm = 0, n;
    cout << "Enter a positive number: ";
    cin >> n;
    // displays the first two terms which is always 0 and 1
    cout << "Fibonacci Series: " << t1 << ", " << t2 << ", ";
    nextTerm = t1 + t2;
    while(nextTerm <= n)
    {
        cout << nextTerm << ", ";
        t1 = t2;
        t2 = nextTerm;
        nextTerm = t1 + t2;
    }
    return 0;
}
```

Enter a positive integer: 100  
Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,

# Do-While Statement

**Is a looping control structure in which the loop condition is tested after each iteration of the loop.**

## **SYNTAX**

**do**

**{**

**Statement**

**} while (*Expression*) ; //note semi colon**

**Loop body statement can be a single statement or a block.**



# Function Using Do-While

```
void GetYesOrNo ( char response )
//see UseofFunction1.cpp
// Inputs a character from the user

// Postcondition:      response has been input
// && response == 'y' or 'n'
{
    do
    {
        cin >> response ;    // skips leading whitespace

        if ( ( response != 'y' ) && ( response != 'n' ) )
            cout << "Please type y or n : " ;

    } while ( ( response != 'y' ) && ( response != 'n' ) ) ;
}
```

## Program code to Display Fibonacci Series in C++:

```
#include<iostream.h>
void main()
{
    int n,f, f1=-1,f2=1;

    cout<<" Enter The Number Of Terms:";
    cin>>n;

    cout<<" The Fibonacci Series is:";

    do
    {
        f=f1+f2;
        f1=f2;
        f2=f;
        cout<<" \n"<<f;
        n--;
    }while(n>0);
}
```

# Do-While Loop vs. While Loop

- POST-TEST loop (exit-condition)
- The looping condition is tested after executing the loop body.
- Loop body is always executed at least once.

- PRE-TEST loop (entry-condition)
- The looping condition is tested before executing the loop body.
- Loop body may not be executed at all.

# Break Statement Revisited

- break statement can be used with Switch or any of the 3 looping structures
- it causes an **immediate exit** from the Switch, while, do-while, or for statement in which it appears
- if the break is inside nested structures, control exits only the **innermost structure** containing it

## The break Statement

```
int j = 40;
while (j < 80){
    j += 10;
    if (j == 70)
        break;
    cout << "j is " << j << "\n";
}
cout << "We are out of the loop as j=70.\n";
```

j is 50

j is 60

We are out of the loop as j=70.

# Continue Statement

- is valid only within loops
- terminates the current loop iteration, but not the entire loop
- in a for or while, continue causes the rest of the body statement to be skipped--in a for statement, the update is done

## The continue Statement

```
int j = 40;
while (j < 80){
    j += 10;
    if (j == 70)
        continue;
    cout << "j is " << j << "\n";
}
cout << "We are out of the loop" << endl;
```

j is 50


j is 60

j is 80

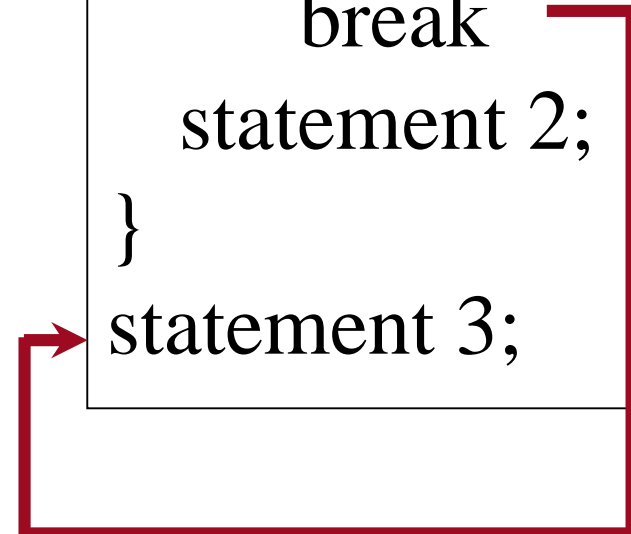
We are out of the loop.

# break *and* continue

```
while ( - - - )  
{  
    statement 1;  
    if( - - - )  
        continue  
    statement 2;  
}  
statement 3;
```



```
while ( - - - )  
{  
    statement 1;  
    if( - - - )  
        break  
    statement 2;  
}  
statement 3;
```







Boolean expressions, simple if. (program to check whether the given number is even or odd)

Simple if-else, program to check whether the given year is leap year or not

Nested if-else. (program for displaying three inputted numbers in ascending order)

switch case and break statement. (program to design a simple calculator)

while loop. Program to compute i) the sum of first n numbers.

ii) sum of the following series  $1+1.2+1.2.3+....+1.2.3...n$

Program to compute GCF and LCM of two numbers using while loop

do-while loop.(program to compute factorial of a given number)

for loop. Fibonacci series, program to find prime numbers in a given range.