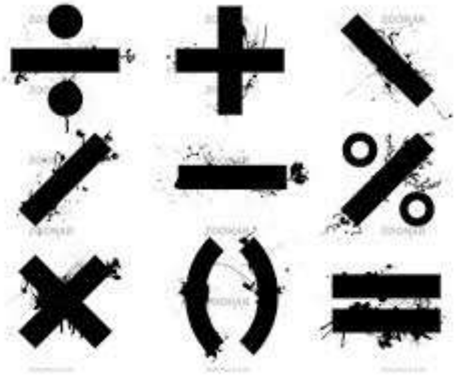
CU
CHANDIGARH UNIVERSITY

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

Operators & Expressions

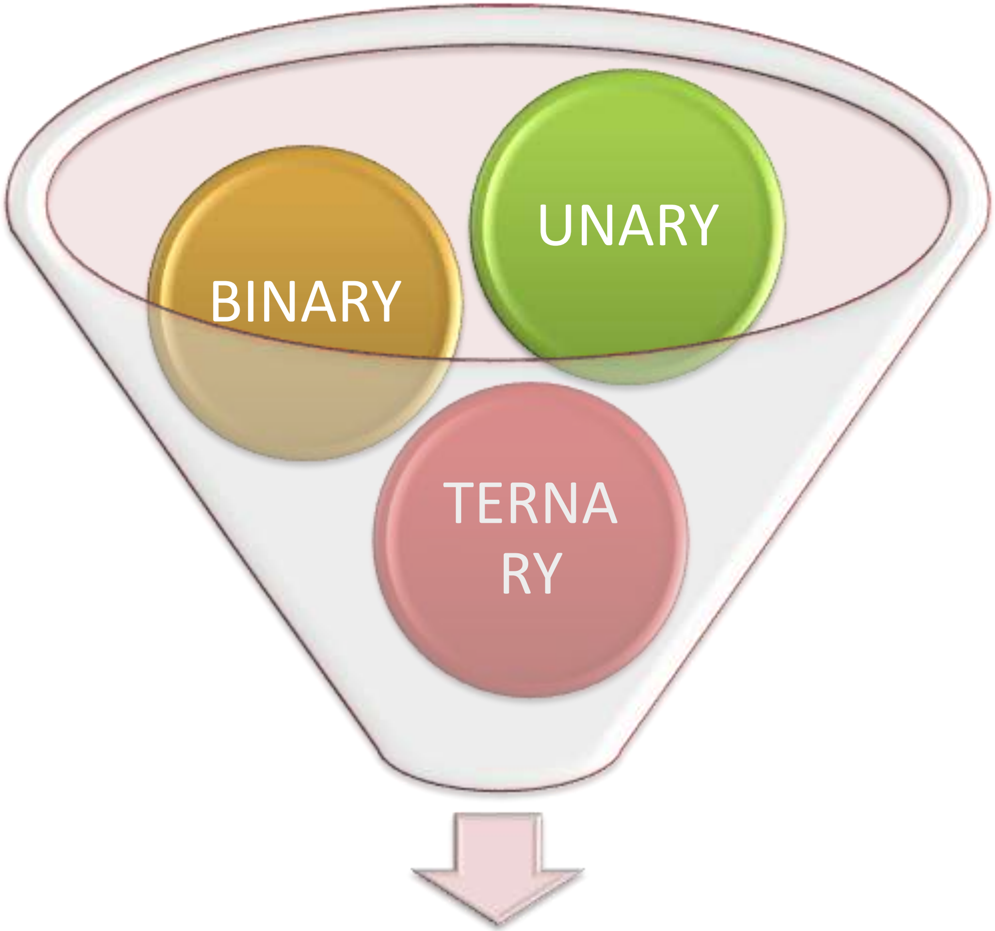
Presented by :
Er Jasleen Kaur
Assistant Professor
Applied Science(CSE)
Chandigarh University
Gharuan (Mohali).

Operators



- C supports rich set of operators.
- An operator is a symbol that tells the compiler to perform certain mathematical or logical manipulations.
- Operators are used in programs to manipulate data and variables.

Types of Operators



BINARY

UNARY

TERNA
RY

Operators

Unary Operators

- A unary operator is one which operates on one value or operand. The minus sign (-) plays a dual role, it is used for subtraction as a binary operator and for negation as a unary operator. This operator has a precedence higher than the rest of the arithmetic operators.
- result = -x * y;
- in the above expression, if x has a value 20 and y has a value 2, then result will contain a negative value of 40 which is -40.

Binary and Ternary Operators

- Binary operators?

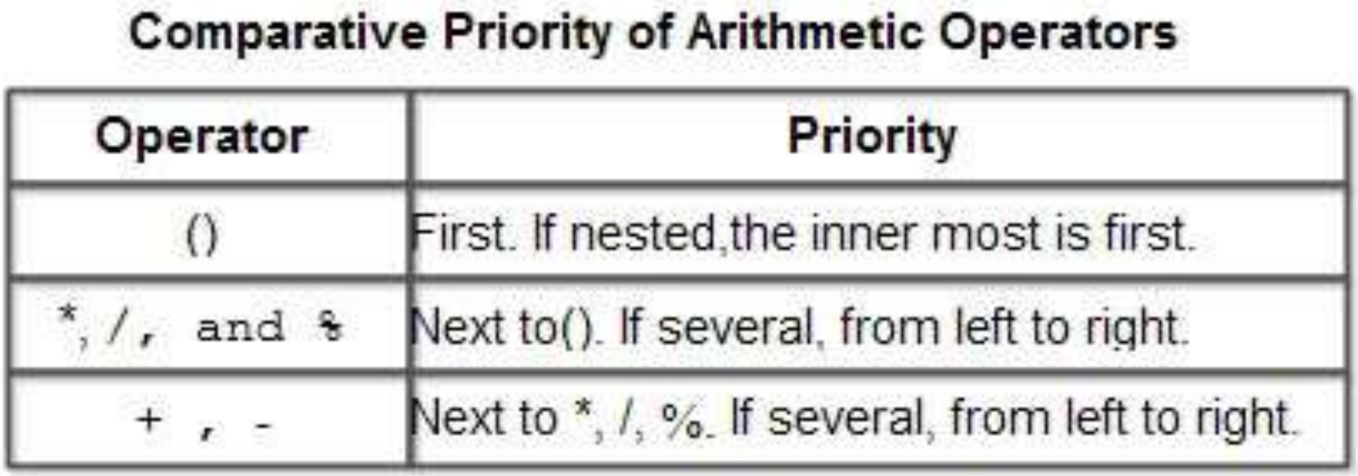
- Ternary operators?

Types of ‘C’ operators

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. **Increment and Decrement operators**
6. **Conditional operators**
7. Bitwise operators
8. Special operators

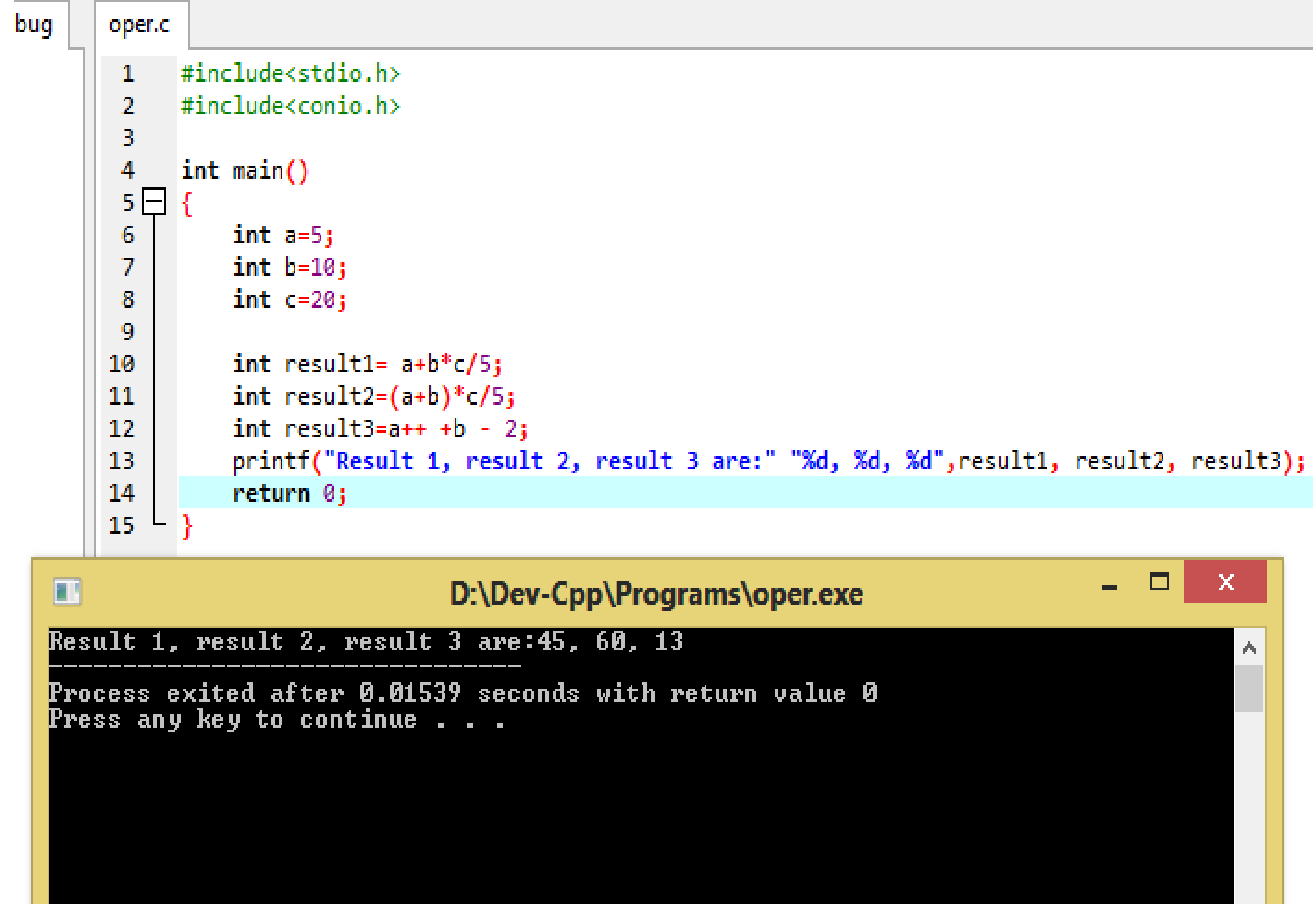
1. Arithmetic operator

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo division



Comparative Priority of Arithmetic Operators

Operator	Priority
()	First. If nested,the inner most is first.
*, /, and %	Next to(). If several, from left to right.
+ , -	Next to *, /, %. If several, from left to right.



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

int main()
{
    int a=5;
    int b=10;
    int c=20;

    int result1= a+b*c/5;
    int result2=(a+b)*c/5;
    int result3=a++ +b - 2;
    printf("Result 1, result 2, result 3 are:" "%d, %d, %d",result1, result2, result3);
    return 0;
}
```

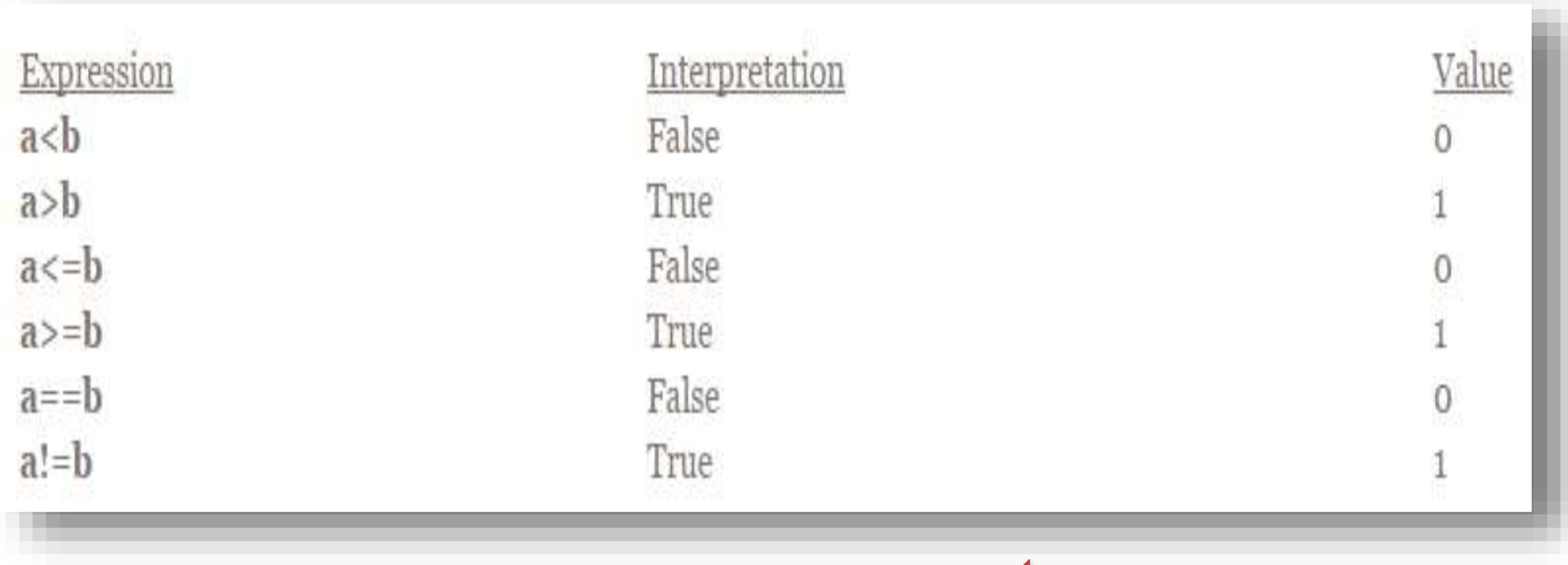
```
Result 1, result 2, result 3 are:45, 60, 13
--------------------------------
Process exited after 0.01539 seconds with return value 0
Press any key to continue . . .
```

2. Relational operator

C supports six Relational Operators

<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Is equal to
!=	Is not equal to

- Suppose that a and b are integer variables whose values are **100** and **4**, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values.



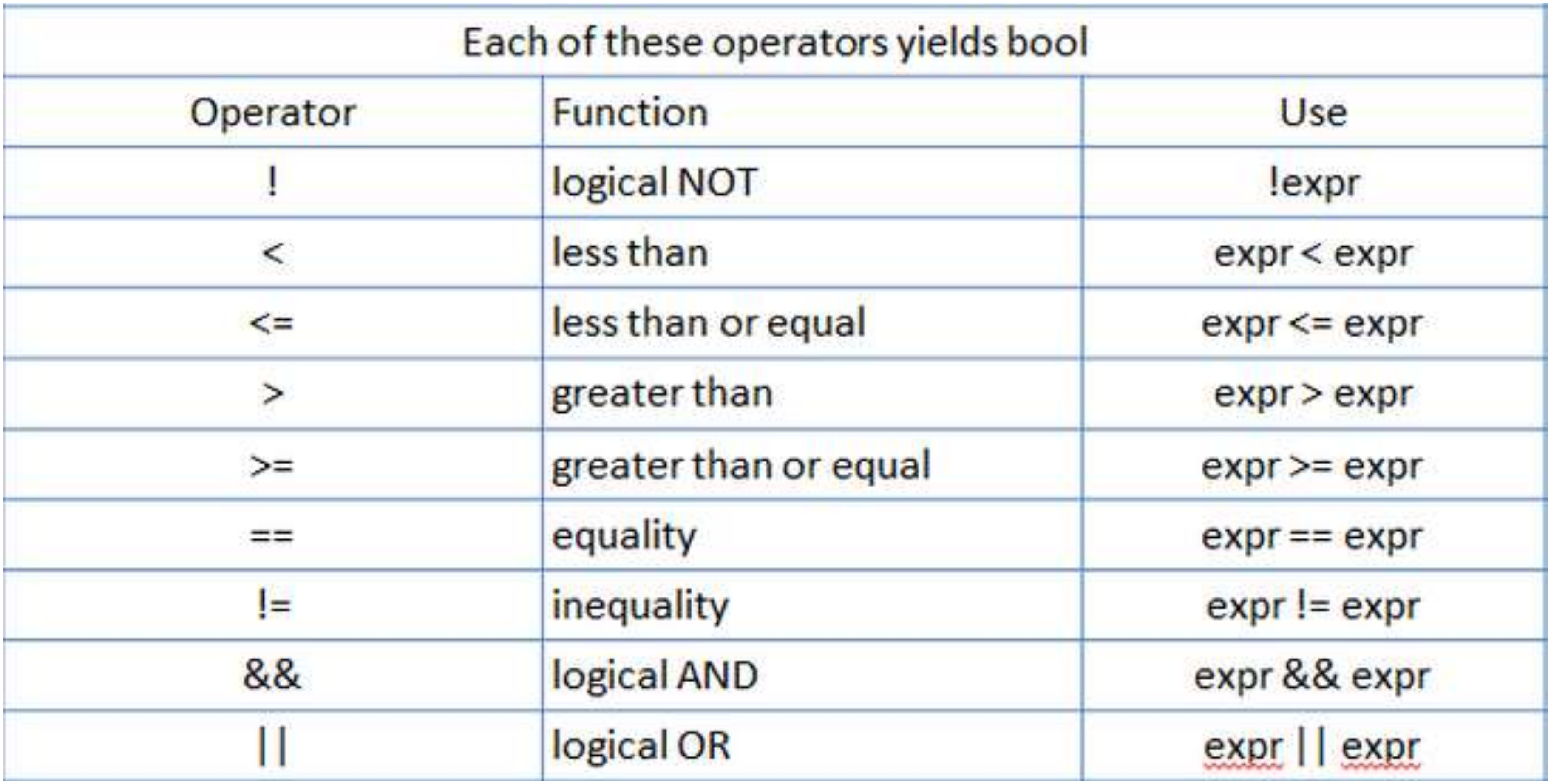
Expression	Interpretation	Value
a<b	False	0
a>b	True	1
a<=b	False	0
a>=b	True	1
a==b	False	0
a!=b	True	1

a=100, b=4

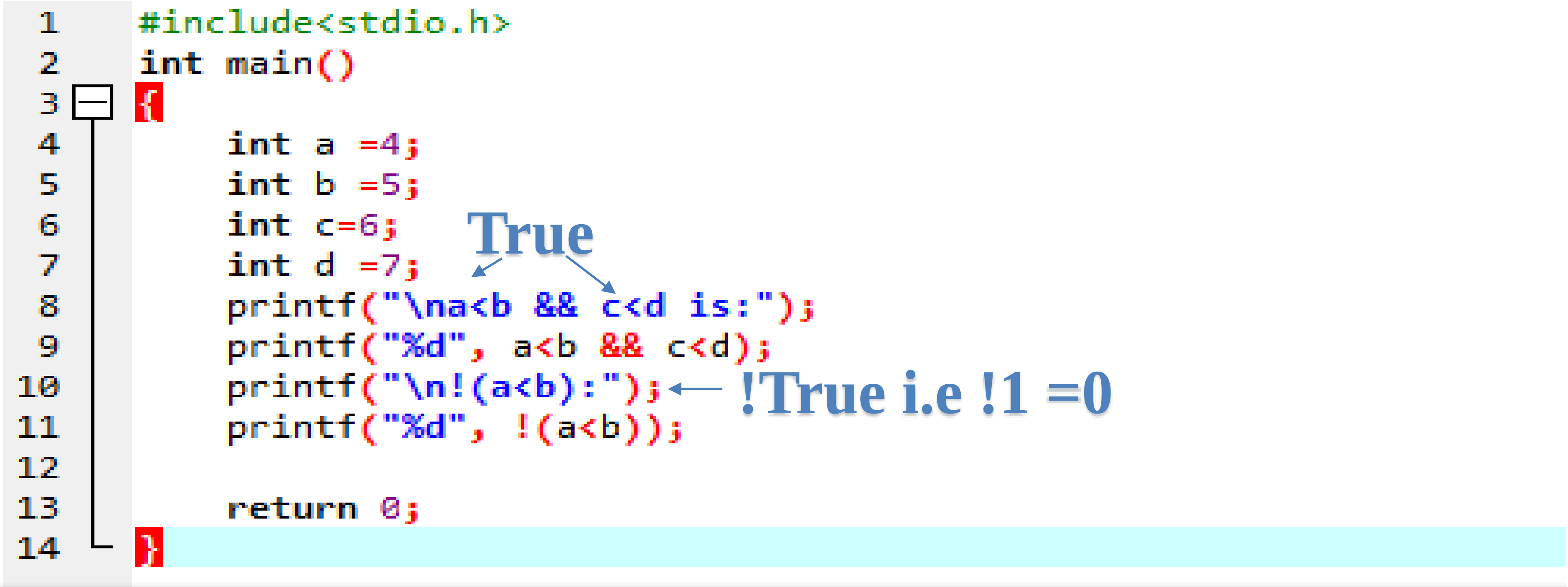
3.Logical operators

- Logical Operators
  - &&, || and ! are the three logical operators.
  - expr1 && expr2 has a value 1 if expr1 and expr2 both are nonzero i.e. if both have values 1(true)
  - expr1 || expr2 has a value 1 if either expr1 or expr2 or both are nonzero i.e 1(true).
  - !expr1 has a value 1 if expr1 is zero else 0.
  - Example
  - if (marks >= 40 && attendance >= 75) grade = ‘P’
  - If (marks < 40 || attendance < 75) grade = ‘N’

Relational And Logical Operators



Each of these operators yields bool		
Operator	Function	Use
!	logical NOT	!expr
<	less than	expr < expr
<=	less than or equal	expr <= expr
>	greater than	expr > expr
>=	greater than or equal	expr >= expr
==	equality	expr == expr
!=	inequality	expr != expr
&&	logical AND	expr && expr
||	logical OR	expr || expr

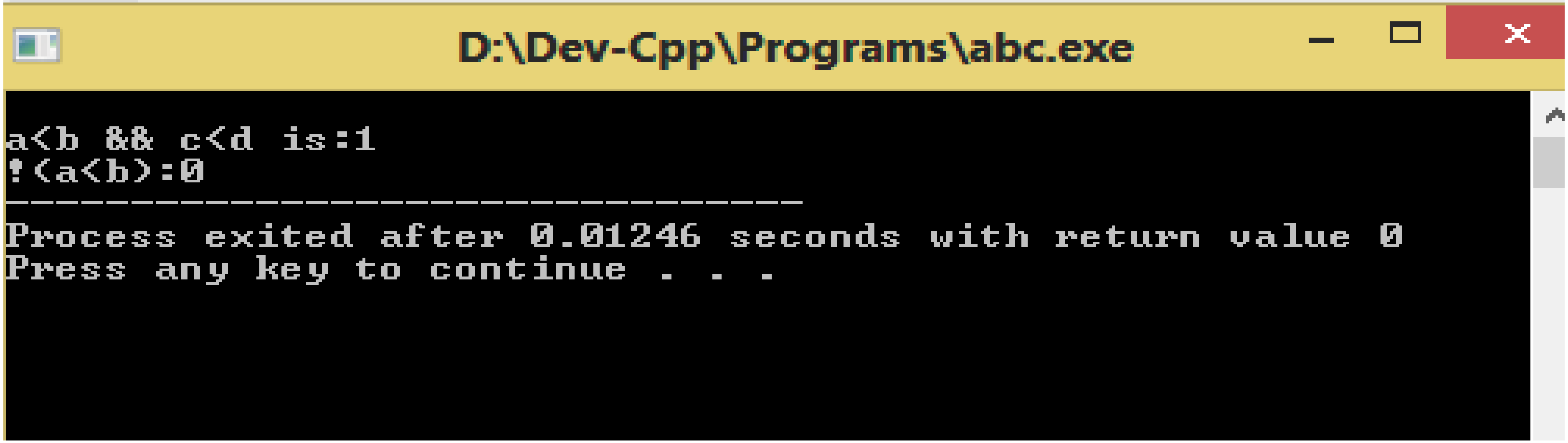


```
#include<stdio.h>
int main()
{
    int a =4;
    int b =5;
    int c=6;
    int d =7;
    printf("\na<b && c<d is:");
    printf("%d", a<b && c<d);
    printf("\n!(a<b):");
    printf("%d", !(a<b));

    return 0;
}
```

True

!True i.e !1 =0



```

a<b && c<d is:1
!(a<b):0
--------------------------------
Process exited after 0.01246 seconds with return value 0
Press any key to continue . . .
```

4. Assignment operators

- Assignment operators are used to assign the result of an expression to a variable.
- C has a set of ‘shorthand’ assignment operator :

variable name =expression;

Exam - a + = 3;

a = a + 3;

Both are same.

Left side must be an object that can receive a value

Shorthand Assignment operators

Simple assignment operator	Shorthand operator
a = a+1	a + =1
a = a-1	a - =1
a = a* (m+n)	a * = m+n
a = a / (m+n)	a / = m+n
a = a %b	a %=b

5. Increment and decrement operators.

- Increment Operator ++

  a=10;

  a++ =10 (post increment but in memory its value is 11)

when you will again call value of a, then a=11

- Decrement Operator --

  b=5;

  b-- =4 in memory but output will be 5; when you will call b again then value will be 4.

- Similarly increment and decrement operator is used in subscripted variables as:

  a[i++]=5;

  is equivalent to

  a[i]=5;

  i=i+1;

6. Conditional operator

- The conditional expression can be used as shorthand for some if-else statements. It is a ternary operator.
- This operator consist of two symbols: the question mark (?) and the colon (:).

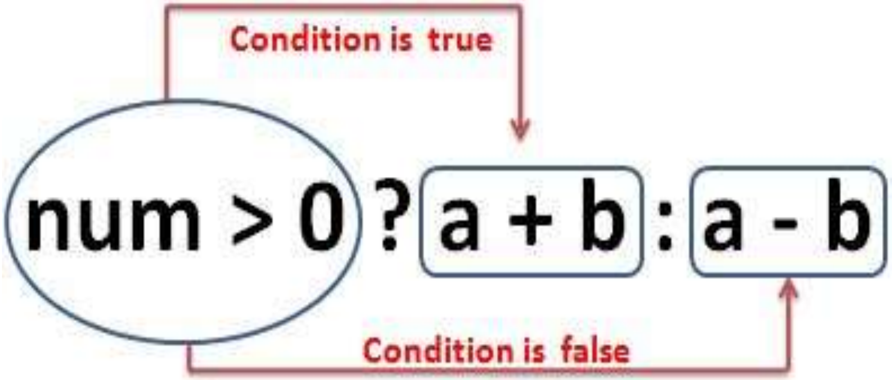
for example:

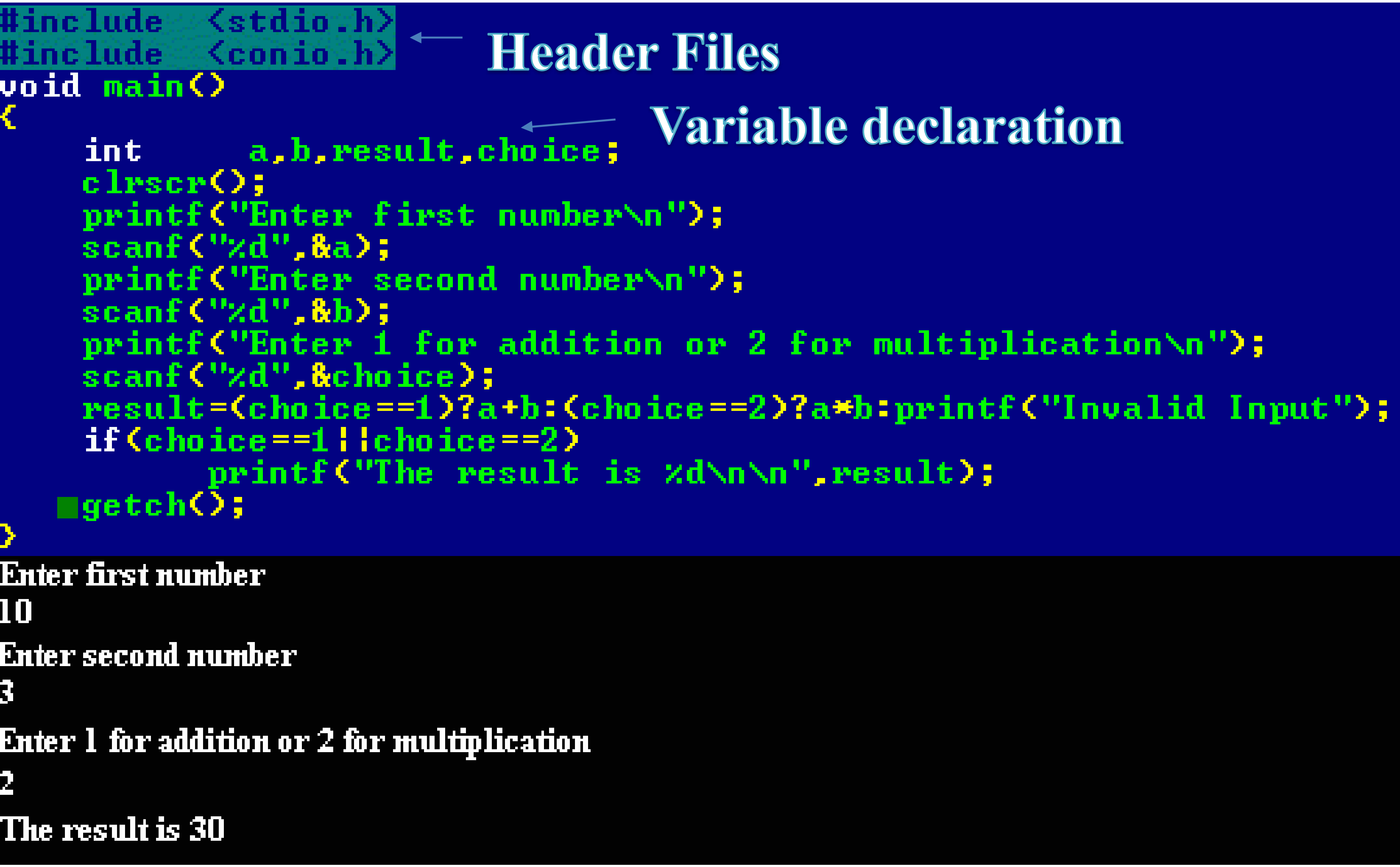
```
a=11;
b=20;
x=(a>b) ? a : b;
```

Exp 1: Exp 2

Identifier

Test Expression





Header Files

Variable declaration

```
#include   <stdio.h>
#include   <conio.h>
void main()
{
    int      a,b,result,choice;
    clrscr();
    printf("Enter first number\n");
    scanf("%d",&a);
    printf("Enter second number\n");
    scanf("%d",&b);
    printf("Enter 1 for addition or 2 for multiplication\n");
    scanf("%d",&choice);
    result=(choice==1)?a+b:(choice==2)?a*b:printf("Invalid Input");
    if(choice==1||choice==2)
          printf("The result is %d\n\n",result);
   getch();
}
```

```
Enter first number
10
Enter second number
3
Enter 1 for addition or 2 for multiplication
2
The result is 30
```

7. Bitwise operator

- C supports bitwise operators for manipulation of data at bit level.
- Bitwise operators may not be applied to float or double.
- Bitwise operators are as follows:

&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right
~	One’s Complements

8. Special operator

- C supports some special operators such as:

  comma operator “,”

  int a=5,b=6;

  size of operator “sizeof()”

  Address operator “&”

  pointer operator “*”

  member selection operator “. and -> ”

Precedence of operators

- Precedence establishes the hierarchy of one set of operators over another when an arithmetic expression has to be evaluated.
- It refers to the order in which c evaluates operators.
- The evaluation of operators in an arithmetic

  expression takes place from left to right for operators having equal precedence .

Precedence of operators

BODMAS RULE-

Brackets of Division Multiplication Addition Subtraction

Brackets will have the highest precedence and have to be evaluated first, then comes of , then comes division, multiplication, addition and finally subtraction.

C language uses some rules in evaluating the expressions and they r called as precedence rules or sometimes also referred to as hierarchy of operations, with some operators with highest precedence and some with least.

The 2 distinct priority levels of arithmetic operators in c are-

Highest priority : * / %

Lowest priority : + -

Associativity of operators

- Associativity tells how an operator associates with its operands.

for eg:

Associativity means whether an expression like x R y R z (where R is a operator such as + or <=) should be evaluated `left-to-right' i.e. as (x R y) R z or `right-to-left' i.e. as x R (y R z)

The assignment operator = associates from right to left.

- Hence the expression on the right is evaluated first and its value is assigned to the variable on the left.
- Associativity also refers to the order in which c evaluates operators in an expression having same precedence.
- Such type of operator can operate either left to right or vice versa.
- The operator **() function call has highest precedence** & the **comma operator has lowest precedence**
- **All unary , conditional & assignment operators associate RIGHT TO LEFT** .
- All other remaining operators associate LEFT TO RIGHT

Rules for evaluation of expression

1. First **parenthesized** sub expression from left to right are evaluated.
2. **If** parentheses are **nested,** the evaluation begins with the **innermost sub expression**
3. The precedence rule is applied in determining the order of application of operators in evaluating sub expressions
4. **The associatively rule is applied when 2 or more operators of the same precedence level appear in a sub expression.**
5. Arithmetic expressions are evaluated from left to right using the rules of precedence
6. When parentheses are used, the expressions within parentheses assume highest priority

Hierarchy of operators

Operator	Description	Associativity
( ), [ ]	Function call, array element reference	Left to Right
+, -, ++, - - ,!,~,*,&	Unary plus, minus, increment, decrement, logical negation, 1’s complement, pointer reference, address	Right to Left
*, / , %	Multiplication, division, modulus	Left to Right

Type Casting

- Type casting is a way to convert a variable from one data type to another data type.
- When **variables and constants of different types are combined** in an expression then they are **converted to same data type.** The process of converting one predefined type into another is called type conversion.

DATATYPE 1 → DATATYPE 2

Implicit Type Casting

- When the type conversion is performed **automatically by the compiler** without programmers intervention, such type of conversion is known as implicit type conversion or type promotion.
- For example when you add values having different data types, both values are first converted to the same type: when a short int value and an int value are added together, the short int value is converted to the int type.

int + short int → int

- C does implicit DataType conversion when the need arises.
- When a floating point value is assigned to an integer variable, the decimal portion is truncated.

When a value 156.43 is assigned to an integer variable, 156 is stored and the decimal portion is discarded.

If an integer 200 is assigned to a floating point variable, the value is converted to 200.000000 and stored.

(integer type variable)a= 156.43 → 156.~~43~~

(float type variable) float b = 200 → 200.000000

Explicit Type Casting

- The type conversion performed by the programmer by posing the data type of the expression of specific type is known as explicit type conversion.
- Type casting in c is done in the following form:

  (data_type) expression;

  where, data_type is any valid c data type, and expression may be constant, variable or expression.

  For example,

```
x=(int)a+b*d;
```

Example

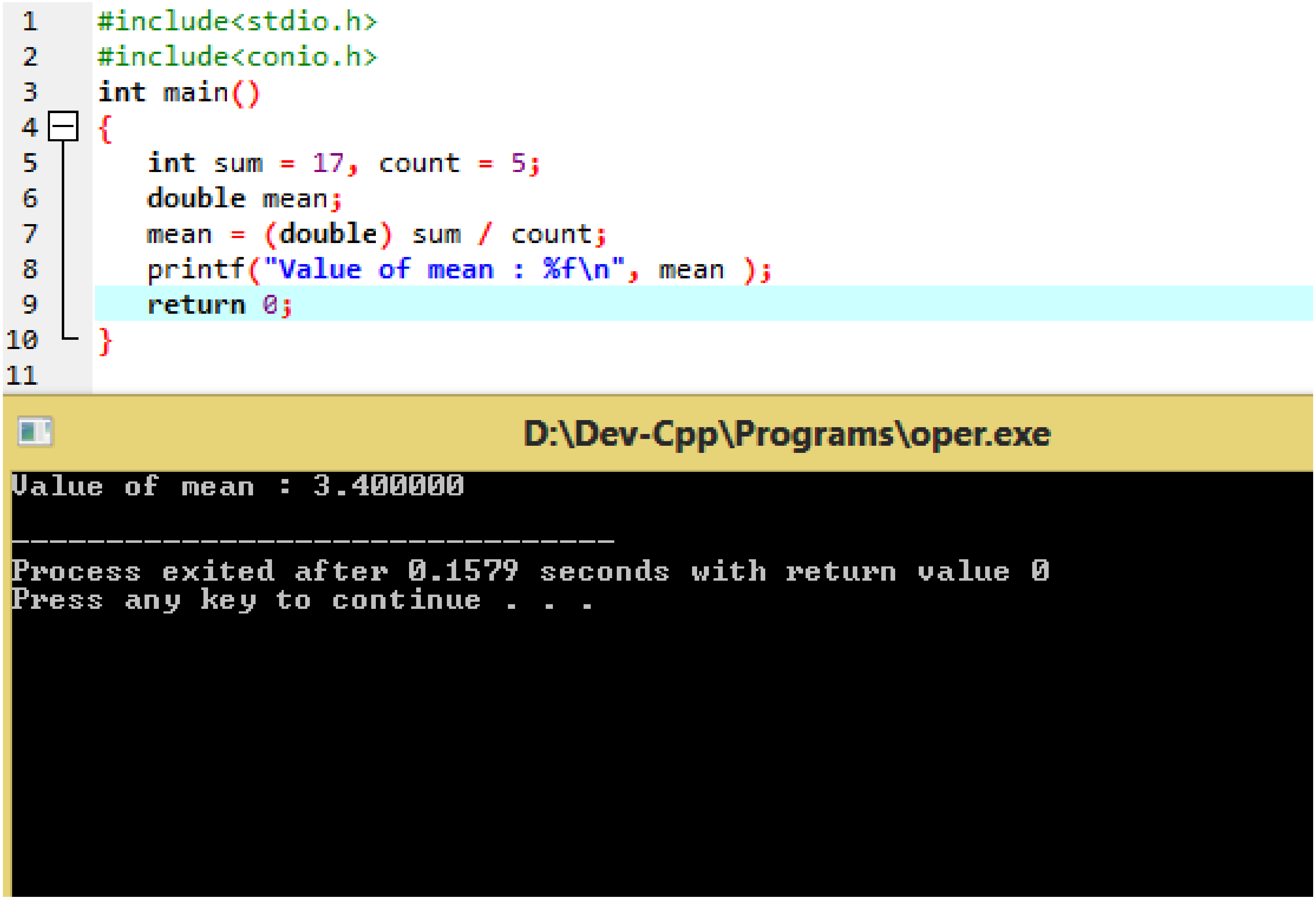
```
#include <stdio.h>
main()
{
   int sum = 17, count = 5;
   double mean;
   mean = (double) sum / count;
   printf("Value of mean : %f\n",
mean );
}
```

Output is

```
Value of mean : 3.400000
```

It should be noted here that the cast operator has precedence over division,

so the value of sum is first converted to type double and finally it gets divided by count yielding a double value.



```c
#include<stdio.h>
#include<conio.h>
int main()
{
    int sum = 17, count = 5;
    double mean;
    mean = (double) sum / count;
    printf("Value of mean : %f\n", mean );
    return 0;
}
```

```
Value of mean : 3.400000

--------------------------------
Process exited after 0.1579 seconds with return value 0
Press any key to continue . . .
```

Rules for Implicit Type Casting

The following rules have to be followed while converting the expression from one type to another to avoid the loss of information:

- All integer types to be converted to float.
- All float types to be converted to double.
- All character types to be converted to integer.

Creating your own header file

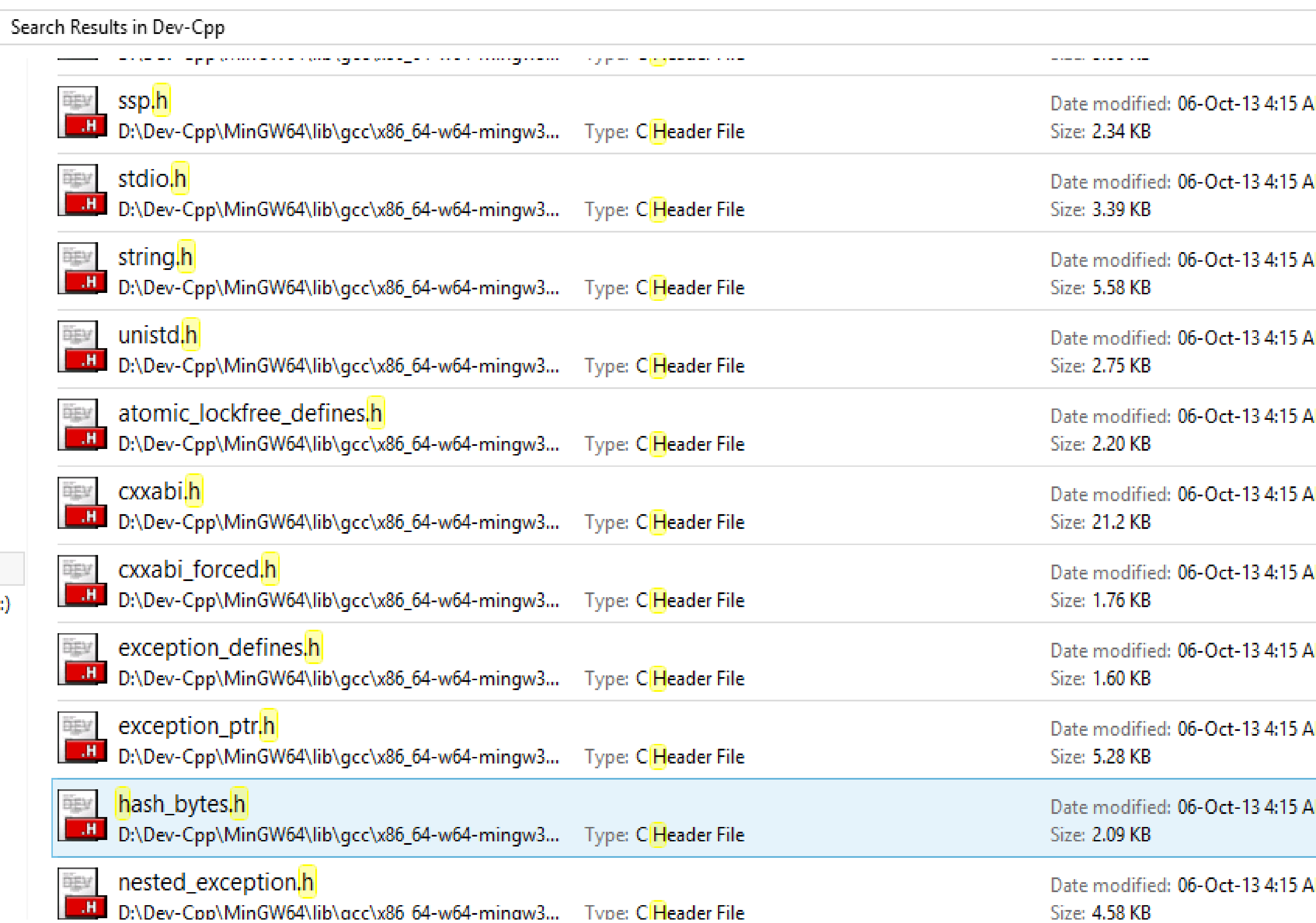
- This topic is not in your syllabus.
- Read only if interested.

- STEP 1: create new file and write following syntax in it............

```
 #ifndef<space>__NAME_H  //NAME OF YOUR HEADER FILE
 #define<space>__NAME_H
int factorial(int  num) //define your function directly
{
     int i,f=1;
     for(i=n;i>=1;i--)
          f*=i;
    return(f);
}
/* if you want to print something in function you would require printf( ) so
you can include stdio.h in function definition also
#ifndef<space>__NAME_H  //NAME OF YOUR HEADER FILE
 #define<space>__NAME_H
#include<stdio.h>
.......................................
....................
*/
```

- SAVE THIS FILE AS NAME.H IN INCLUDE FOLDER OF TC //YOU CAN USE ANY NAME //DEFINED ABOVE
- NOW CREATE NEW FILE SAY ANKIT.C /ANKIT.CPP
- TO USE HEADER FILE WRITE

```
#include<name.h>  //your header file name
#include<stdio.h>
#include<conio.h>
void main( )
{ int c;
  clrscr( );
  c=factorial(5);
  printf("factorial is %d",c);  //change code accordingly in cpp
  getch( );
}
```



Search Results in Dev-Cpp

ssp.h
D:\Dev-Cpp\MinGW64\lib\gcc\x86_64-w64-mingw3... Type: C Header File
Date modified: 06-Oct-13 4:15 A
Size: 2.34 KB

stdio.h
D:\Dev-Cpp\MinGW64\lib\gcc\x86_64-w64-mingw3... Type: C Header File
Date modified: 06-Oct-13 4:15 A
Size: 3.39 KB

string.h
D:\Dev-Cpp\MinGW64\lib\gcc\x86_64-w64-mingw3... Type: C Header File
Date modified: 06-Oct-13 4:15 A
Size: 5.58 KB

unistd.h
D:\Dev-Cpp\MinGW64\lib\gcc\x86_64-w64-mingw3... Type: C Header File
Date modified: 06-Oct-13 4:15 A
Size: 2.75 KB

atomic_lockfree_defines.h
D:\Dev-Cpp\MinGW64\lib\gcc\x86_64-w64-mingw3... Type: C Header File
Date modified: 06-Oct-13 4:15 A
Size: 2.20 KB

cxxabi.h
D:\Dev-Cpp\MinGW64\lib\gcc\x86_64-w64-mingw3... Type: C Header File
Date modified: 06-Oct-13 4:15 A
Size: 21.2 KB

cxxabi_forced.h
D:\Dev-Cpp\MinGW64\lib\gcc\x86_64-w64-mingw3... Type: C Header File
Date modified: 06-Oct-13 4:15 A
Size: 1.76 KB

exception_defines.h
D:\Dev-Cpp\MinGW64\lib\gcc\x86_64-w64-mingw3... Type: C Header File
Date modified: 06-Oct-13 4:15 A
Size: 1.60 KB

exception_ptr.h
D:\Dev-Cpp\MinGW64\lib\gcc\x86_64-w64-mingw3... Type: C Header File
Date modified: 06-Oct-13 4:15 A
Size: 5.28 KB

hash_bytes.h
D:\Dev-Cpp\MinGW64\lib\gcc\x86_64-w64-mingw3... Type: C Header File
Date modified: 06-Oct-13 4:15 A
Size: 2.09 KB

nested_exception.h
D:\Dev-Cpp\MinGW64\lib\gcc\x86_64-w64-mingw3... Type: C Header File
Date modified: 06-Oct-13 4:15 A
Size: 4.58 KB

Thank you