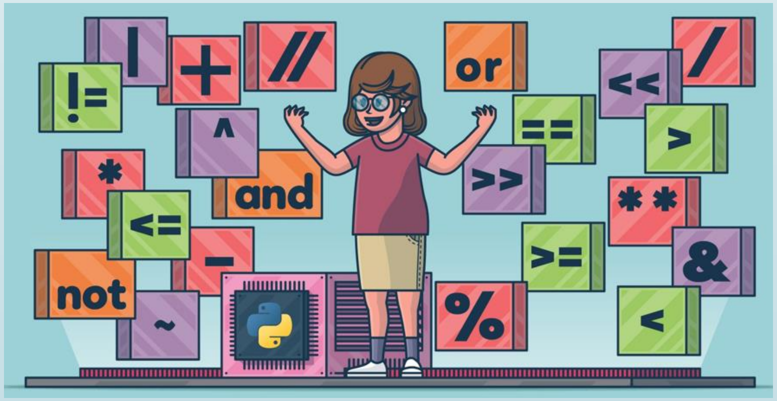
Expressions and Operators

Lecture 03 – ICT1132

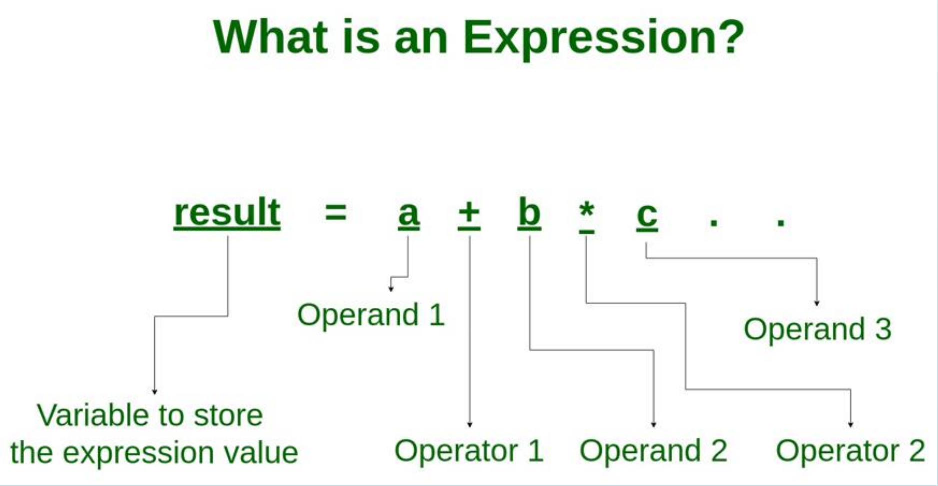


Piyumi Wijerathna
Department of ICT
Faculty of Technology

Expressions

- An expression is a combination of variables, constants, operators, and function invocations, which are constructed according to the syntax of the language.

Example: area = pi * pow(r , 2);



There are several types of expressions:

1. expression that assign a value to a variable.

   x = 7

   This expression uses the = operator to assign the value seven to the variable x.

2. expression that simply have a value.

   3 + 4

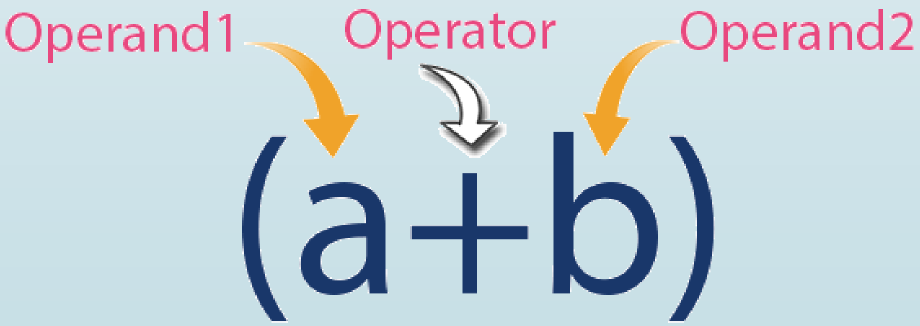
   expression uses the + operator to add three and four together without assigning the result, seven, to a variable.

3. Expression that combines two or more expressions.

   x+3 = 5x+6

Operands and Operators

- Every expression consists of at least one operator and at least one or more operands.
- Operands are values, whereas operators are symbols that represent particular computational (mathematical or logical) actions.



Operand1

Operator

Operand2

(a+b)

Arithmetic Operators

- Use to perform mathematical operations on numeric values.

Operator	Meaning	Example	Result if A=6, B=2
+	Addition	A + B	8
-	Subtraction	A - B	4
*	Multiplication	A * B	12
/	Division	A / B	3
%	Modulus	A % B	0

Mixed Mode Arithmetic Expression

- If operands in an expression contains both INTEGER and REAL(Floating Points) constants or variables, it is a mixed mode arithmetic expression.
- INTEGER operands are always converted to REAL before carrying out any computations.
- Hence the result of a mixed mode expression is of REAL type.

Operand	INTEGER	REAL
INTEGER	integer	real
REAL	real	real

Examples:

- 1 + 2.5
- 4.0 / 2

- If the computation involves signed and unsigned values the signed values are treated as unsigned.

Arithmetic Operations On Characters

- Whenever a character variable is used in the expression then it is automatically converted into Integer Value called ASCII value.
- All the characters can be Manipulated by the ASCII Integer Value.

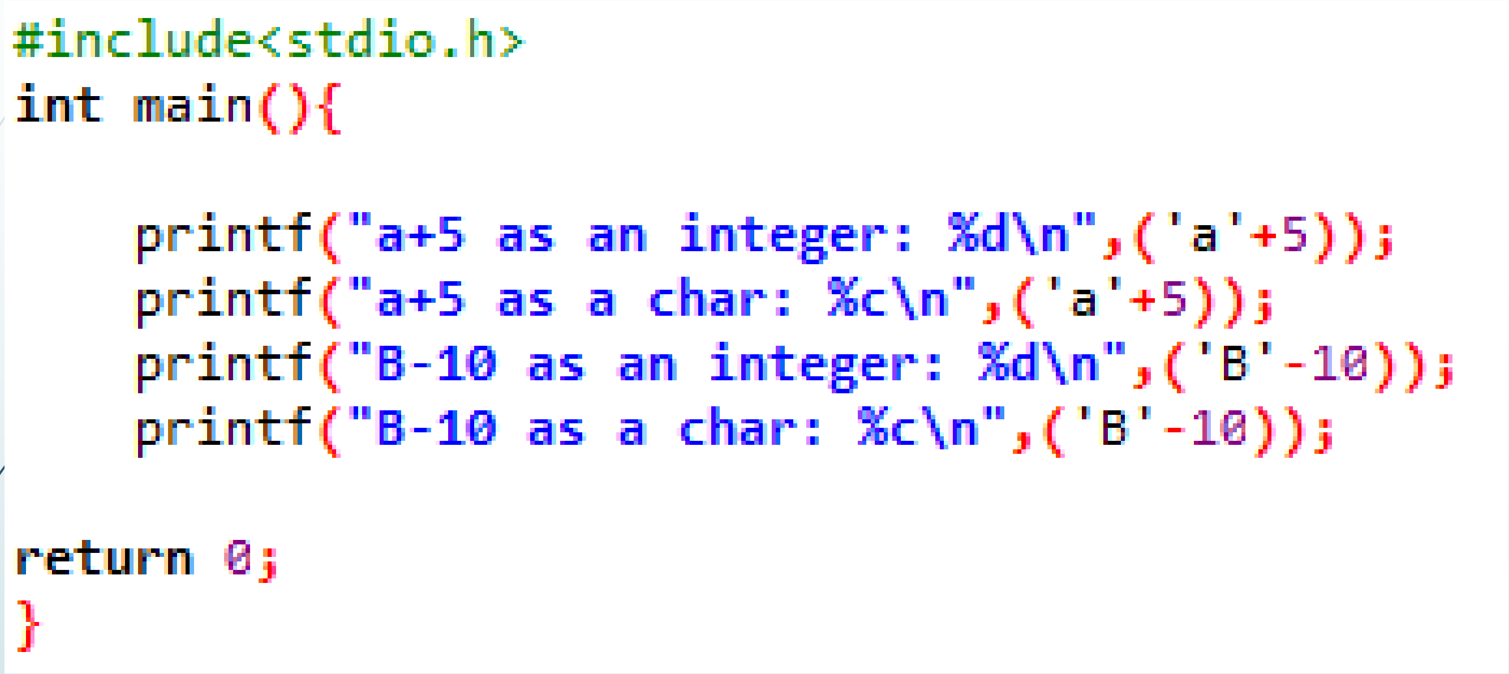
Examples:

ASCII value of ‘a’ is 97.

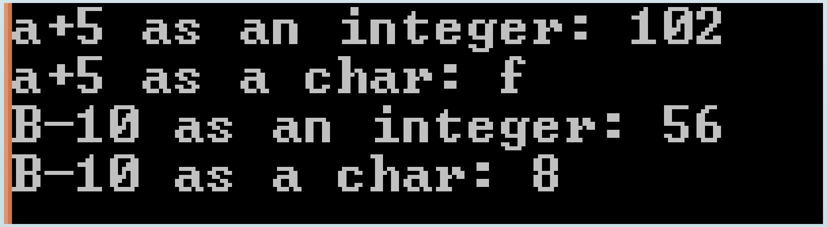
ASCII value of ‘B’ is 66.

- ‘a’ + 5 = 102 → ’f’
- ‘B’ – 10 = 56 → 8

Example



```
#include<stdio.h>
int main(){

    printf("a+5 as an integer: %d\n",('a'+5));
    printf("a+5 as a char: %c\n",('a'+5));
    printf("B-10 as an integer: %d\n",('B'-10));
    printf("B-10 as a char: %c\n",('B'-10));

return 0;
}
```



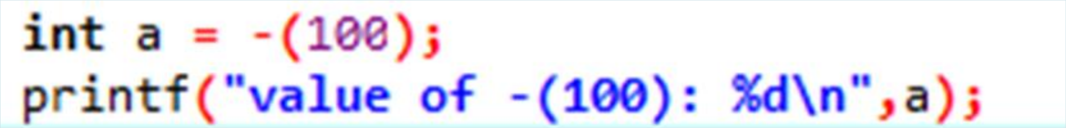
```
a+5 as an integer: 102
a+5 as a char: f
B-10 as an integer: 56
B-10 as a char: 8
```

Divide(/) and Modulus Division(%) Operators

- When both operands of the divide operator are integers, the result is the integer quotient.
  - 5/2 = 2
- When at least one of the operands of the divide operator is real, the result is the real number quotient.
  - 5/2.0 = 2.500000
- To obtain the integer remainder of the division of two integers use the modulus operator.
  - 5%2 = 1
- The modulus operator should not be used with negative operands since the result is not consistent from one computer to another.

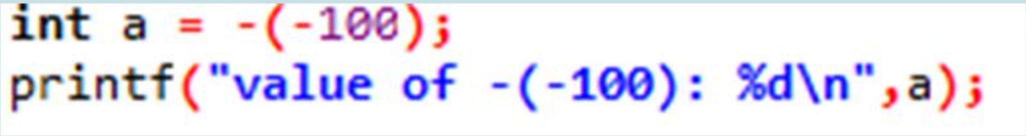
Unary Minus Operator

- “Unary Operator” is an operator that performs on a single operand in an expression.
- “Unary Minus Operator” change the sign of the operand.
- Same as multiplying the operand by –1.
- Different from arithmetic subtraction operator (requires two operands).



```
int a = -(100);
printf("value of -(100): %d\n",a);
```

Output : -100



```
int a = -(-100);
printf("value of -(-100): %d\n",a);
```

Output : 100

Assignment Operators

- Use to assign the right side value to the left side variable in an expression.

Operators	Example/Description	Result, if sum=10
=	**sum = 10**; 10 is assigned to variable sum	
+=	sum += 10; This is same as **sum = sum + 10**	20
-=	sum -= 10; This is same as **sum = sum – 10**	0
*=	sum *= 10; This is same as **sum = sum * 10**	100
/=	sum /= 10; This is same as **sum = sum / 10**	1
%=	sum %= 10; This is same as **sum = sum % 10**	0

```
#include<stdio.h>
int main(){
        int a = 2;
        printf(“Value of a: %d\n”, a);
        a += 5;
        printf(“Value of a now: %d\n”, a);

        int b = 9;
        printf(“Value of b: %d\n”, b);
        b -= 3;
        printf(“Value of b now: %d\n”, b);

        return 0;
}
```

Output

```
Value of a: 2
Value of a now: 7
Value of b: 9
Value of b now: 6
```

Relational Operators

- Find the relationship between left and right operands.

Operators	Meaning	Example	Example	result
==	Equal	x == y	3==4	false
!=	Not equal	x != y	5!=2	true
>	Greater than	x > y	3>2	true
<	Less than	x < y	5<3	false
>=	Greater than or equal	x >= y	2>=6	false
<=	Less than or equal	x <= y	5<=5	true

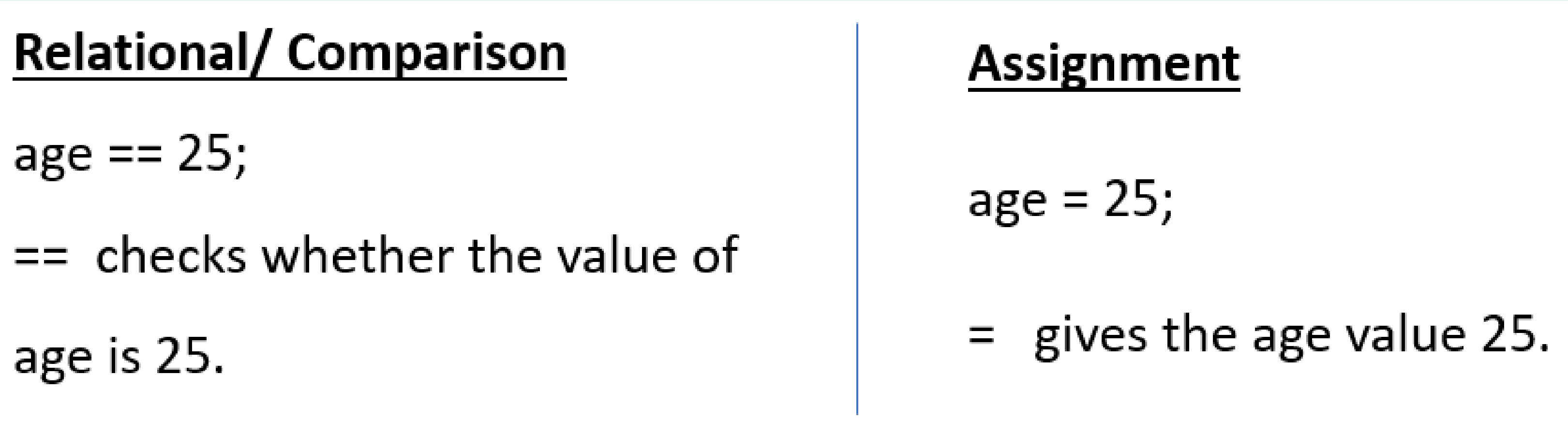
More on Relational Operators

- When an operator consists of two keystrokes there can be no space in between the symbols.

  > = wrong

  >= correct

Note : A common error is to use the assignment operator(=) instead of the equivalence operator(==).



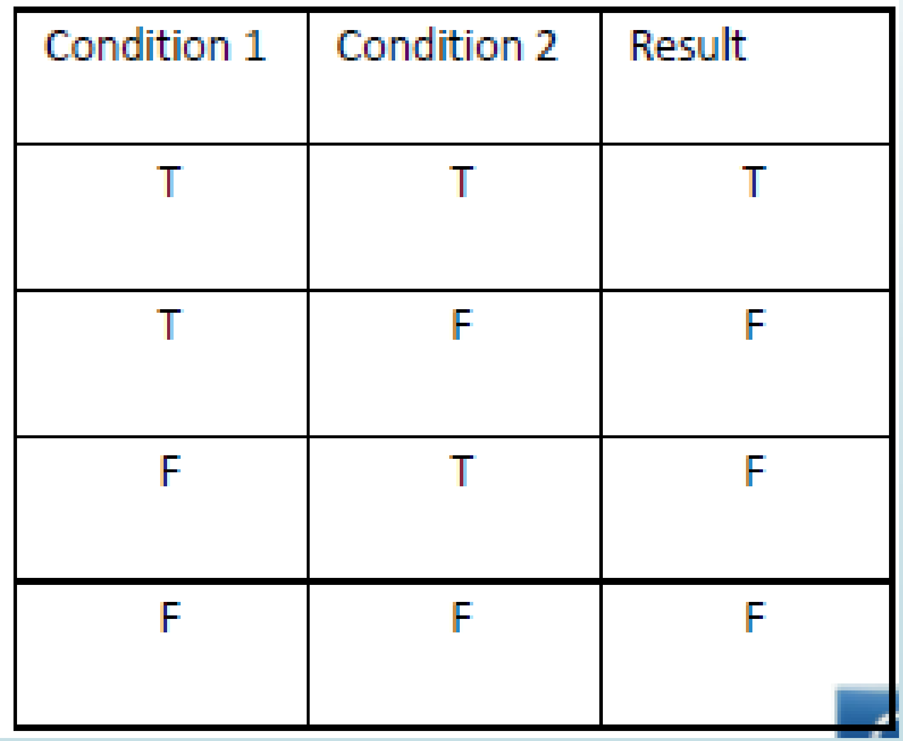
Relational/ Comparison	**Assignment**
age == 25;	age = 25;
== checks whether the value of age is 25.	= gives the age value 25.

Logical Operators

- These operators are used to perform logical operations on the given expressions.

1. Logical **AND** Operator

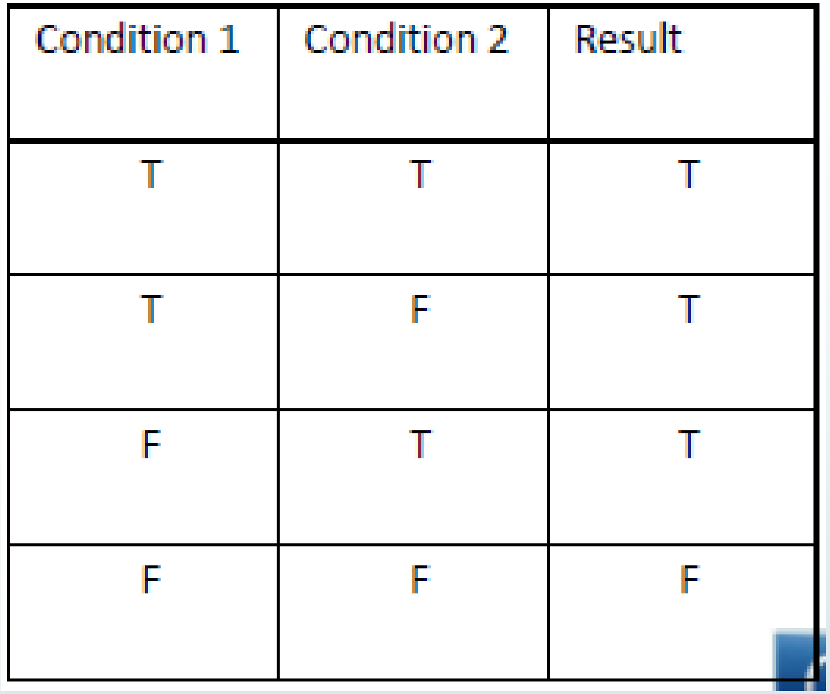
A && B



Condition 1	Condition 2	Result
T	T	T
T	F	F
F	T	F
F	F	F

2. Logical **OR** Operator

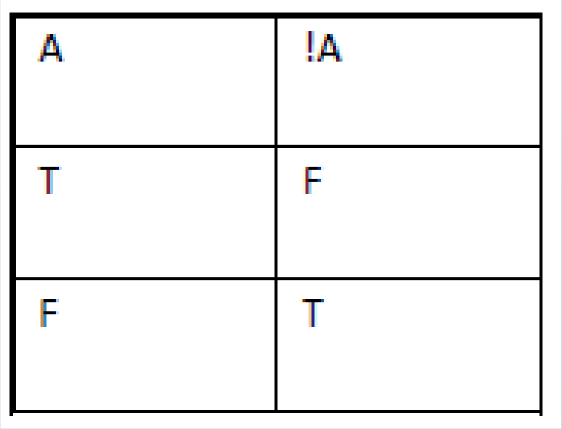
A || B



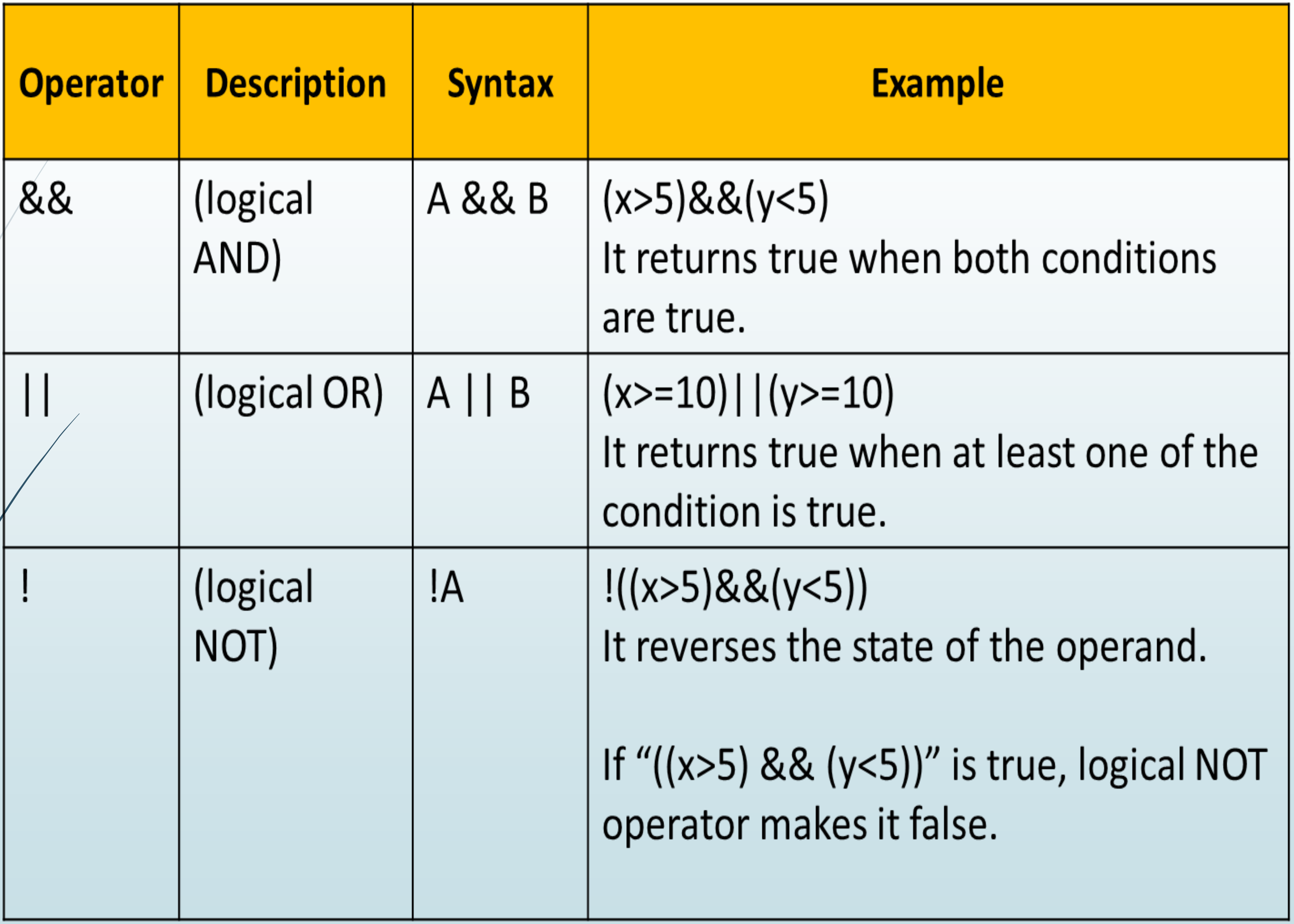
Condition 1	Condition 2	Result
T	T	T
T	F	T
F	T	T
F	F	F

3. Logical **NOT** Operator

! A



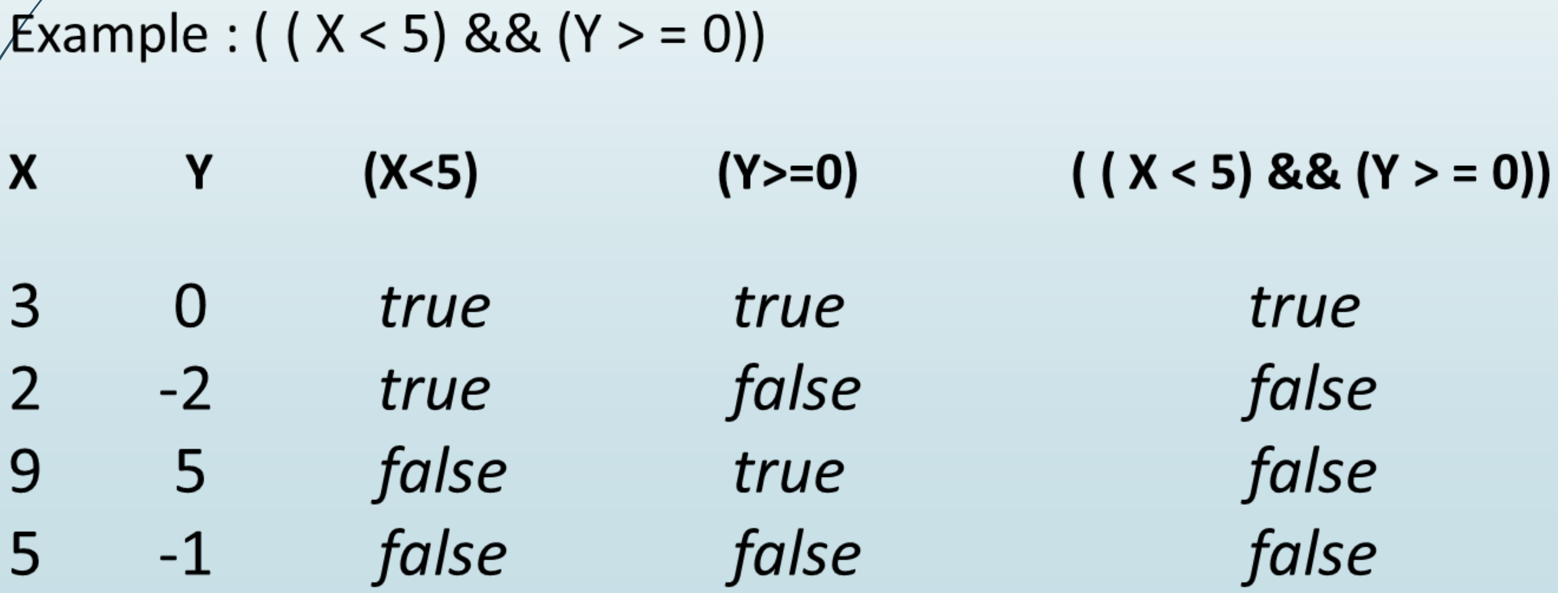
A	!A
T	F
F	T



Operator	Description	Syntax	Example
&&	(logical AND)	A && B	(x>5)&&(y<5) It returns true when both conditions are true.
||	(logical OR)	A || B	(x>=10)||(y>=10) It returns true when at least one of the condition is true.
!	(logical NOT)	!A	!((x>5)&&(y<5)) It reverses the state of the operand. If “((x>5) && (y<5))” is true, logical NOT operator makes it false.

Logical “And” Operation

- The compound condition resulting from using the logical “and” operation evaluates as true if and only if both expressions are true.



Example : ((X < 5) && (Y > = 0))

X	Y	(X<5)	(Y>=0)	((X < 5) && (Y > = 0))
3	0	*true*	*true*	*true*
2	-2	*true*	*false*	*false*
9	5	*false*	*true*	*false*
5	-1	*false*	*false*	*false*

Logical “OR” Operation

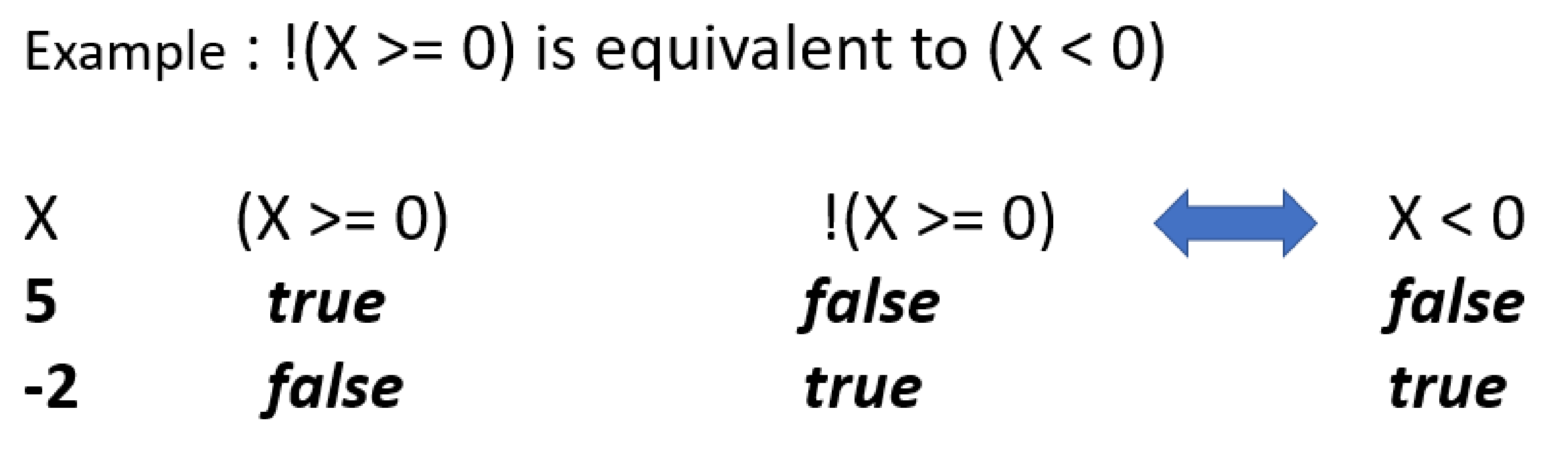
- The compound condition resulting from using the logical “or” operation evaluates as false if and only if both expressions are false.

Example : **((X < 5) | | (Y > = 0))**

X	**Y**	**(X<5)**	**(Y>=0)**	**((X < 5) | | (Y > = 0))**
3	0	*true*	*true*	*true*
2	-2	*true*	*false*	*true*
9	5	*false*	*true*	*true*
5	-1	*false*	*false*	*false*

Logical “NOT” Operation

- The logical “not” operation negates a logical expression.
- If the expression originally evaluated to true then the logical not of that expression evaluates to false.



Example : !(X >= 0) is equivalent to (X < 0)

X	(X >= 0)	!(X >= 0)	⟺	X < 0
5	***true***	***false***		***false***
-2	***false***	***true***		***true***

Short-Circuit Evaluation

- Short-Circuit Evaluation: Short-circuiting is a programming concept by which the compiler skips the execution or evaluation of some sub-expressions in a logical expression.
- The compiler stops evaluating the further sub-expressions as soon as the value of the expression is determined.
- When the computer stops evaluation of a logical expression as soon as it can determine the value –true or false- of the expression, it is referred to as short-circuit evaluation.

```
if (a == b | | c == d | | e == f) {
        // do_something
}
```

- In the above expression, If the expression a == b is true, then c==d and e==f are never evaluated at all because the expression’s result has already been determined.
- Similarly, if the logical AND (&&) operator instead of logical OR (| |) and the expression a == b is false, the compiler will skip evaluating other sub-expressions.

More on Logical Operators

- Operands of a logical operator must be logical values.
- Expressions such as 10 < x < 100 are not valid logical expressions in C.
- Pay attention to the operator precedence rules.
- Clarify operator precedence by using parentheses.

Increment and Decrement Operators

- Increment operators are used to increase the value of the variable by **one**.
- Decrement operators are used to decrease the value of the variable by **one**.

Example:

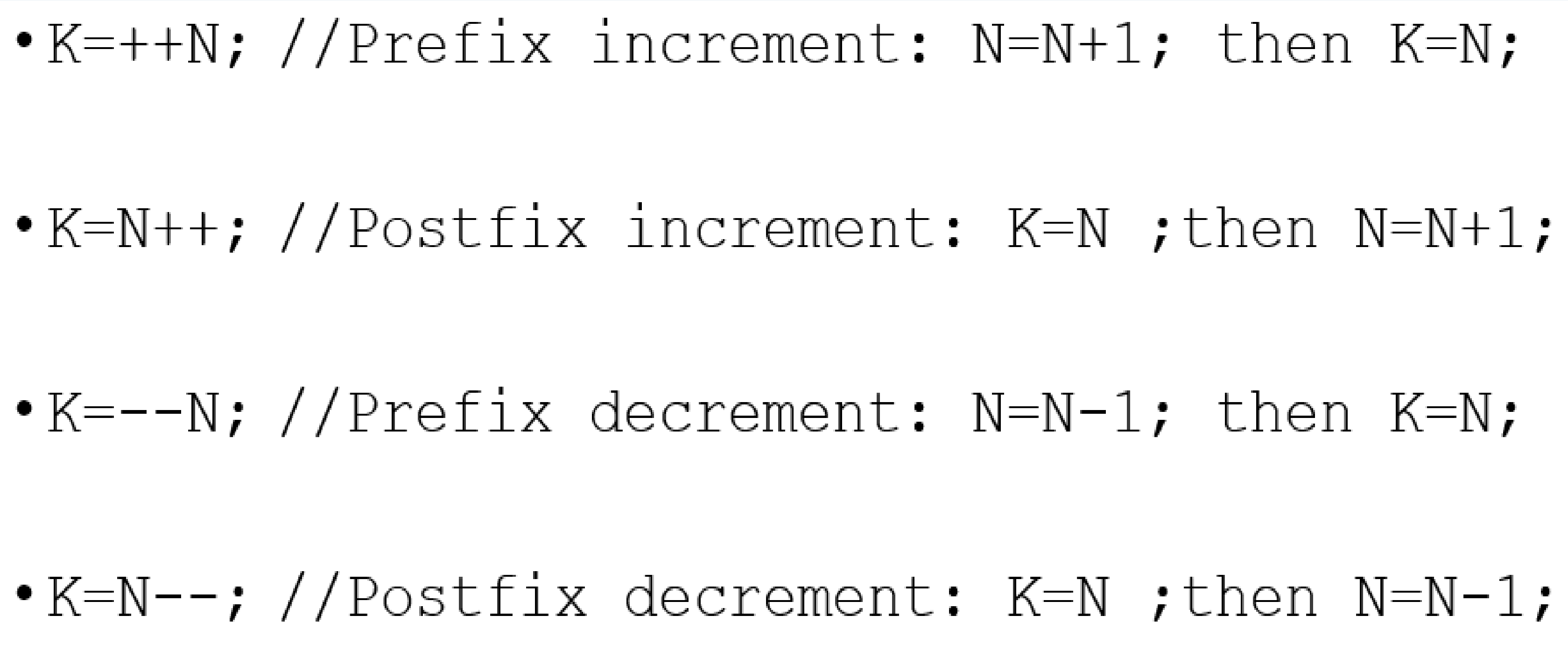
Increment operator : ++ i ; i ++ ;

Decrement operator : – – i ; i – – ;

Pre/Post Increment & Decrement Operators

Operator	Operator/Description
Pre increment (++i)	value of *i* is incremented before assignment
Post increment (i++)	value of *i* is incremented after assignment
Pre decrement (--i)	value of *i* is decremented before assignment
Post decrement (i--)	value of *i* is decremented after assignment

- If you are using prefix form then increment or decrement will be done before rest of the expression.
- If you are using postfix form, then increment or decrement will be done after the complete expression is evaluated.



- K=++N; //Prefix increment: N=N+1; then K=N;
- K=N++; //Postfix increment: K=N ;then N=N+1;
- K=--N; //Prefix decrement: N=N-1; then K=N;
- K=N--; //Postfix decrement: K=N ;then N=N-1;

//Example for increment operators

```
#include<stdio.h>
void main()
{
      int a, b;
      Int  x=10, y=10;

      a = x++;
      b = ++y;

      printf("Value of a : %d“,a);
      printf("Value of b : %d“,b);
}
```

What will be the output?????

//Example for decrement operators

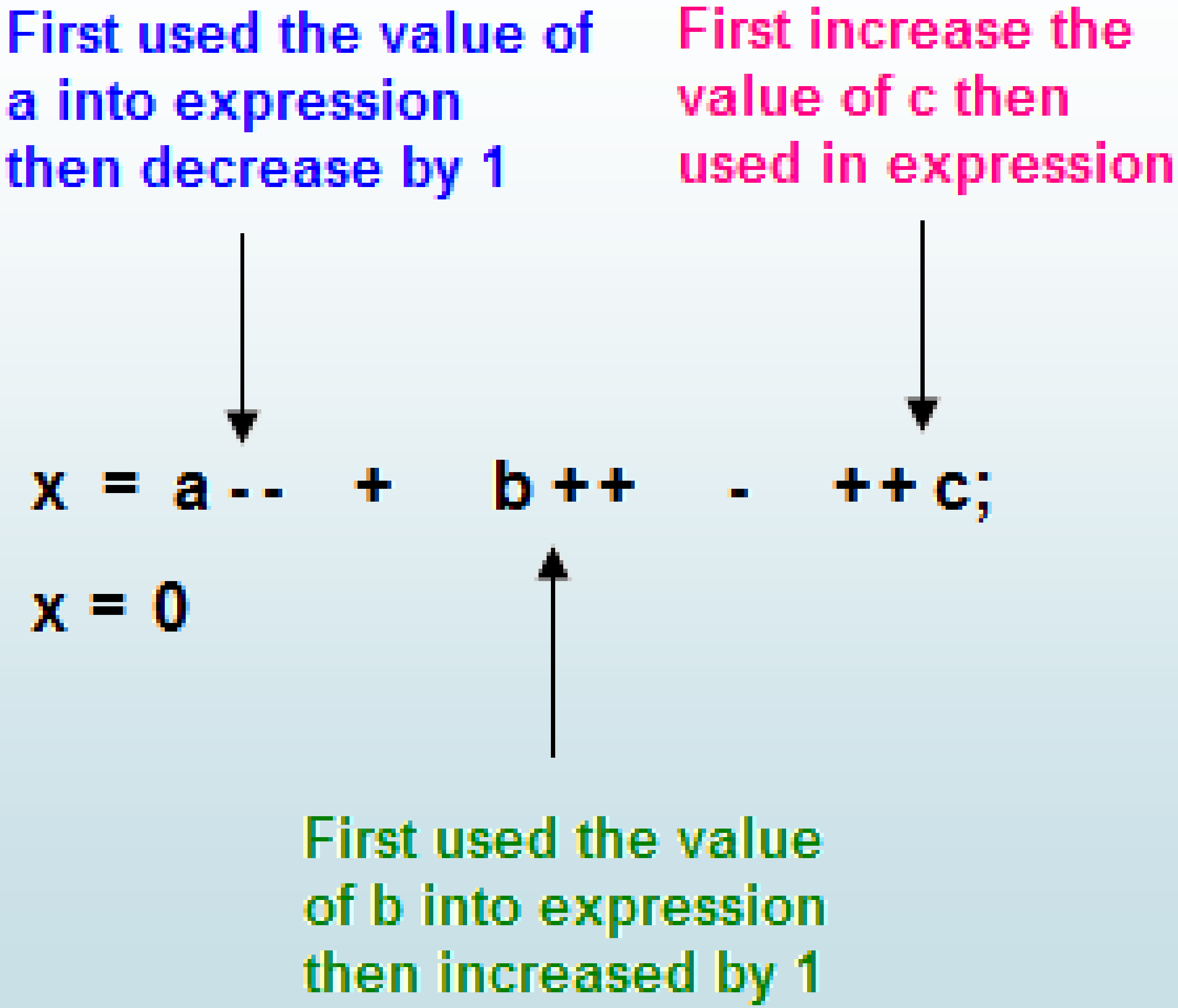
```
#include<stdio.h>
void main()
{
      int a,b;
      Int x=10,y=10;

      a = x--;
      b = --y;

      printf("Value of a : %d“,a);
      printf("Value of b : %d“,b);
}
```

What will be the output?????

Example:

```
#include<stdio.h>
void main()
{
    int x, a, b, c;
    a = 2;
    b = 4;
    c = 5;

    x = a-- + b++ - ++c;

    printf("x: %d",x);
}
```

What will be the output?????



First used the value of a into expression then decrease by 1

First increase the value of c then used in expression

```
x  =  a--   +   b++   -   ++c;
x = 0
```

First used the value of b into expression then increased by 1

Conditional Operators

- Conditional operators return one value if condition is true and returns another value if condition is false.
- This operator is also called as ternary operator.
- Syntax: (Condition ? True_value : false_value)

Example : (A > 100 ? 0 : 1);

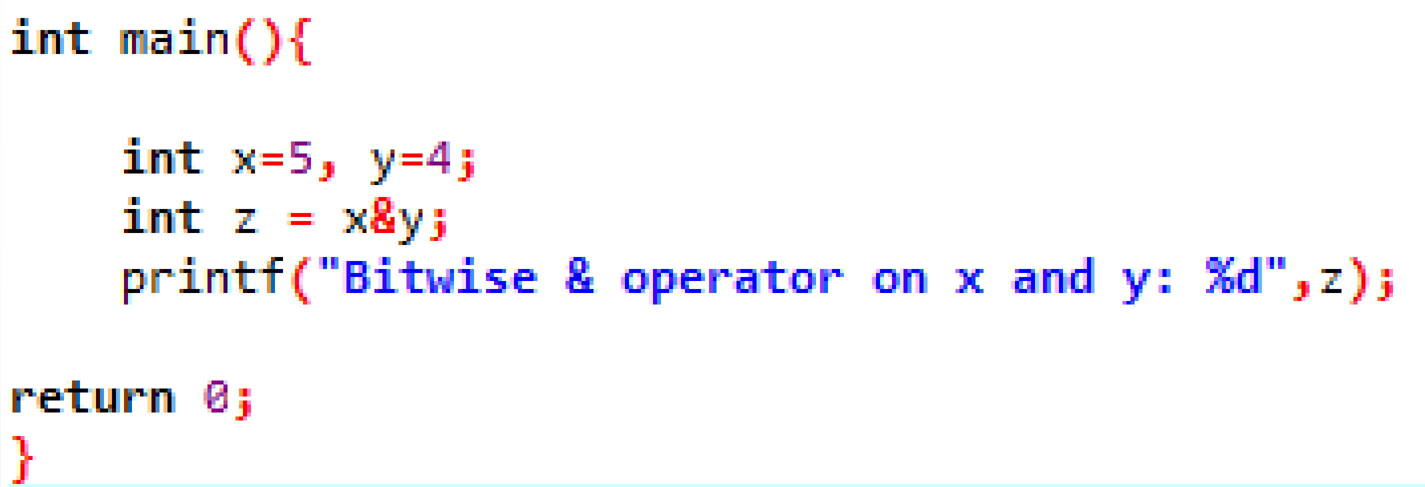
- if A is greater than 100, 0 is returned else 1 is returned.

Bitwise Operators

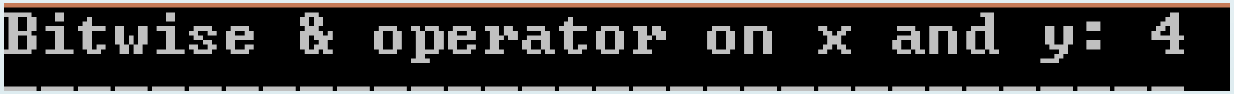
- Used to perform bit operations.
- Only be applied to integral operands →char, short, int, and long, whether signed or unsigned.
- Decimal values are converted into binary values which are the sequence of bits, and bit wise operators work on these bits.

&	bitwise AND
|	bitwise inclusive OR
^	bitwise exclusive OR
<<	left shift
>>	right shift
~	one's complement (unary)

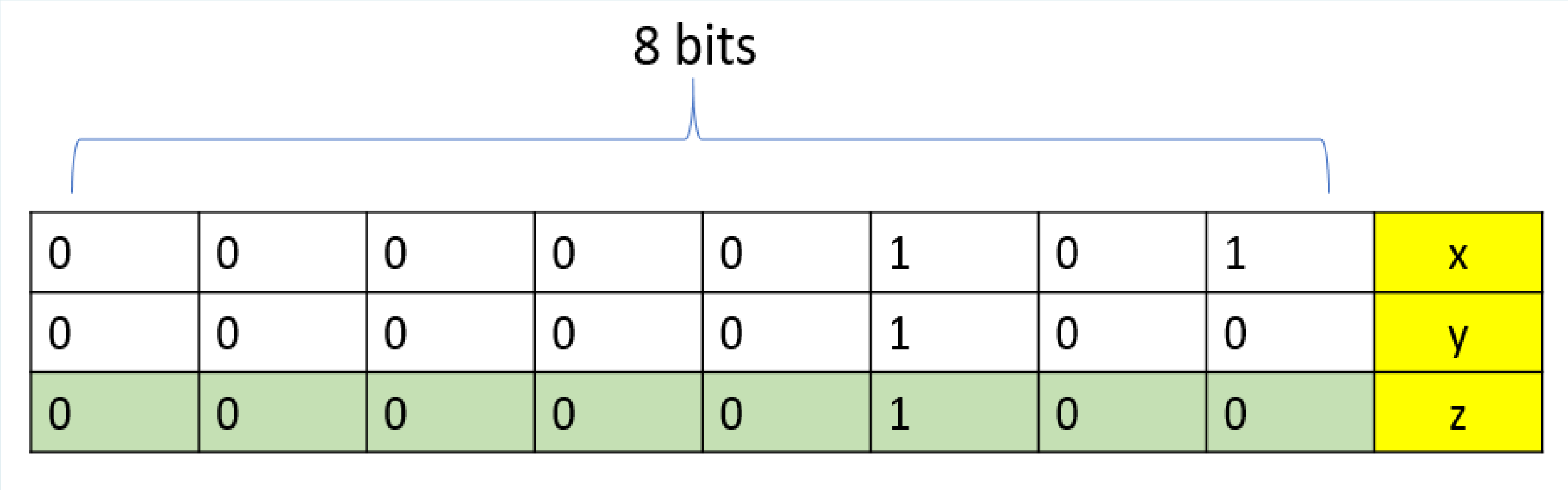
Example



```c
int main(){

    int x=5, y=4;
    int z = x&y;
    printf("Bitwise & operator on x and y: %d",z);

return 0;
}
```



```
Bitwise & operator on x and y: 4
```



8 bits

0	0	0	0	0	1	0	1	x
0	0	0	0	0	1	0	0	y
0	0	0	0	0	1	0	0	z

Operators Order of Precedence

- Operator precedence establishes the priority of an operator with respect to all other operators and decides how an expression is evaluated.
- Some operators have higher precedence than others.
- * operator has a higher precedence than the + operator.
- Within an expression, higher precedence operators will be evaluated first.

  a = 6 + 5 * 3 → Answer= 33 ? or Answer=21 ?

- Parentheses can be used to modify the normal order of execution of an expression.

- Operators with the highest precedence appear at the top of the table.

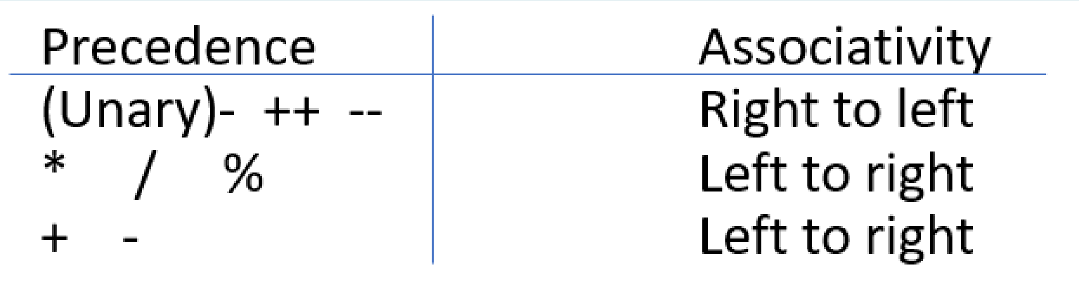
( )	left to right (inside out)
! – (unary) ++ --	right to left
* / %	left to right
+ – (binary)	left to right
< <= > >=	left to right
== !=	left to right
&&	left to right
||	left to right
= += – = *= /= %=	right to left

Operator Associativity

- Operator associativity establishes the order in which operators of the same precedence are to be executed.

Example :

- Operator + and – have the same precedence
- But when evaluating within an expression, left to right execution.



Precedence	Associativity
(Unary)- ++ --	Right to left
* / %	Left to right
+ -	Left to right

Example

```
74 / 10 % 2 * 2 – 10 % ( 5 – 1 )
```

- First deal with ( ).
- Next evaluate * , / and % operators from left to right.
- Finally perform the subtraction.

Answer:

74 / 10 % 2 * 2 – 10 % 4

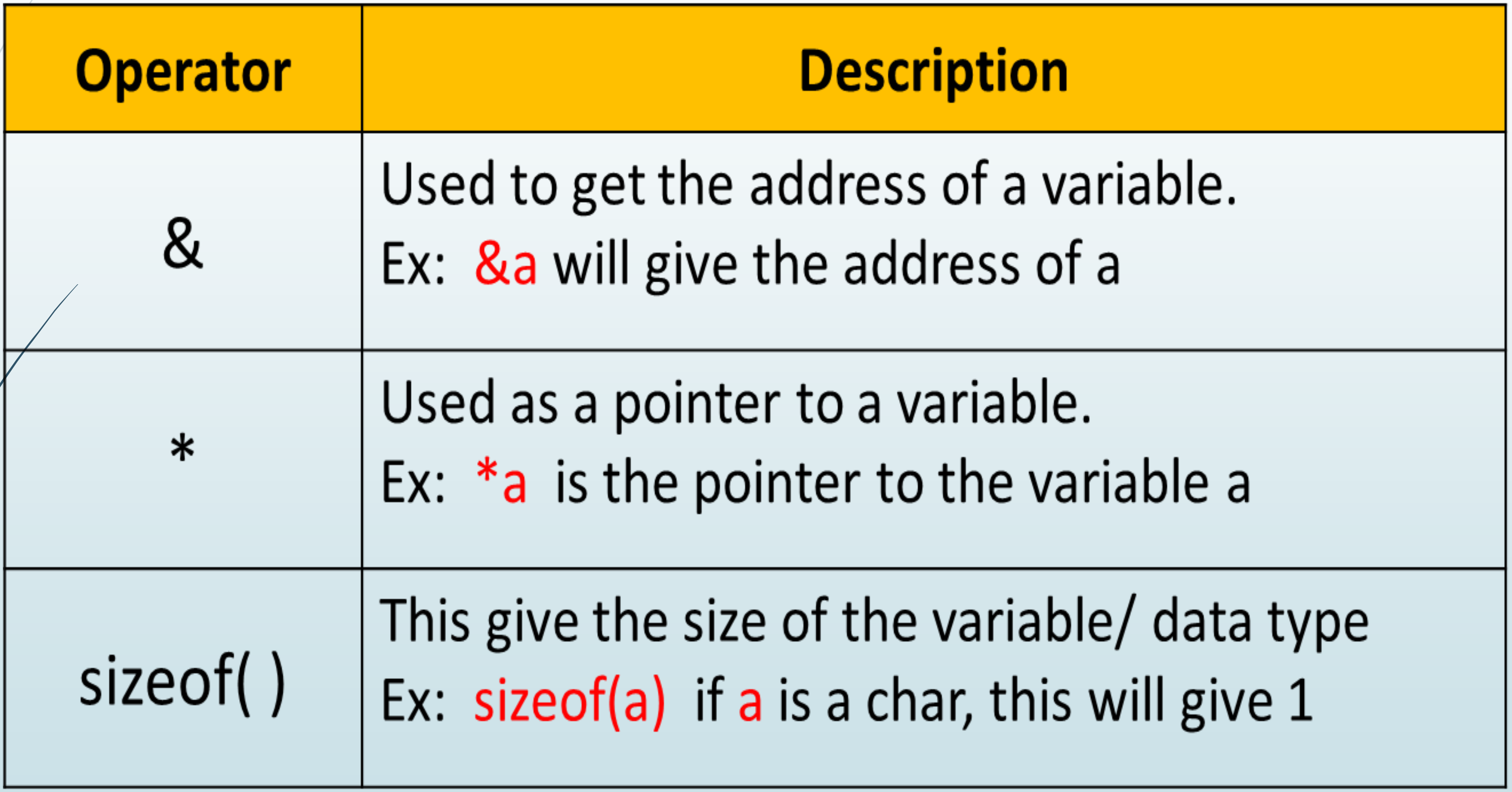
74 / 10 % 4 – 10 % 4

7 % 4 – 10 % 4

3 – 2

1

Special Operators in C



Operator	Description
&	Used to get the address of a variable. Ex: &a will give the address of a
*	Used as a pointer to a variable. Ex: *a is the pointer to the variable a
sizeof( )	This give the size of the variable/ data type Ex: sizeof(a) if a is a char, this will give 1

Compound Statements (Blocks)

Compound Statements (Blocks)

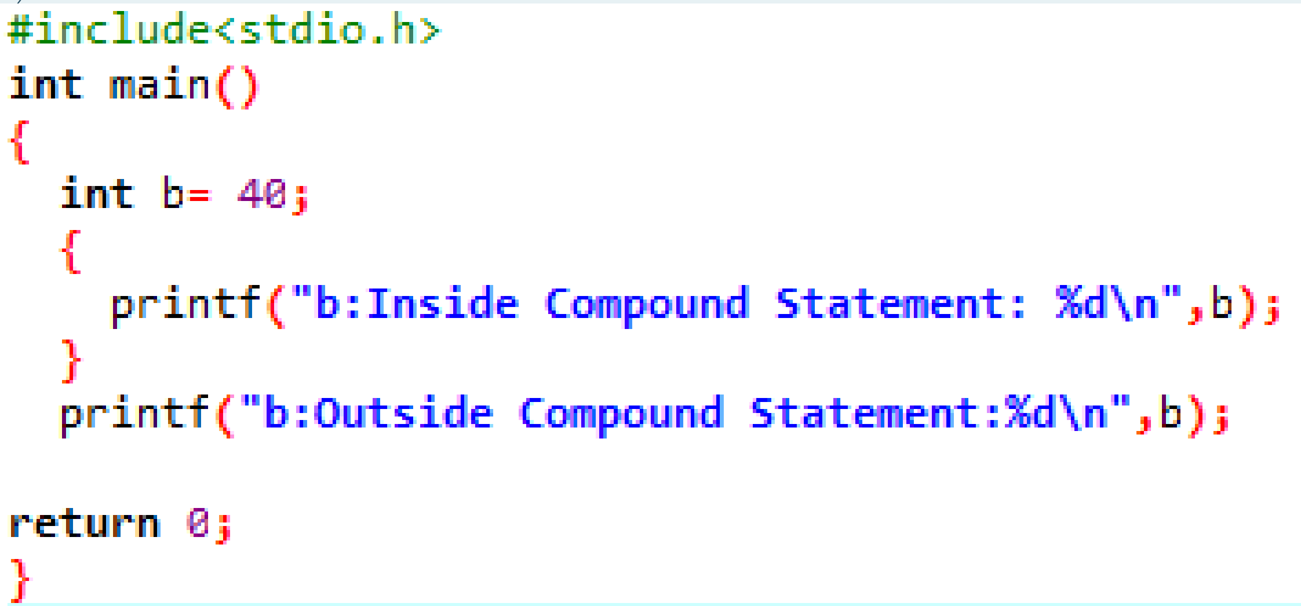
- A compound statement is a list of statements enclosed by pair of curly braces { }.
- The individual statements maybe expression statements, compound statements or control statements.
- All statements within a compound statement work as a single block of code.
- Unlike expression statements, a compound statements does not end with a semicolon.

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

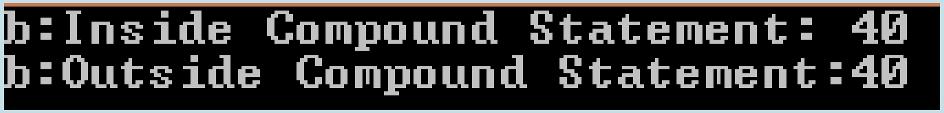
Block Scope 1

- A variable which is declared above (outside) the block, is accessible both inside and outside that block.

- When blocks are nested, the inner block can use variables from the outer block, and no need to declare them again.



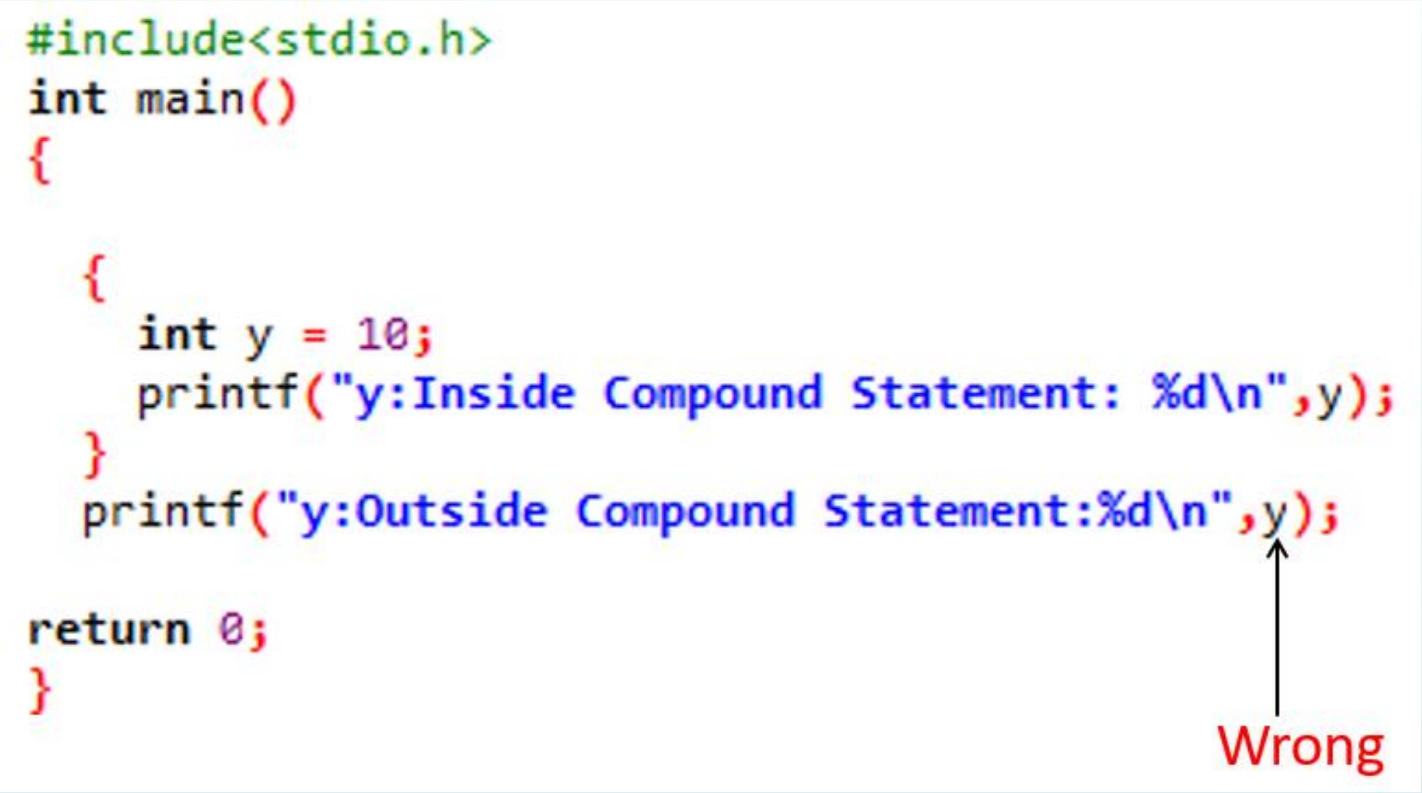
```
#include<stdio.h>
int main()
{
  int b= 40;
  {
    printf("b:Inside Compound Statement: %d\n",b);
  }
  printf("b:Outside Compound Statement:%d\n",b);

return 0;
}
```



```
b:Inside Compound Statement: 40
b:Outside Compound Statement:40
```

Block Scope 2

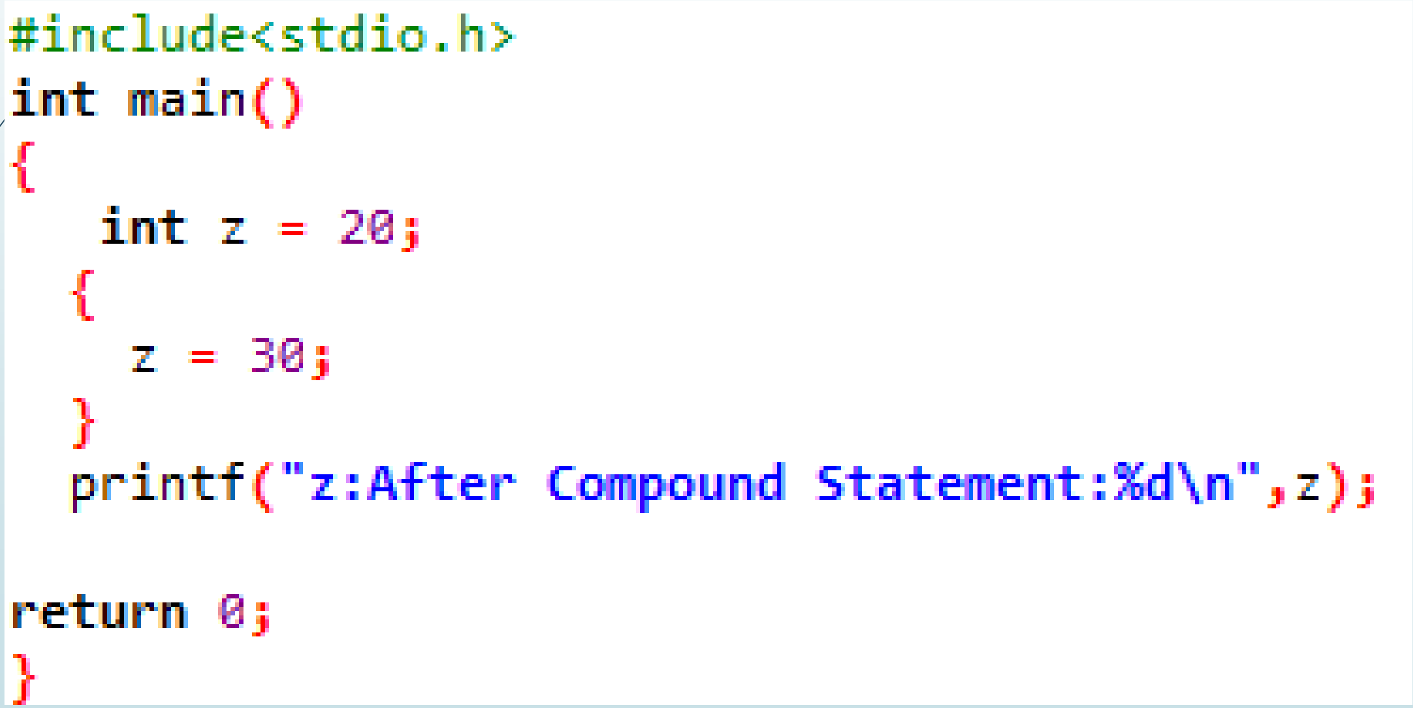
- A variable which is declared inside the block, it is valid only within that block of code(not accessible outside the block).



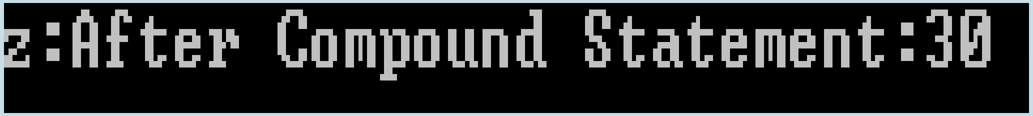
```
#include<stdio.h>
int main()
{

  {
    int y = 10;
    printf("y:Inside Compound Statement: %d\n",y);
  }
  printf("y:Outside Compound Statement:%d\n",y);

return 0;
}
```

Block Scope 3

- However for already declared variables, any changes done within the block will be released back to the system when the block of code is executed.



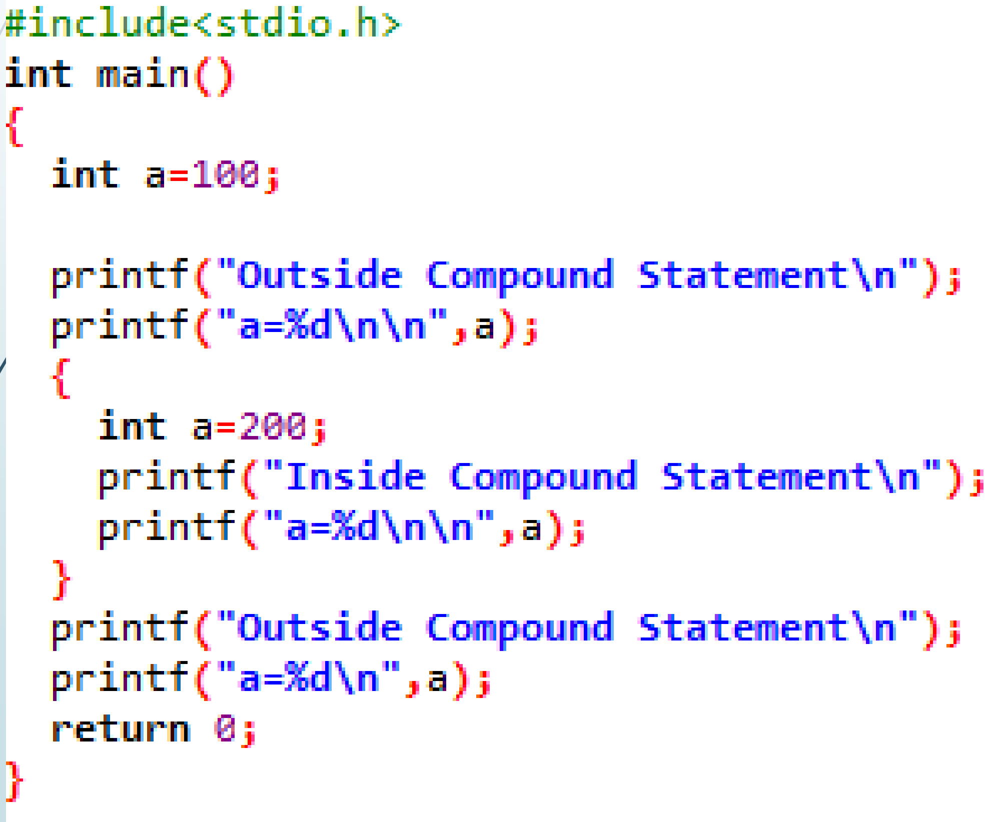
```
#include<stdio.h>
int main()
{
   int z = 20;
  {
    z = 30;
  }
  printf("z:After Compound Statement:%d\n",z);

return 0;
}
```



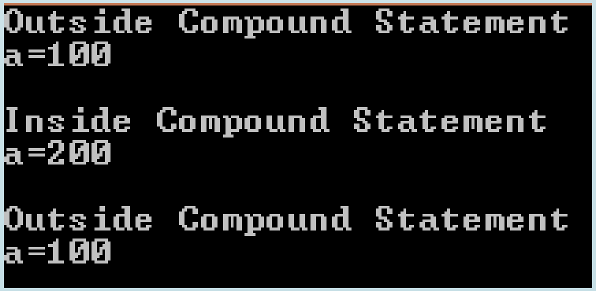
```
z:After Compound Statement:30
```

Block Scope 4

- It is possible to declare a variable with the same name both inside and outside the Blocks.

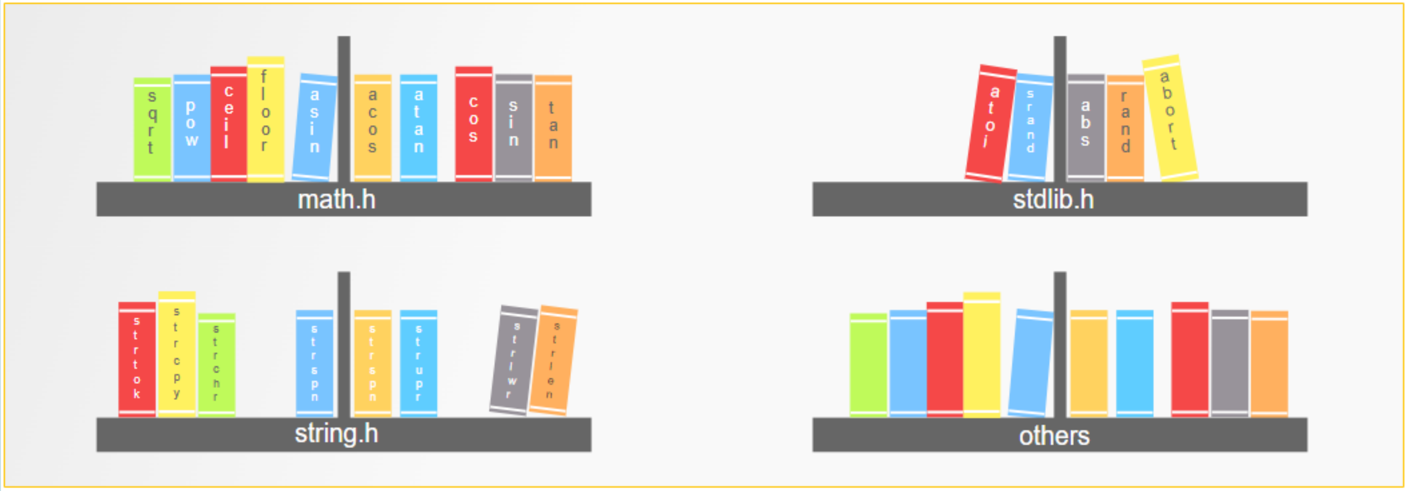


```c
#include<stdio.h>
int main()
{
   int a=100;

   printf("Outside Compound Statement\n");
   printf("a=%d\n\n",a);
   {
      int a=200;
      printf("Inside Compound Statement\n");
      printf("a=%d\n\n",a);
   }
   printf("Outside Compound Statement\n");
   printf("a=%d\n",a);
   return 0;
}
```



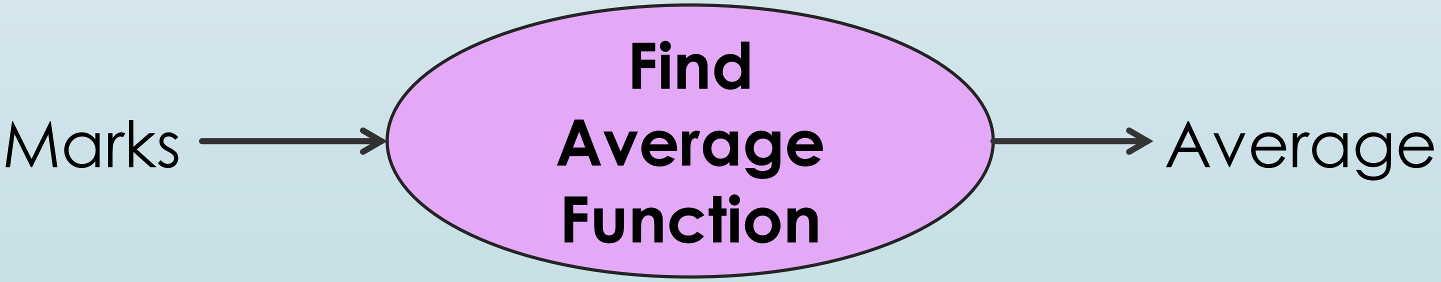
```
Outside Compound Statement
a=100

Inside Compound Statement
a=200

Outside Compound Statement
a=100
```

Library Functions in C



What is a Function?

- Function is a separately written set of codes to perform a particular task.
- They accept input, and return a single value.
- Functions can be in-built or user-defined.



Marks

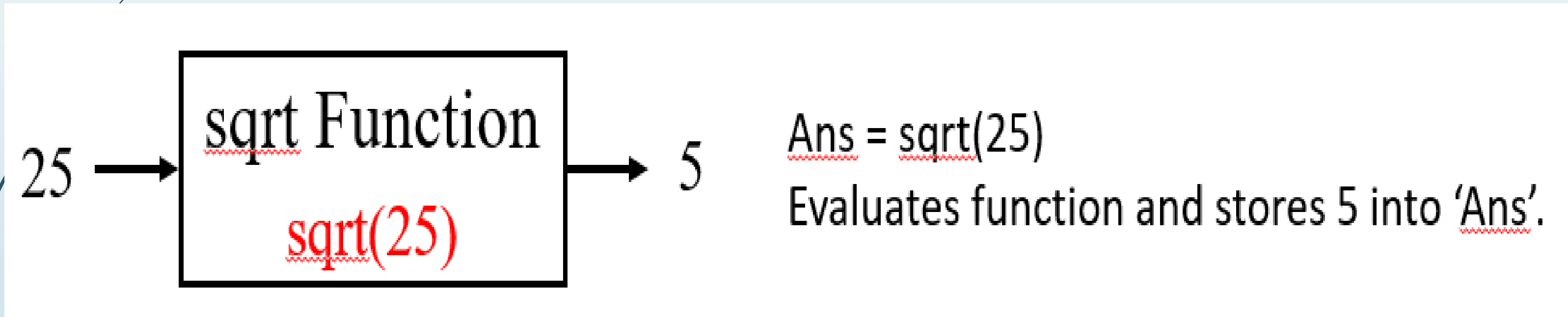
Find Average Function

Average

Library Functions

- Library functions are inbuilt functions which are grouped together and placed in a common place called “library”.
- Each library function in C performs specific operation.
- All C standard library functions are declared in many header files which are saved as file_name.h
- We are including these header files in our C program using “#include<file_name.h>” command.

- The C Preprocessor will process these header files and allow to use the functions of them at the compilation.



Ans = sqrt(25)

Evaluates function and stores 5 into ‘Ans’.

Commonly used Library functions available in C

Header file	Description	Example Functions

stdio.h	Standard input/output functions are declared	printf(), scanf()
		putchar(), getchar()

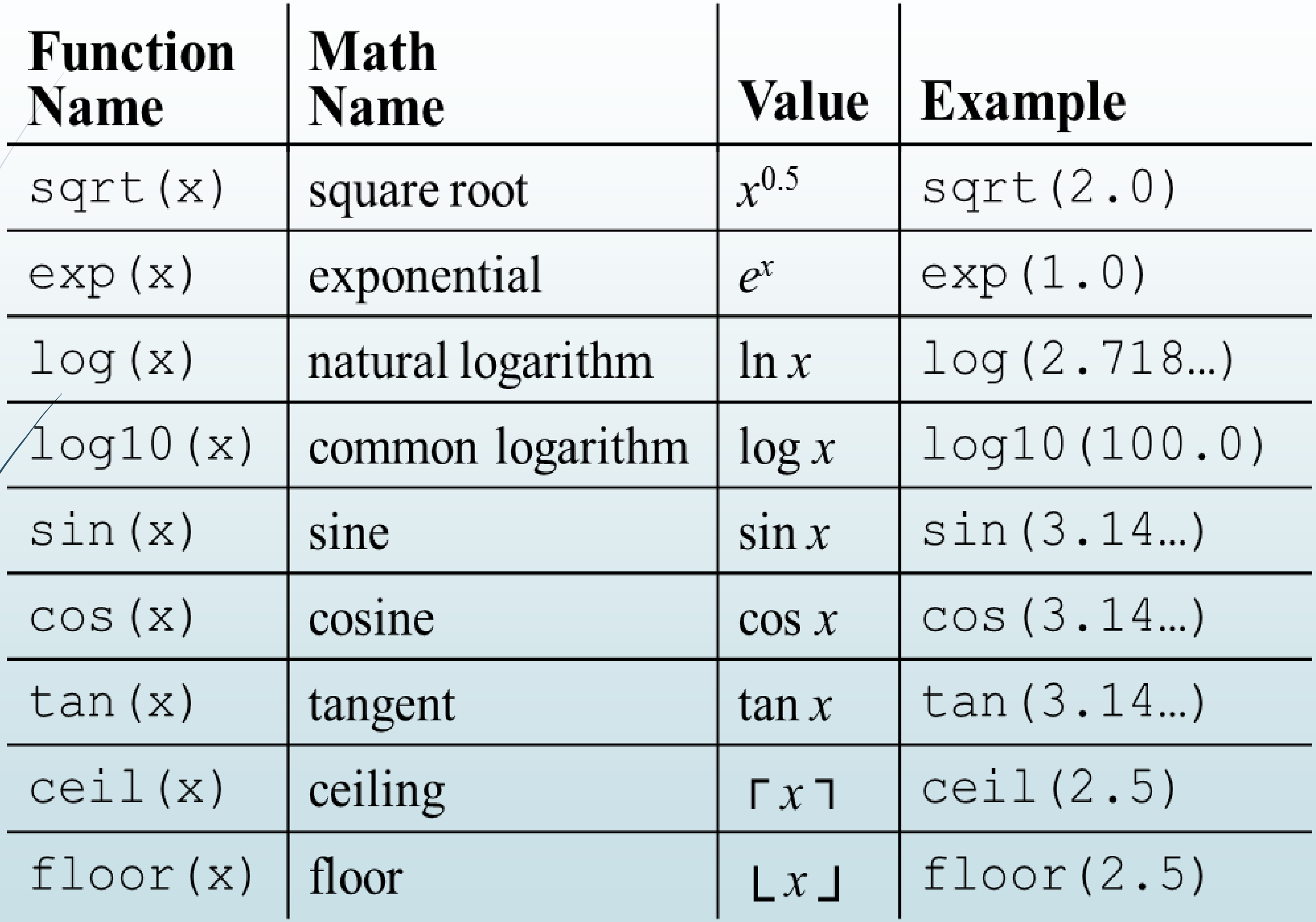
string.h	All string related functions	strcat(), strcmp(), strcpy()
stdlib.h	General functions used in C programs	malloc(), rand(), delay(), abs()
math.h	All maths related functions	cos(), sin(), sqrt(), pow()
time.h	Time and clock related functions	time(), difftime(), clock()

Mathematical Functions in C
math.h Library

Function	Description
round(double x)	Use to round up the value
sin(double x)	Use to calculate sine value
cos(double x)	Use to calculate cosine value
exp(double x)	Use to calculate the exponential “e” to the x^{th} power
log(double x)	Use to calculate the natural logarithm
log10(double x)	Use to calculate base 10 logarithm
sqrt(double x)	Find square root of the argument passed
pow(double x, double y)	Find the power of the given number (x^y)

Note: you need to include the header file math.h

```
#include<math.h>
```

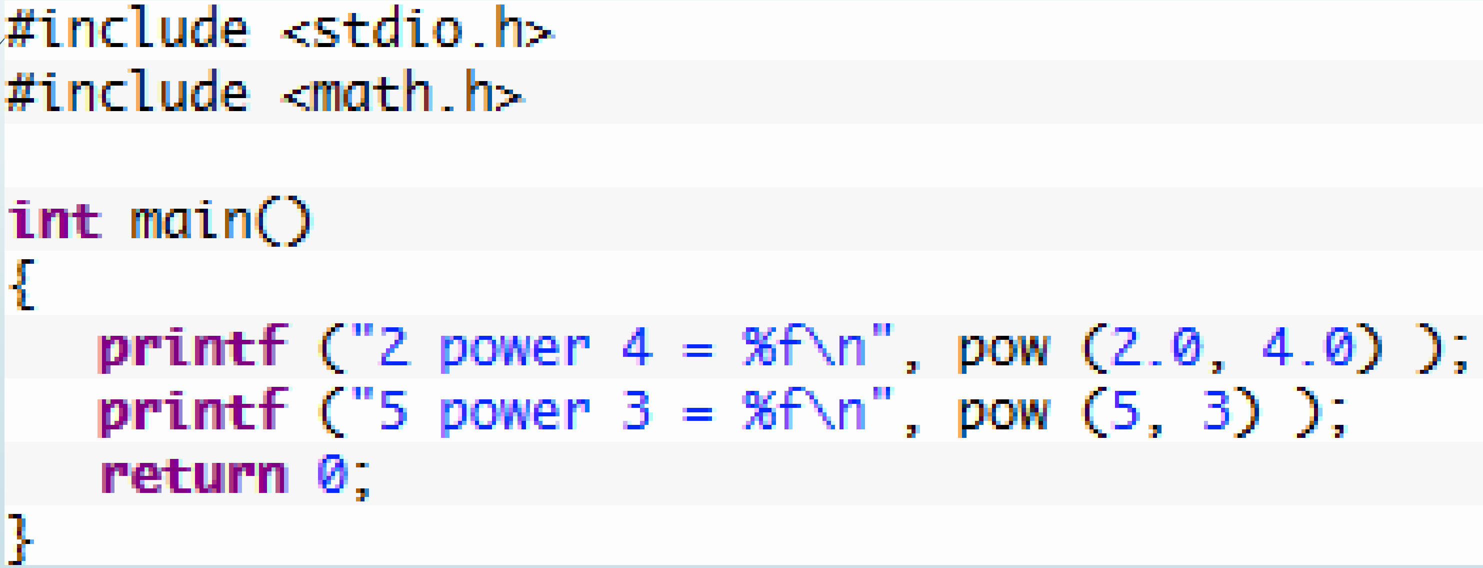


Function Name	Math Name	Value	Example
`sqrt(x)`	square root	$x^{0.5}$	`sqrt(2.0)`
`exp(x)`	exponential	e^x	`exp(1.0)`
`log(x)`	natural logarithm	$\ln x$	`log(2.718...)`
`log10(x)`	common logarithm	$\log x$	`log10(100.0)`
`sin(x)`	sine	$\sin x$	`sin(3.14...)`
`cos(x)`	cosine	$\cos x$	`cos(3.14...)`
`tan(x)`	tangent	$\tan x$	`tan(3.14...)`
`ceil(x)`	ceiling	$\lceil x \rceil$	`ceil(2.5)`
`floor(x)`	floor	$\lfloor x \rfloor$	`floor(2.5)`

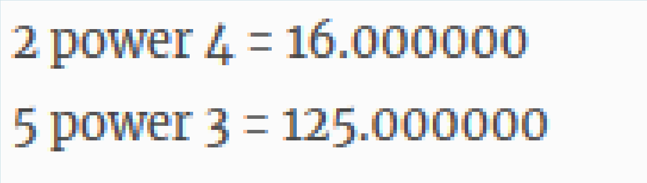
Example

Syntax:

pow(double base, double exponent);



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

int main()
{
    printf ("2 power 4 = %f\n", pow (2.0, 4.0) );
    printf ("5 power 3 = %f\n", pow (5, 3) );
    return 0;
}
```



```
2 power 4 = 16.000000
5 power 3 = 125.000000
```

Any
Question

THANK YOU… !