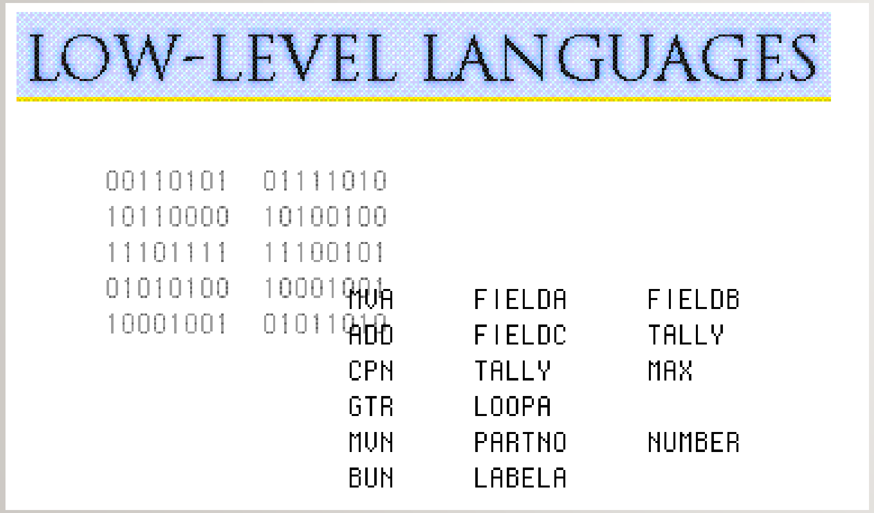
AMIT LEARNING

SESSION 1 HELLO C PROGRAMMING

CONTENTS

- Low Level Language
  - Machine code and Assembly language.
  - Machine understandable language.
  - Internal Machine Code dependent.
  - Fast to Run, but slow to write and understand.



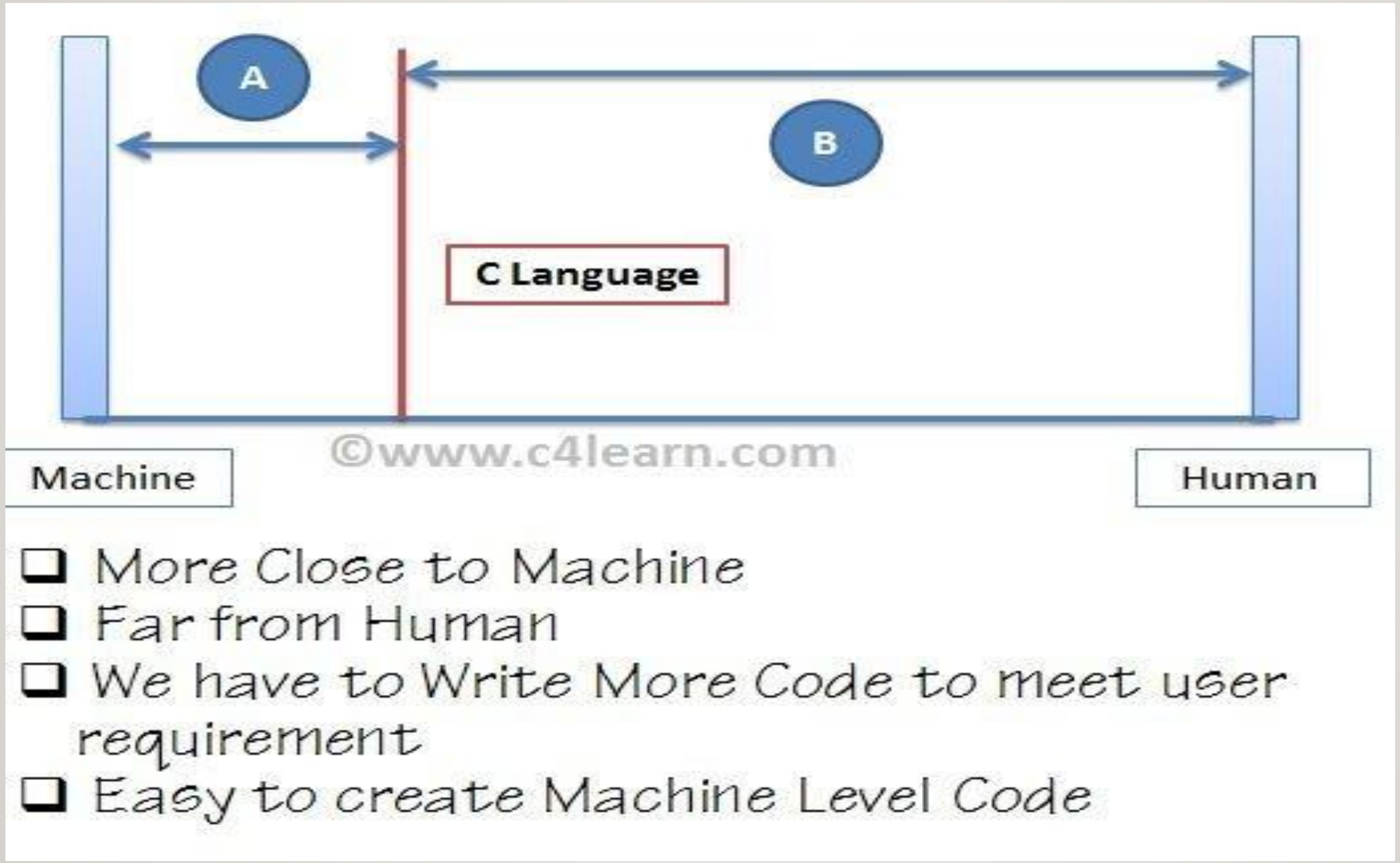
LOW-LEVEL LANGUAGES

```
00110101   01111010
10110000   10100100
11101111   11100101
01010100   10001001
10001001   01011010
```

```
MVA     FIELDA    FIELDB
ADD     FIELDC    TALLY
CPN     TALLY     MAX
GTR     LOOPA
MVN     PARTNO    NUMBER
BUN     LABELA
```

1 What is type of programming language

1.2 Middle Level Language

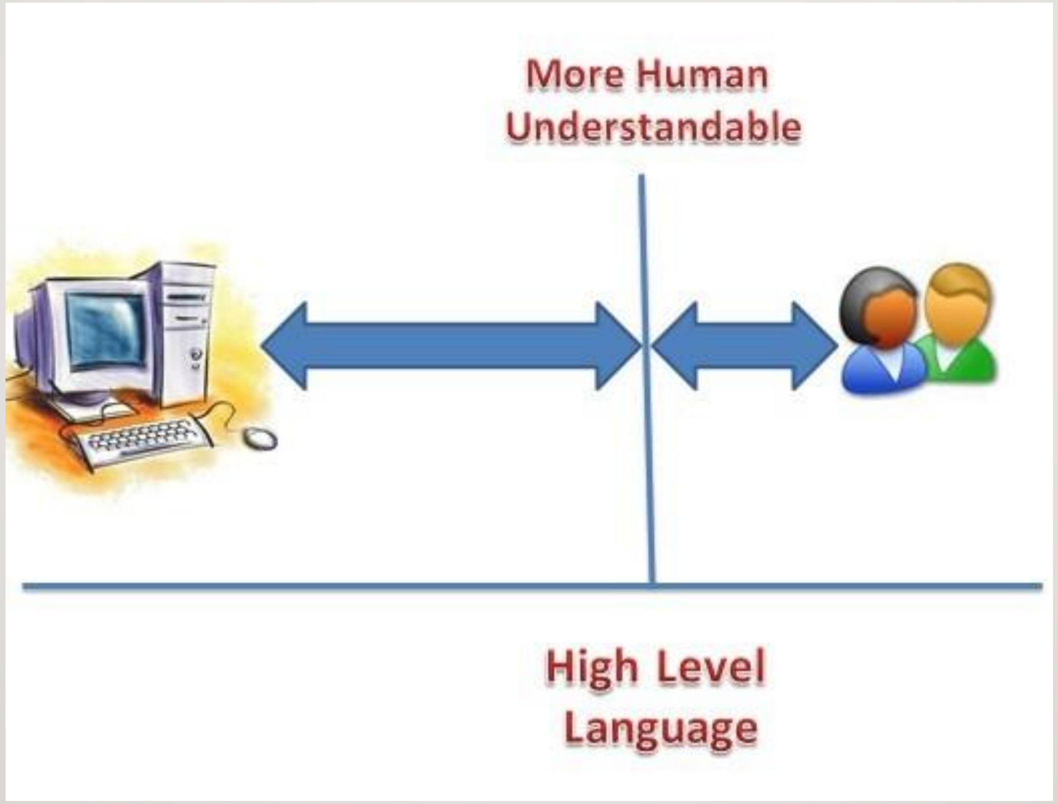


- More Close to Machine
- Far from Human
- We have to Write More Code to meet user requirement
- Easy to create Machine Level Code

Why C is Middle Level Language:

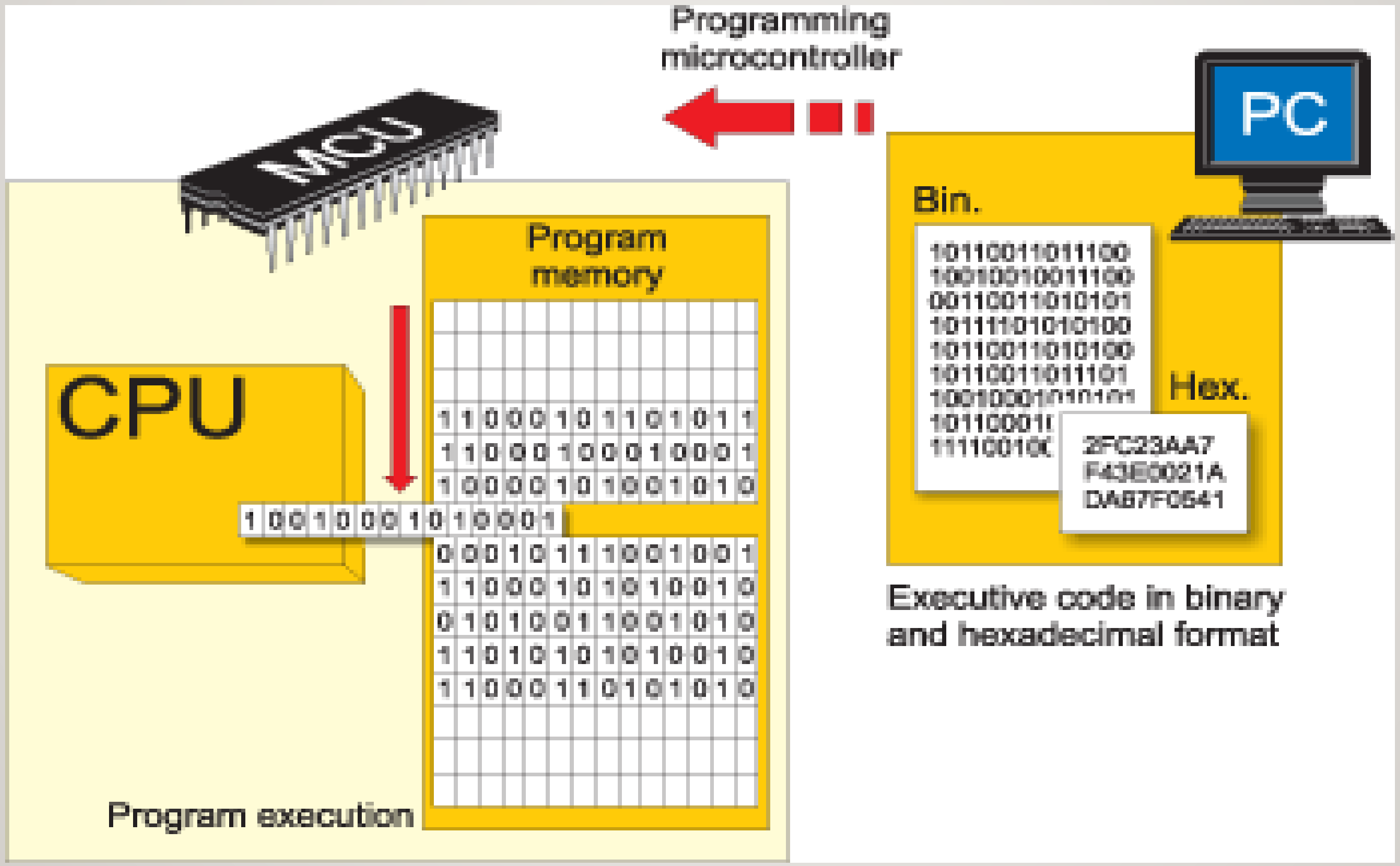
- C Programming supports Inline Assembly language programs.
- Using inline assembly feature in C, we can directly access system registers.
- C programming is used to access memory directly using pointers.
- C programming also supports high level language features.
- It is more user friendly as compare to previous languages so C is middle level language.

- High Level Language:
  - Python · Java · C++ · C# ·Visual Basic · JavaScript.

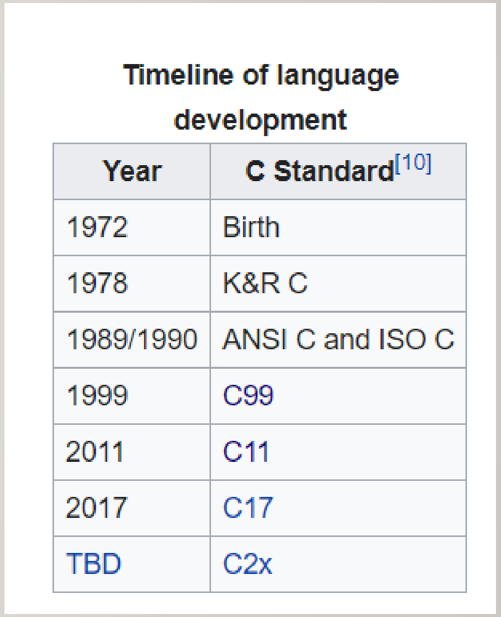


1 What is type of programming language

Embedded System


Programming microcontroller
MCU
PC
Bin.
Hex.
Program memory
CPU
Program execution
Executive code in binary and hexadecimal format
2FC23AA7
F43E0021A
DA87F0541

- These are Language Standards Supported by GCC(GNU Compiler Collection)..
- Mainly GCC supports three versions of the C standard.
- Although support for the most recent version is not yet complete.
- C language standards:
  - C89/C90 standard
    - First standardized specification for C language was developed by the American National Standards Institute in 1989.
    - C89 and C90 standards refer to the same programming language.
  - C99 standard
    - Next revision was published in 1999 that introduced new features like advanced data types and other changes.
  - C11 is the current standard for the C programming language.
    - Notable features introduced over the previous revision include improved Unicode support.Type-generic expressions using the new _Generic keyword, a cross-platform multi-threading API (threads.h) and atomic types support in both core language and the library (stdatomic.h).



Timeline of language development

Year	C Standard[10]
1972	Birth
1978	K&R C
1989/1990	ANSI C and ISO C
1999	C99
2011	C11
2017	C17
TBD	C2x

C language STANDARED

C89 VS C99

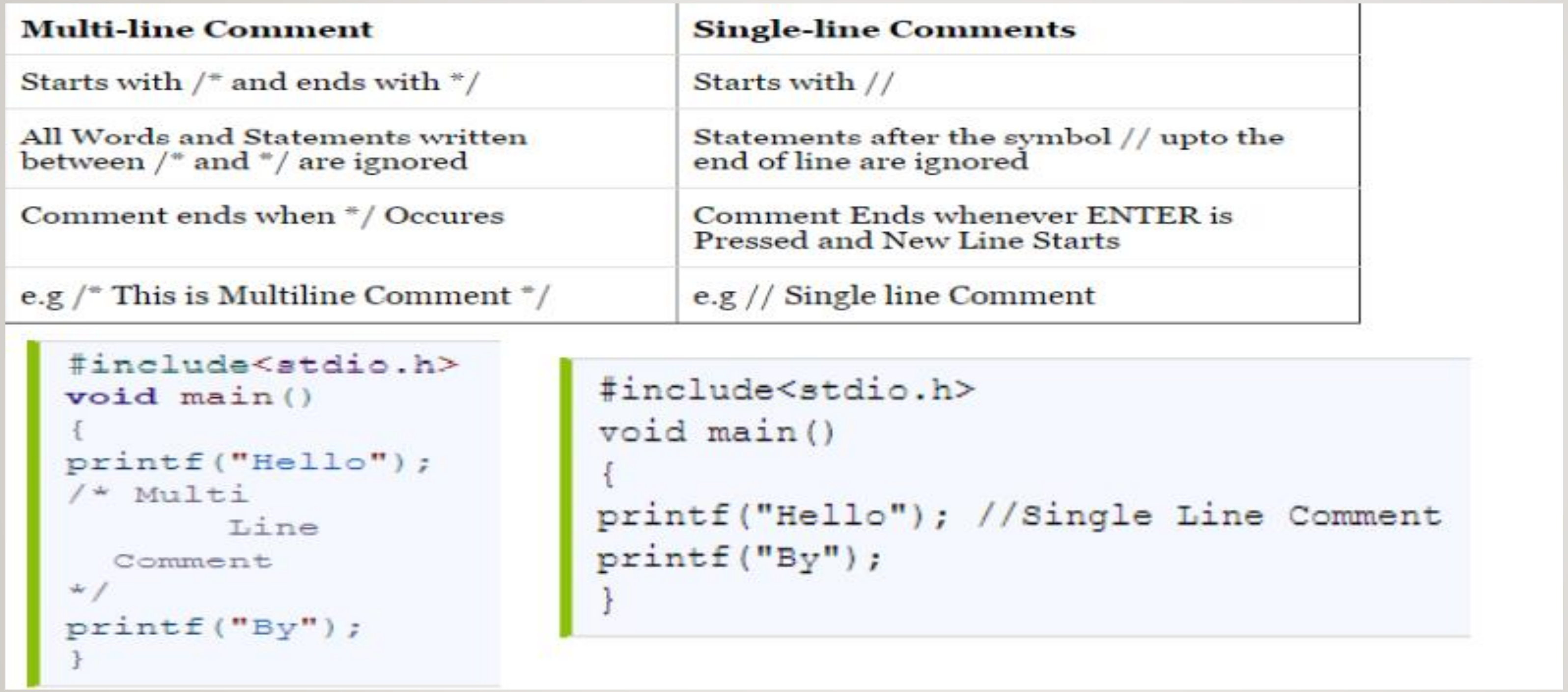
C89	C99
▪ Valid: loop variables has to be declared before the code initialization.	▪ Valid: inline , restrict, _Bool datatype, unsigned long long , pragma, ▪ Valid: we can declare loop variable at the loop initialization part ▪ Designated Initialization in Arrays: **EX**: int a[5] = {[0] = 100, [1] = 200 };

▪ Valild integer number constant inside array or constant symbol:
ex int arr[2] or #define x=2;
	▪ Valid Variable Length Arrays:
	Ex int x=1,arr[x];
	▪ Valild static,volatile , const inside array:
	Ex :int arr[static x];

▪ Valid multi line comment: /* */ ▪ Declaration of Identifiers: all Identifiers should be declared at the start of the code block.	▪ Valid Single Line Comments: // ▪ Valid Compound literals: Ex int *a = (int[]){11, 12, 13}; ▪ Designated Initialization in Arrays : EX: int a[5] = {[0] = 100, [1] = 200 }; ▪ can access member in struct{.a=1}; ▪ Declaration of Identifiers: we can declare identifiers whenever we need in a code.

Comment in C

Single line vs. Multi line



Multi-line Comment	Single-line Comments
Starts with /* and ends with */	Starts with //
All Words and Statements written between /* and */ are ignored	Statements after the symbol // upto the end of line are ignored
Comment ends when */ Occures	Comment Ends whenever ENTER is Pressed and New Line Starts
e.g /* This is Multiline Comment */	e.g // Single line Comment

```
#include<stdio.h>
void main()
{
printf("Hello");
/* Multi
        Line
   Comment
*/
printf("By");
}
```

```
#include<stdio.h>
void main()
{
printf("Hello"); //Single Line Comment
printf("By");
}
```

what Numeral system

Common Numeral System

System	Base	Symbols	Human Readable ?	Used in computers?
Decimal	10	0, 1, … 9	✓	✗
Binary	2	0, 1	✗	✓
Octal	8	0, 1, … 7	✗	✗
Hexa-decimal	16	0, 1, … 9, A, B, … F	✗	✓

What Numeral System

Common Numeral system

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7

what Numeral system

Common Numeral system

Decimal	Binary	Octal	Hexadecimal
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

what Numeral system

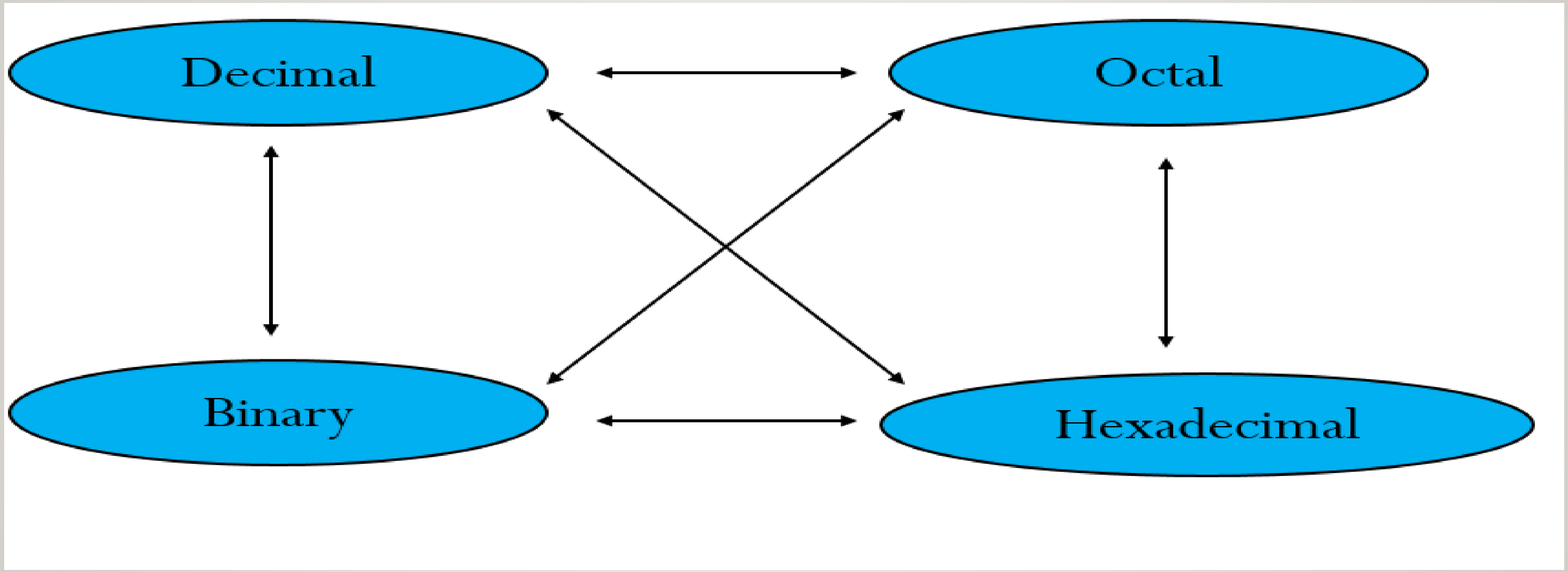
Common Numeral system

Decimal	Binary	Octal	Hexadecimal
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
21	10101	25	15
22	10110	26	16
23	10111	27	17

what Numeral system

Conversion among Bases

- The possibilities:



what Numeral system

Conversion among Bases

- Quick Example:

25d = 11001b = 31O = 19h

- Binary to Decimal Conversion:
  - Technique:
    - Multiply each bit by 2n, where n is the "weight" of the bit
    - The weight is the position of the bit, starting from 0 on the right
    - Add the results
- Octal to Decimal Conversion:
- Technique :
  - Multiply each bit by 8n, where n is the "weight" of the bit.
  - The weight is the position of the bit, starting from 0 on the right.
  - Add the results.
- Hexadecimal to Decimal Conversion:
- Technique :
  - Multiply each bit by 16n, where n is the "weight" of the bit.
  - The weight is the position of the bit, starting from 0 on the right.
  - Add the results.

what Numeral system

Conversion among Bases

- Binary to Hexadecimal Conversion:
  - Technique :
    - Group bits in fours, starting on right.
    - Convert to hexadecimal digits.

- Hexadecimal to Octal:
  - Technique :
    - Use binary as an intermediary.

C keywords and identifier

What is Keyword in C Programming.

- Keywords are the reserved words used in programming.

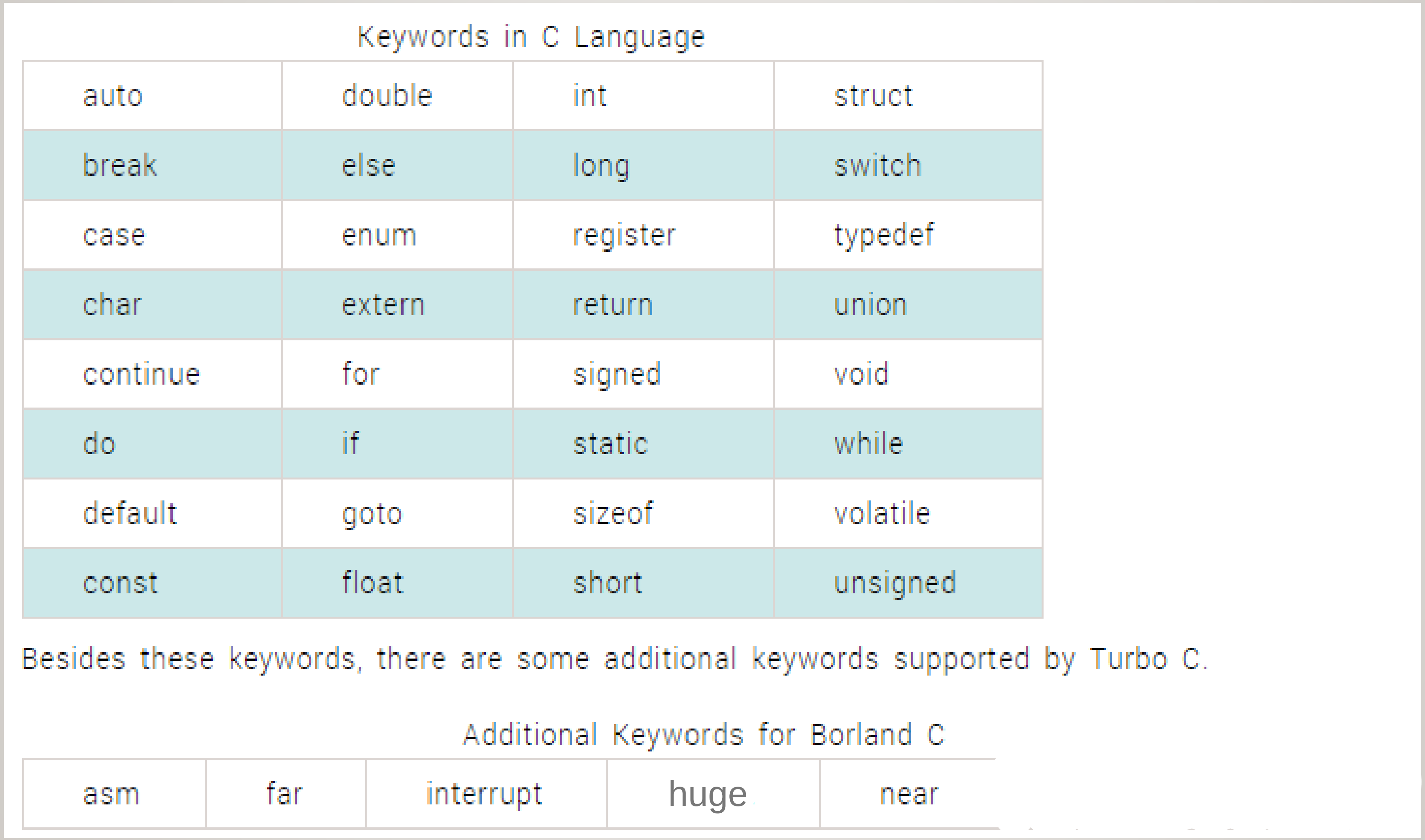
- Each keywords has fixed meaning and that cannot be changed by user.

- C programming is case sensitive, all keywords must be written in lowercase.

- A keyword name can not be used as a variable name.

C keywords and identifier

What is Keyword in C Programming.



Keywords in C Language

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile
const	float	short	unsigned

Besides these keywords, there are some additional keywords supported by Turbo C.

Additional Keywords for Borland C

asm	far	interrupt	huge	near

C keywords and identifier

Rules for constructing identifier in C.

- Identifiers :
  - Identifiers are names given to C entities, such as variables, functions, structures.
- Rules for writing identifier :
  - Composed of letters (both uppercase and lowercase letters), digits and underscore '_' only.
  - The first letter of identifier should be either a letter or underscore.
  - There is no rule for the length of an identifier.
  - Keywords cannot be used as identifiers.
  - Identifiers are case sensitive.
  - The first 31 letters of two identifiers in a program should be different.
  - No special characters, such as a semicolon, period, whitespaces, slash, or comma are permitted to be used in or as an Identifier.
  - Special Characters not valid in identifiers:

, < > . _ ( ) ; ? : % [ ] #

' & { } " ^ ! * / | - \ ~ +

C keywords and identifier

Rules for constructing identifier in C.

- Examples For Identifier Construction:
- Use allowed Characters
  - Valid Names Examples:
    - num
    - Num
    - Num1
    - _NUM
    - NUM_temp2
- Blanks are not allowed
  - Invalid Names Examples:
    - number 1
    - num 1
- No special symbols other that underscore
  - Valid Identifier Example:s
    - num_1
    - number_of_values
    - status_flag
- Invalid Identifier Example:
  - num@1

C keywords and identifier

Rules for constructing identifier in C.

- First Character must be underscore or Alphabet
  - Valid Identifier Examples:
    - _num1
    - Num
    - Num_
    - _
    - __
  - Invalid Identifier Examples:
    - 1num
    - 1_num
    - 365_days

C keywords and identifier

Rules for constructing identifier in C.

- However if we capitalize any Letter from Reserve word then it will become legal variable name.
  - Valid Identifier Examples:
    - Int.
    - Char.
    - Continue.
    - CONTINUE.
  - Invalid Identifier Examples:
    - int
    - char
    - continue

C PROGRAMMING DATA TYPE

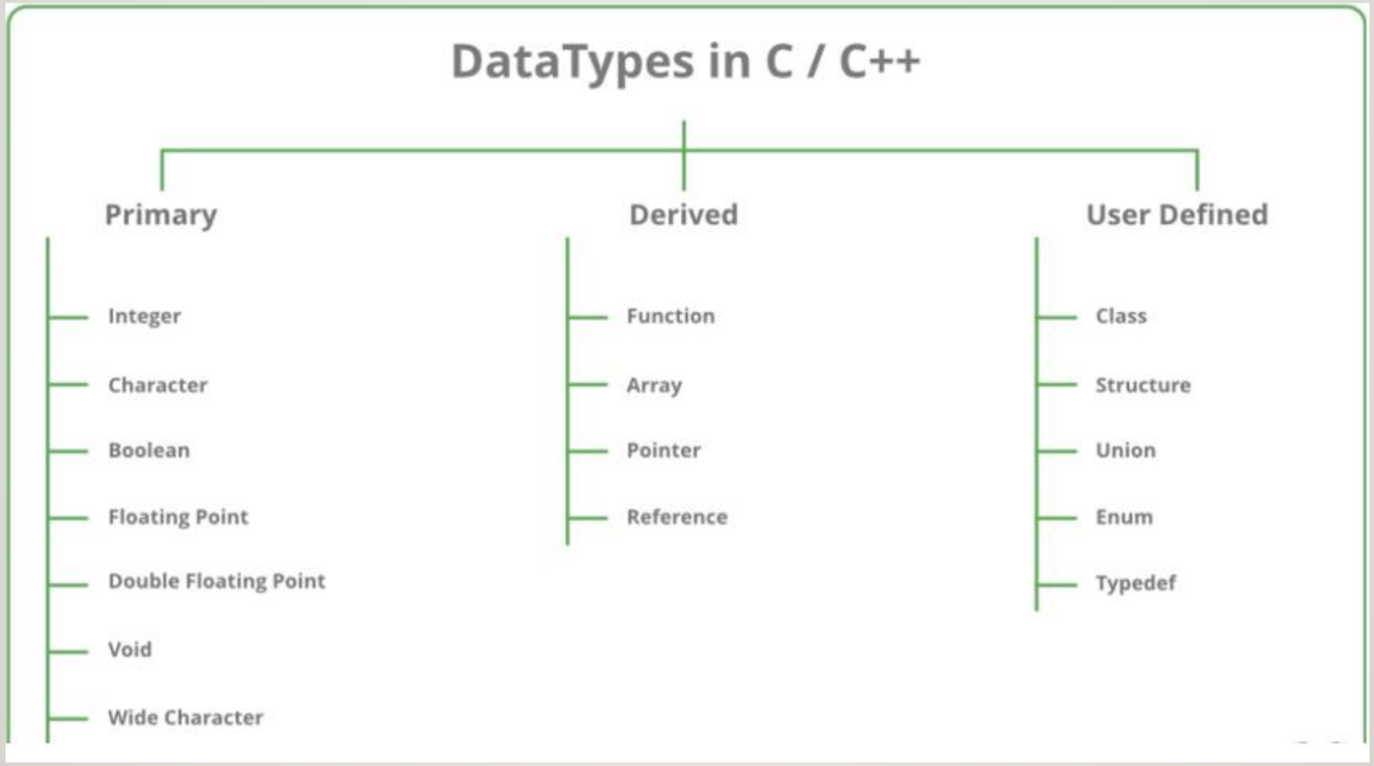
What is Data type.

- A data type is used to
  - Identify the type of a variable when the variable is defined.
  - Identify the type of the return value of a function.
  - Identify the type of a parameter expected by a function.

C PROGRAMMING DATA TYPE

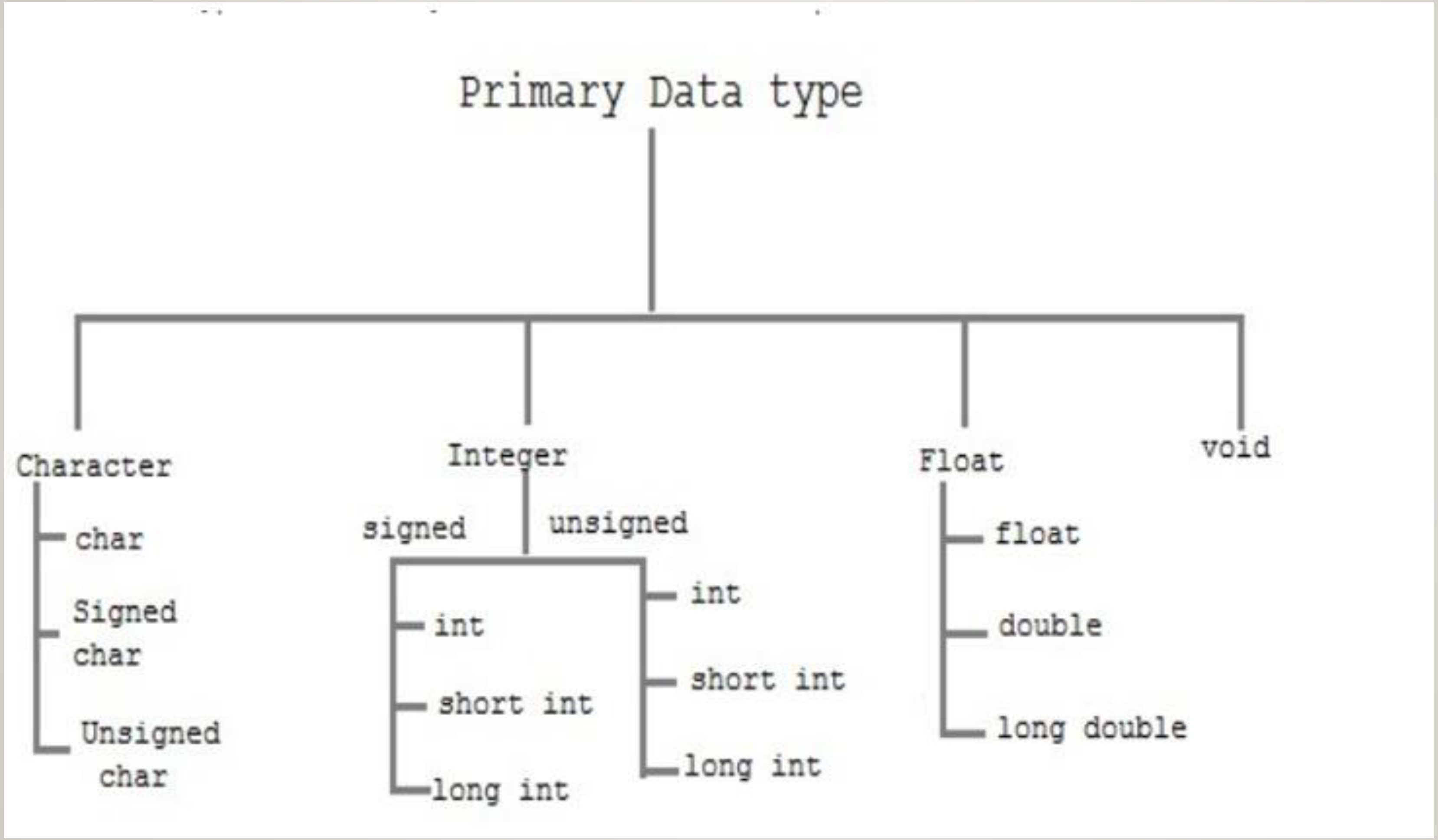
What is Data type.

- Data types are the keywords, which are used for assigning a type to a variable.



C PROGRAMMING DATA TYPE

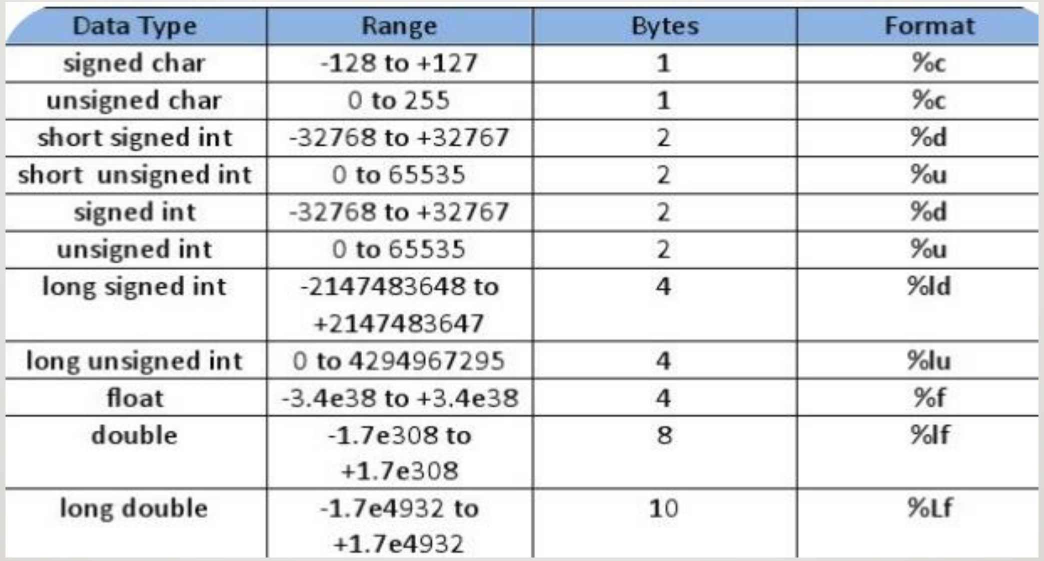
What is Data type.



C PROGRAMMING DATA TYPE

What is Data type.

- **We can use the sizeof() operator to check the size of a variable.**
- Can calculate range of data type:
  - Signed data type: $-2^{(n-1)}$ to $+2^{(n-1)} - 1$.
  - Un Signed data type: 0 to $+2^{(n)} - 1$.



Data Type	Range	Bytes	Format
signed char	-128 to +127	1	%c
unsigned char	0 to 255	1	%c
short signed int	-32768 to +32767	2	%d
short unsigned int	0 to 65535	2	%u
signed int	-32768 to +32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	-2147483648 to +2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to +3.4e38	4	%f
double	-1.7e308 to +1.7e308	8	%lf
long double	-1.7e4932 to +1.7e4932	10	%Lf

C PROGRAMMING DATA TYPE

What is Data type.

- Declaration of a variable:

```
int var_1;
```

- Initialization of variable:

```
var_1=0;
```

Initialization + declaration : data type _variable _name=initial value; int var_1=0;

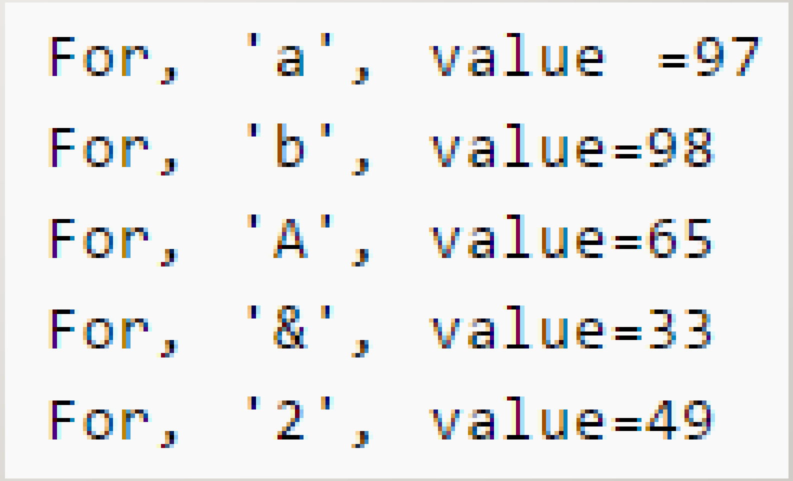
- Integer data types :The size of int is either 2 bytes(In older PC's) or 4 bytes depend in compiler and compiler take size of data type from architecture bus.
- Signed vs unsigned :Signed and Unsigned variables tell us whether it can take positive values, negative values or both.
- **signed** keyword is used for those variables which can take positive as well as negative values.
- **unsigned** keyword is used for those variables which can take only values which are zero or positive i.e., without - (negative sign).
- We can use signed and unsigned keywords with only 'int' and 'char' data types.
- It is not necessary to give signed keyword to any variable because all the variables are signed by default.

C PROGRAMMING DATA TYPE

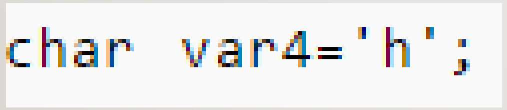
What is Data type.

- Difference between float and double:
  - Size of float(Single precision float data type) is 4 bytes
  - double(Double precision float data type) is 8 bytes
  - Floating point variables has a precision of 6 digits whereas the precision of double is 14 digits.
  - Note: Precision describes the number of significant decimal places that a floating values carries.

- Character types:
  - The size of char is 1 byte..
  - The character data type consists of ASCII characters & it can take integer number but with respect to his size.
  - Each Character is given a specific value.
  - When a become 128 through a++, the range is exceeded and as a result the first number from negative side of the range (i.e. -128) gets assigned to variable.



```
For, 'a', value =97
For, 'b', value=98
For, 'A', value=65
For, '&', value=33
For, '2', value=49
```



```
char var4='h';
```

C PROGRAMMING DATA TYPE

Signed vs. unsigned Number

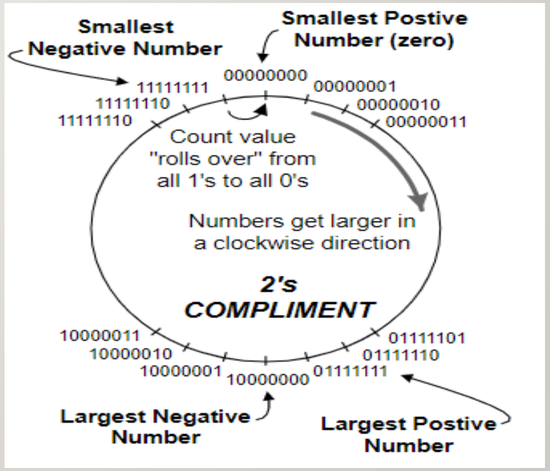
➢ Signed vs. unsigned Number:

- Signed and Unsigned variables tell us whether it can take positive values, negative values or both. **signed** keyword is used for those variables which can take positive as well as negative values. **unsigned** keyword is used for those variables which can take only values which are zero or positive i.e., without **-** (negative sign).

- We can use signed and unsigned keywords with only int and char data types.

- When the range is exceeded and as a result the first number from negative side of the range incase signed number.
- When the range is exceeded and as a result the first number from positive side of the range incase signed number.
- When the range is exceeded and as a result the first number from positive side of the range incase unsigned number .

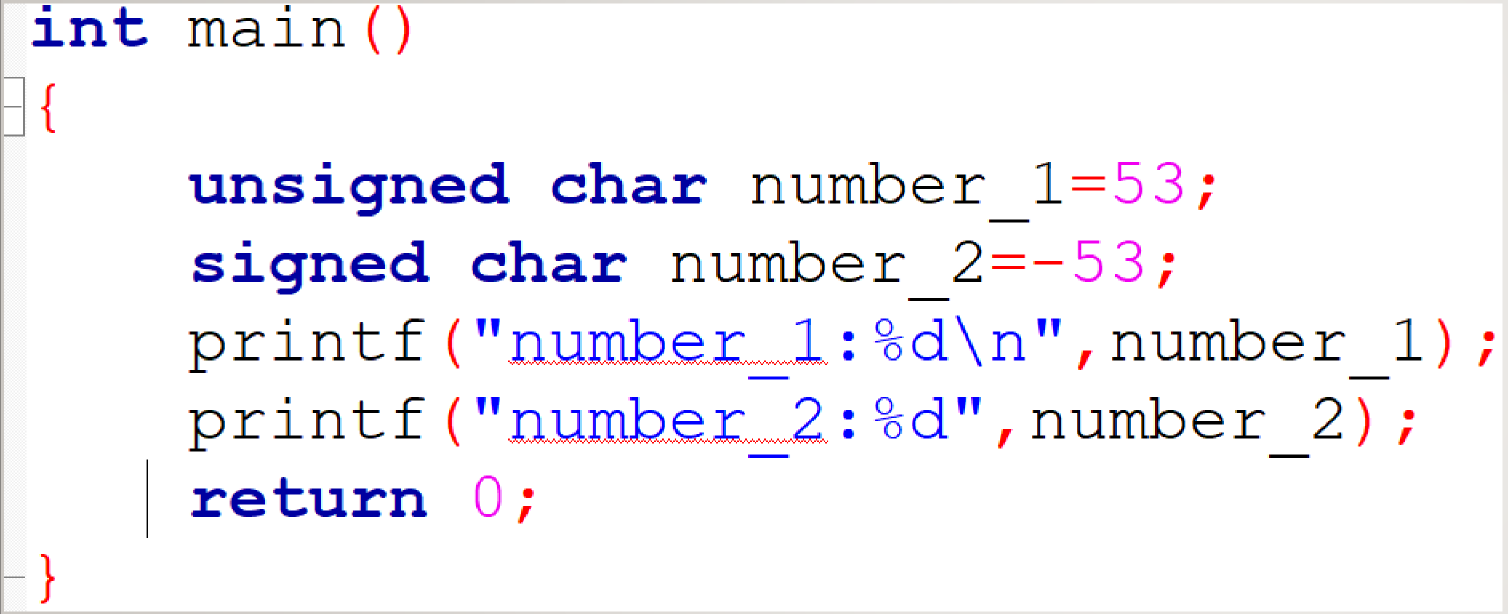
That's because the overflow happens.



C PROGRAMMING DATA TYPE

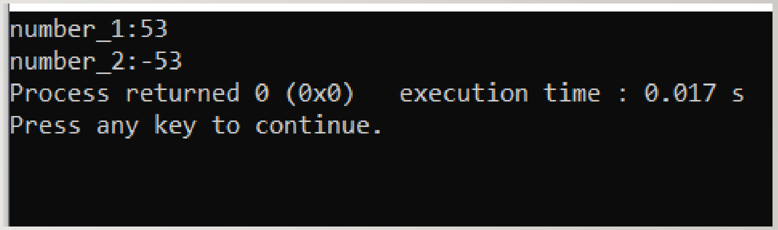
Signed vs. unsigned Number

CODE



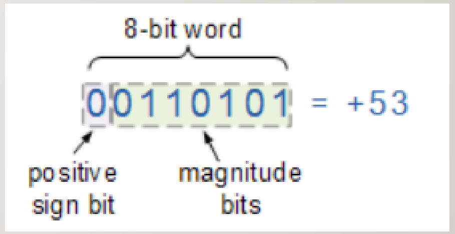
```
int main()
{
    unsigned char number_1=53;
    signed char number_2=-53;
    printf("number_1:%d\n",number_1);
    printf("number_2:%d",number_2);
    return 0;
}
```

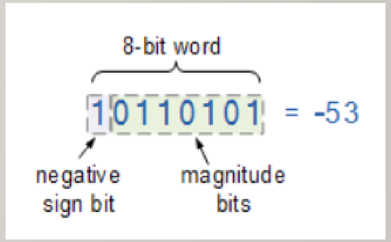
OUTPUT

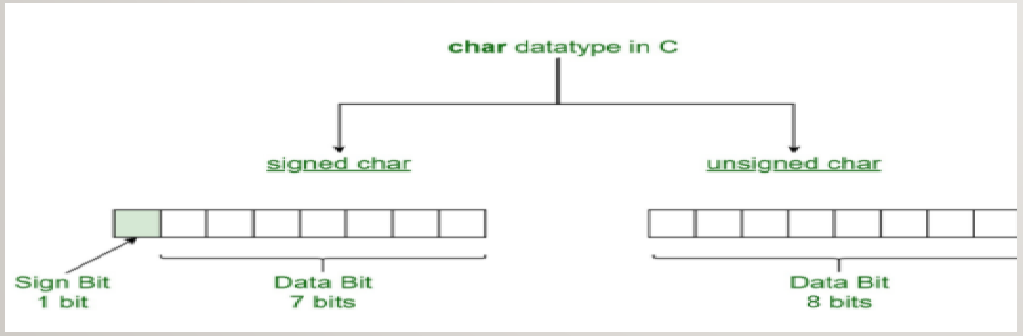


```
number_1:53
number_2:-53
Process returned 0 (0x0)   execution time : 0.017 s
Press any key to continue.
```

DESCRIPTION



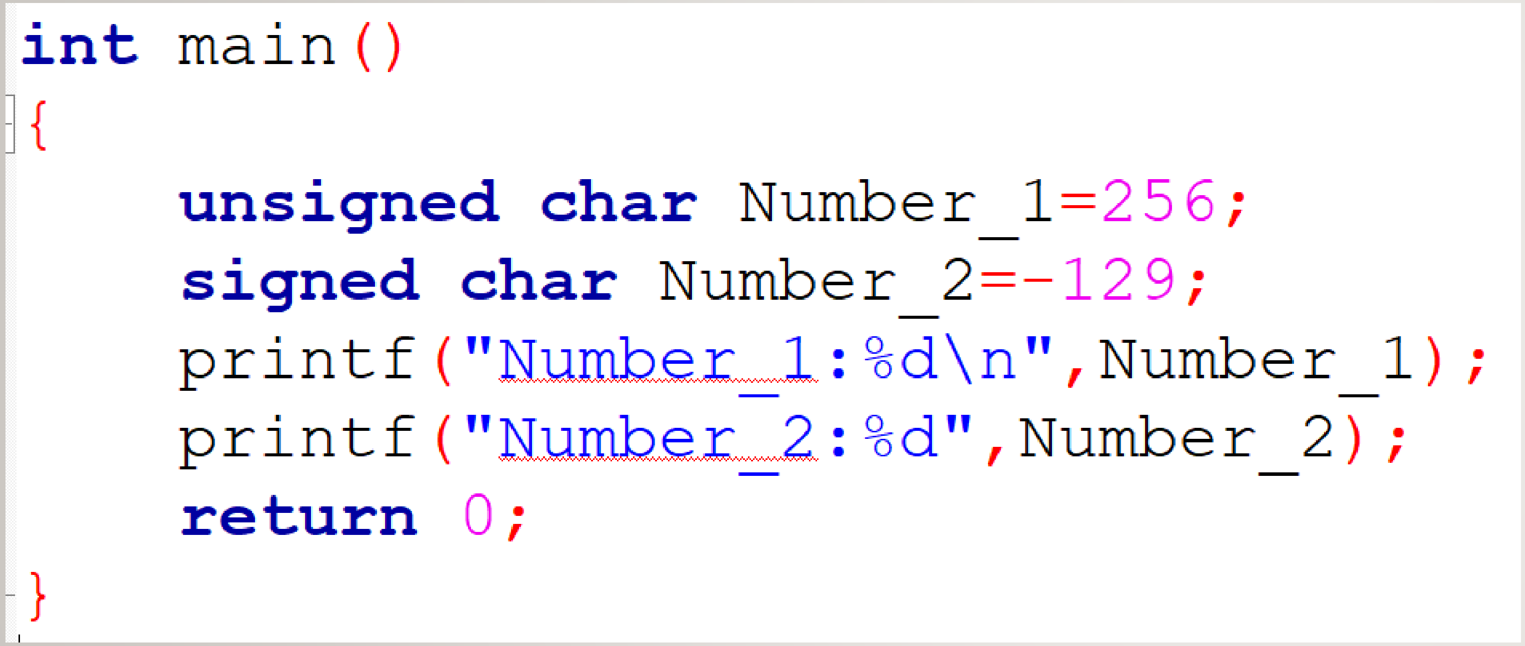




C PROGRAMMING DATA TYPE

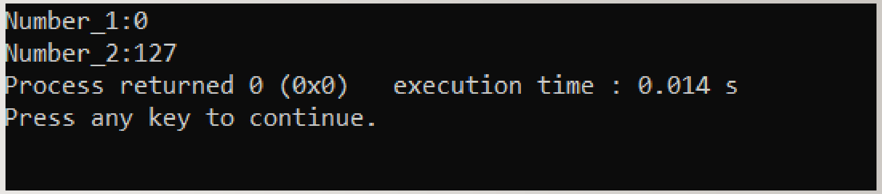
Signed vs. unsigned Number

CODE



```
int main()
{
    unsigned char Number_1=256;
    signed char Number_2=-129;
    printf("Number_1:%d\n",Number_1);
    printf("Number_2:%d",Number_2);
    return 0;
}
```

OUTPUT



```
Number_1:0
Number_2:127
Process returned 0 (0x0)   execution time : 0.014 s
Press any key to continue.
```

DESCRIPTION

- In this example we create two variable

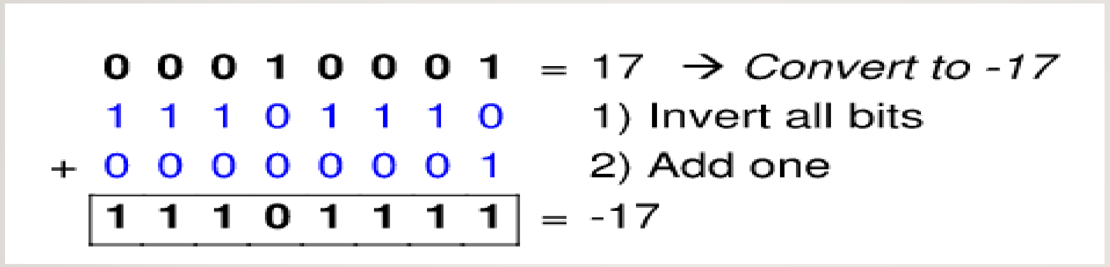
✓ Number_1,Number_2:

Exceed the limit so overflow happened .

C PROGRAMMING DATA TYPE

Negative Number Representation.

- Represent negative number:



```
   0 0 0 1 0 0 0 1  = 17  → Convert to -17
   1 1 1 0 1 1 1 0    1) Invert all bits
 + 0 0 0 0 0 0 0 1    2) Add one
   1 1 1 0 1 1 1 1  = -17
```

Type Casting in C Programming .

Type Casting in C Programming .

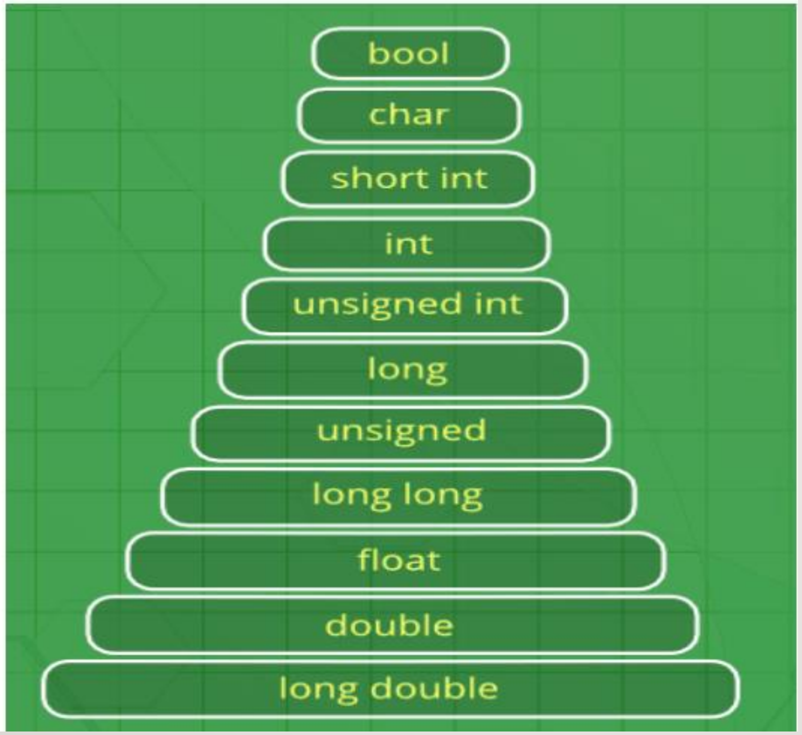
- Definition:
  - Type Casting is converting a data type to another data type, it's also called as type conversion.
  - The aim of type casting is converting data types without losing its original meaning, and to allow operands with different signes to co-exist in the same expression.
- There are two types of type casting:
  - Implicit type casting .
  - Explicit type casting.
- Advantages of Type Conversion:
  - ❑This is done to take advantage of certain features of type hierarchies or type representations.
  - ❑It helps us to compute expressions containing variables of different data types.

- <u>implicit type casting:</u>
  - 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 loss of data.
  - All the data types of the variables are upgraded to the data type of the variable with largest data type.
  - 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).

Type Casting in C Programming .

Implicit type casting

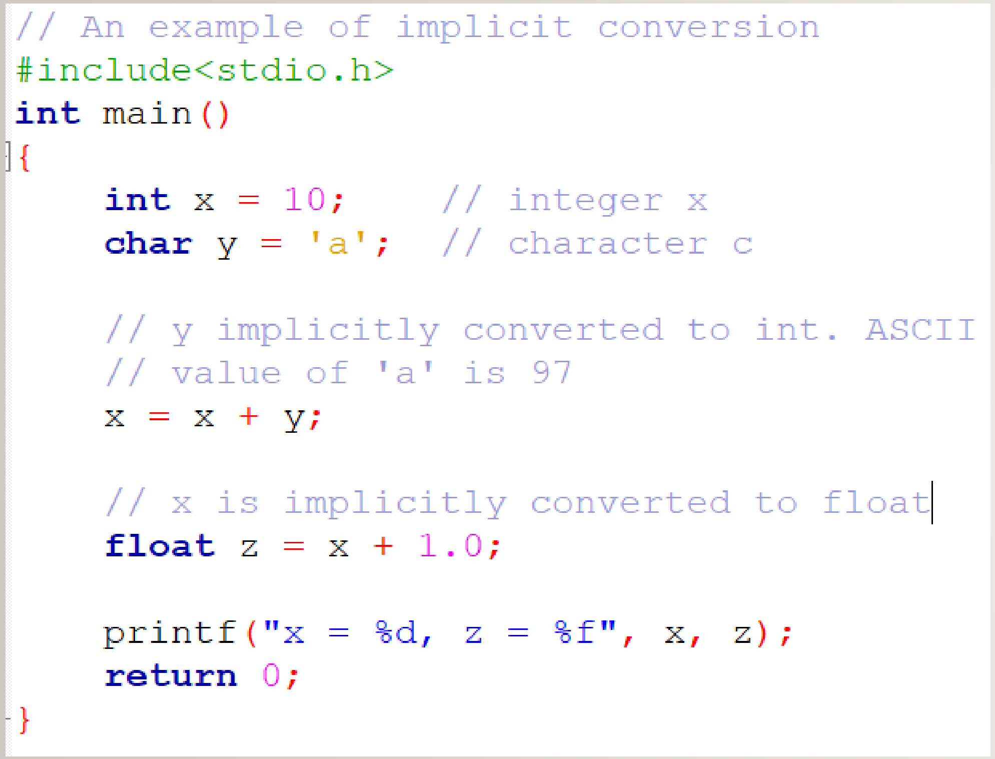
Types



Type Casting in C Programming .

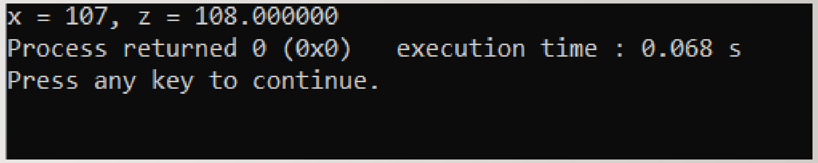
Implicit type casting

CODE



```c
// An example of implicit conversion
#include<stdio.h>
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;

    printf("x = %d, z = %f", x, z);
    return 0;
}
```

OUTPUT



```
x = 107, z = 108.000000
Process returned 0 (0x0)   execution time : 0.068 s
Press any key to continue.
```

DESCRIPTION

1 – variable y is automatic implicit casting to integer.

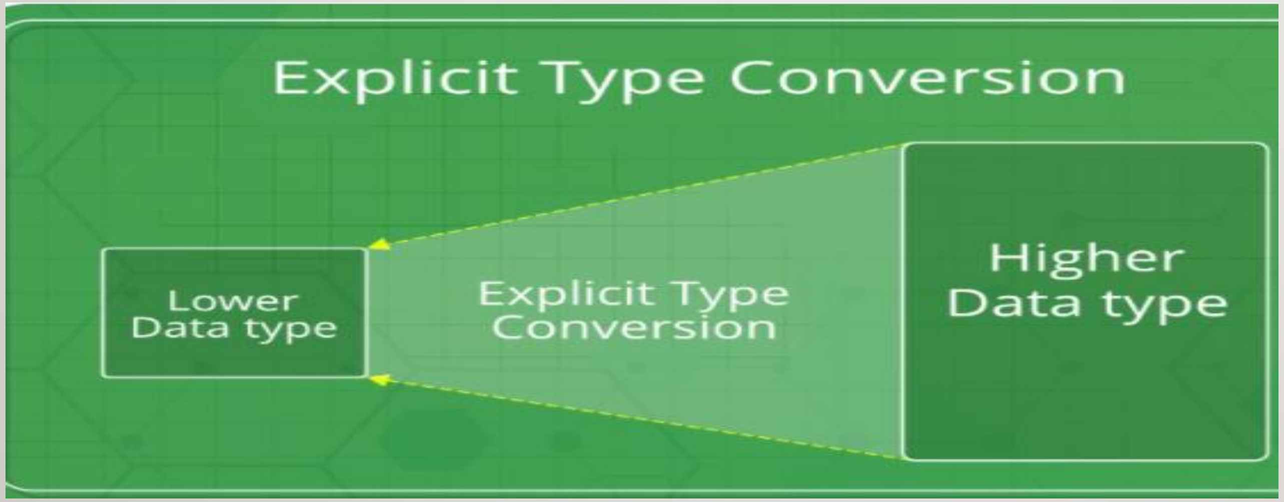
2 – variable x is automatic implicit casting to float.

Type Casting in C Programming .

Explicit type casting

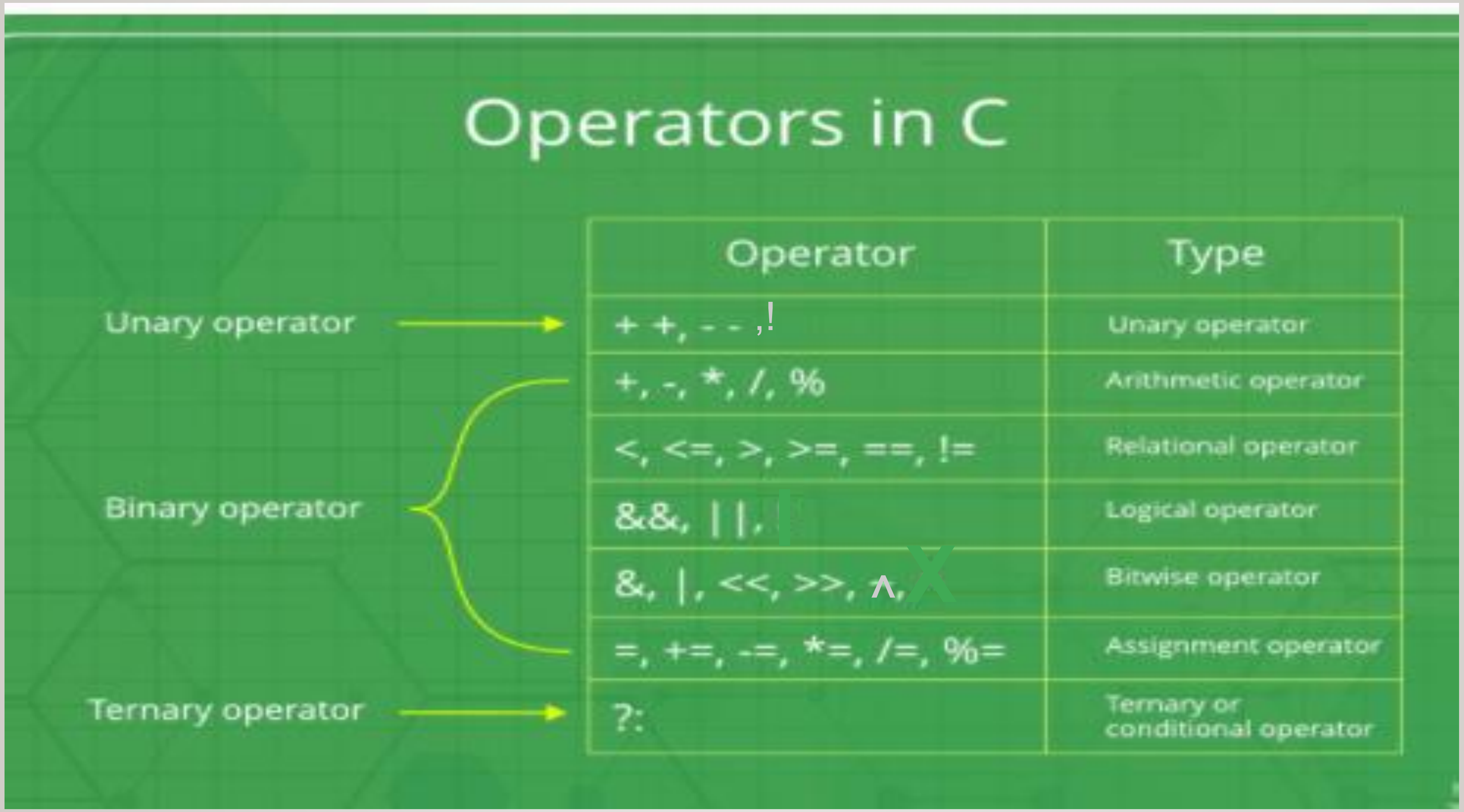
- This process is also called type casting and it is user defined. Here the user can type cast the result to make it of a particular data type.
- Type indicated the data type to which the final result is converted.
- The syntax in C:

```
(type) expression
```



C operators in c.

C operators in c.



Operators in C

	Operator	Type
Unary operator	+ +, - - ,!	Unary operator
Binary operator	+, -, *, /, %	Arithmetic operator
	<, <=, >, >=, ==, !=	Relational operator
	&&, ||,	Logical operator
	&, |, <<, >>, ^,	Bitwise operator
	=, +=, -=, *=, /=, %=	Assignment operator
Ternary operator	?:	Ternary or conditional operator

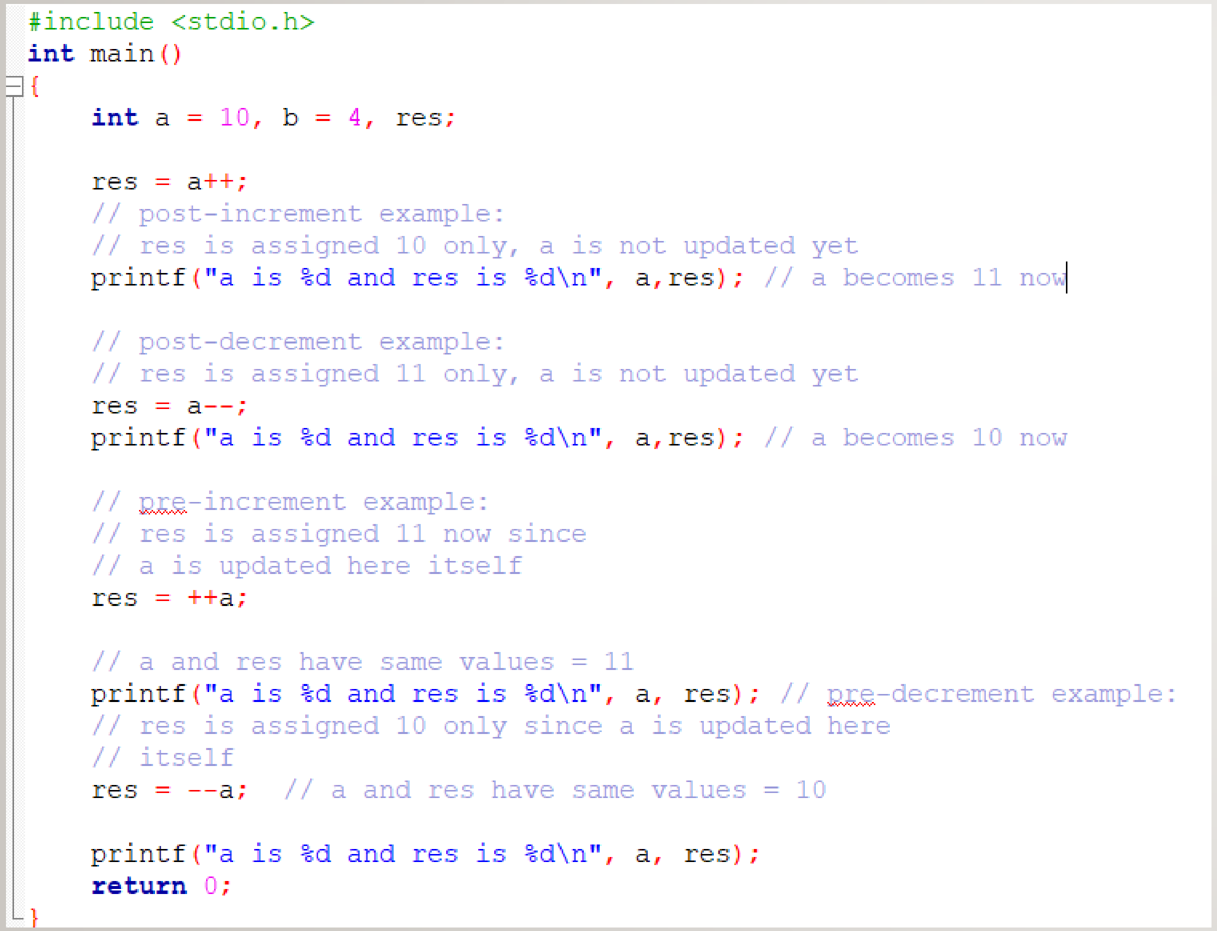
Unary Operators:

- **Operators that operates or works with a single operand are unary operators.**
- Increment and decrement operators(++,--).
- Prefix (like: ++var), ++ var will increment the value of var then return it.
- Postfix(like: var++), operator will return the value of operand first and then only increment it.

C operators in c.

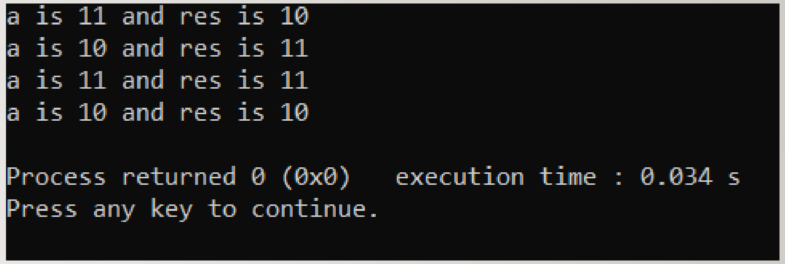
Unary Operator

CODE



```
#include <stdio.h>
int main()
{
    int a = 10, b = 4, res;

    res = a++;
    // post-increment example:
    // res is assigned 10 only, a is not updated yet
    printf("a is %d and res is %d\n", a,res); // a becomes 11 now

    // post-decrement example:
    // res is assigned 11 only, a is not updated yet
    res = a--;
    printf("a is %d and res is %d\n", a,res); // a becomes 10 now

    // pre-increment example:
    // res is assigned 11 now since
    // a is updated here itself
    res = ++a;

    // a and res have same values = 11
    printf("a is %d and res is %d\n", a, res); // pre-decrement example:
    // res is assigned 10 only since a is updated here
    // itself
    res = --a;  // a and res have same values = 10

    printf("a is %d and res is %d\n", a, res);
    return 0;
}
```

OUTPUT



```
a is 11 and res is 10
a is 10 and res is 11
a is 11 and res is 11
a is 10 and res is 10

Process returned 0 (0x0)   execution time : 0.034 s
Press any key to continue.
```

DESCRIPTION

- In this example we create variable a , b and operation happened in post-increment and pre-increment

C operators in c.

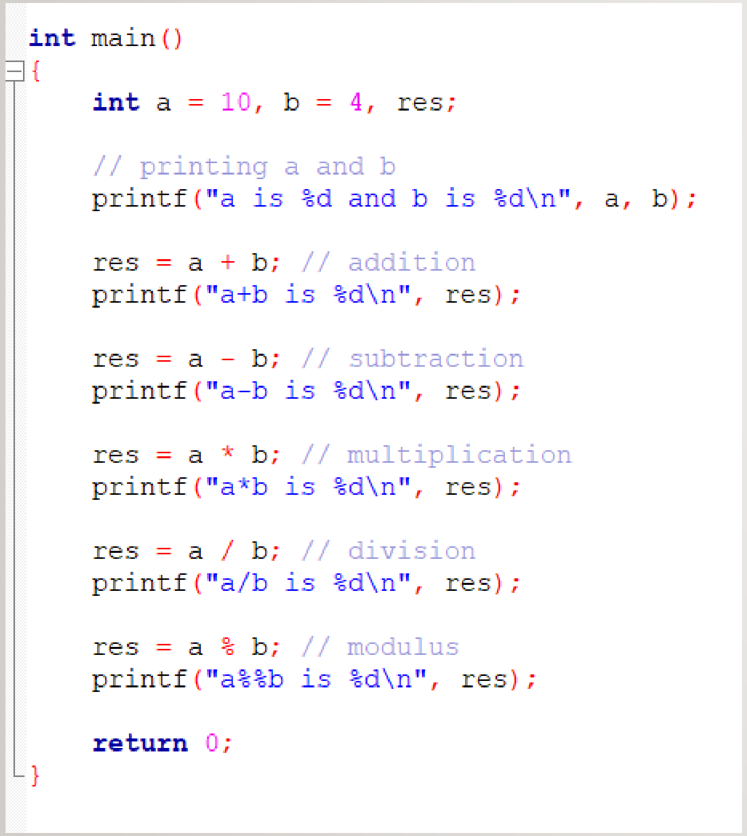
Arithmetic Operator

Operator	Meaning	Example
+	Addition Operator	10 + 20 = 30
-	Subtraction Operator	20 – 10 = 10
*	Multiplication Operator	20 * 10 = 200
/	Division Operator	20 / 10 = 2
%	Remainder (Modulo Operator)	20 % 6 = 2

C operators in c.

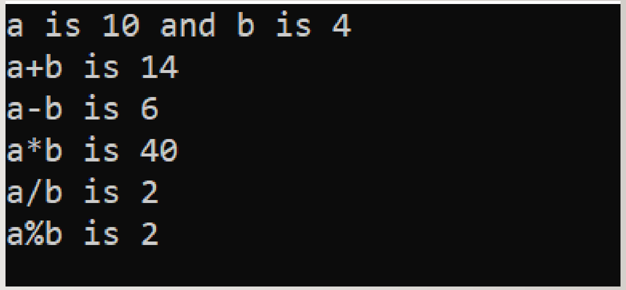
Arithmetic Operator

CODE



```c
int main()
{
    int a = 10, b = 4, res;

    // printing a and b
    printf("a is %d and b is %d\n", a, b);

    res = a + b; // addition
    printf("a+b is %d\n", res);

    res = a - b; // subtraction
    printf("a-b is %d\n", res);

    res = a * b; // multiplication
    printf("a*b is %d\n", res);

    res = a / b; // division
    printf("a/b is %d\n", res);

    res = a % b; // modulus
    printf("a%%b is %d\n", res);

    return 0;
}
```

OUTPUT



```
a is 10 and b is 4
a+b is 14
a-b is 6
a*b is 40
a/b is 2
a%b is 2
```

DESCRIPTION

- In this example we create variable a , b and operation happened in Arithmetic operator.

C operators in c.

Assignment Operators

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A

C operators in c.

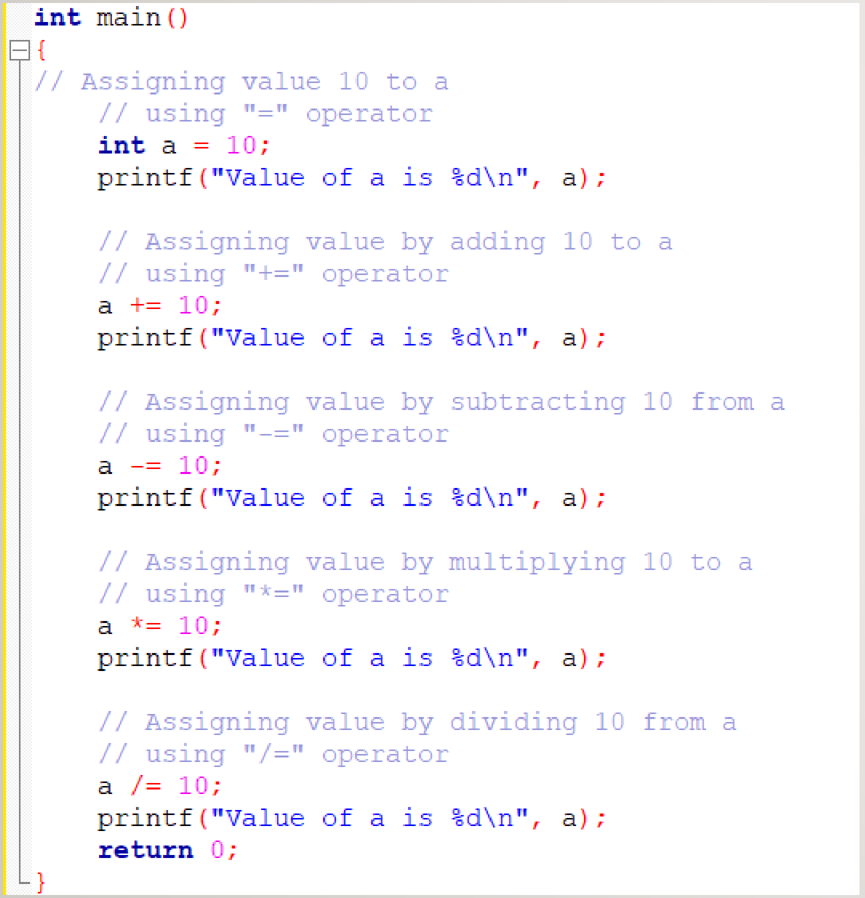
Assignment Operators

Operator	Description	Example
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

C operators in c.

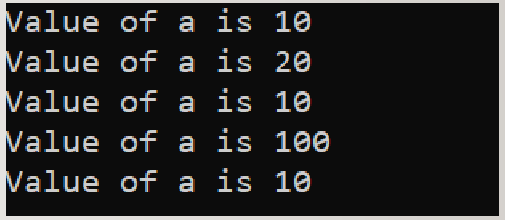
Assignment Operators

CODE



```
int main()
{
// Assigning value 10 to a
    // using "=" operator
    int a = 10;
    printf("Value of a is %d\n", a);

    // Assigning value by adding 10 to a
    // using "+=" operator
    a += 10;
    printf("Value of a is %d\n", a);

    // Assigning value by subtracting 10 from a
    // using "-=" operator
    a -= 10;
    printf("Value of a is %d\n", a);

    // Assigning value by multiplying 10 to a
    // using "*=" operator
    a *= 10;
    printf("Value of a is %d\n", a);

    // Assigning value by dividing 10 from a
    // using "/=" operator
    a /= 10;
    printf("Value of a is %d\n", a);
    return 0;
}
```

OUTPUT



```
Value of a is 10
Value of a is 20
Value of a is 10
Value of a is 100
Value of a is 10
```

DESCRIPTION

- In this example we create variable a and operation happened in Assignment operator.

C operators in c.

Relational Operators

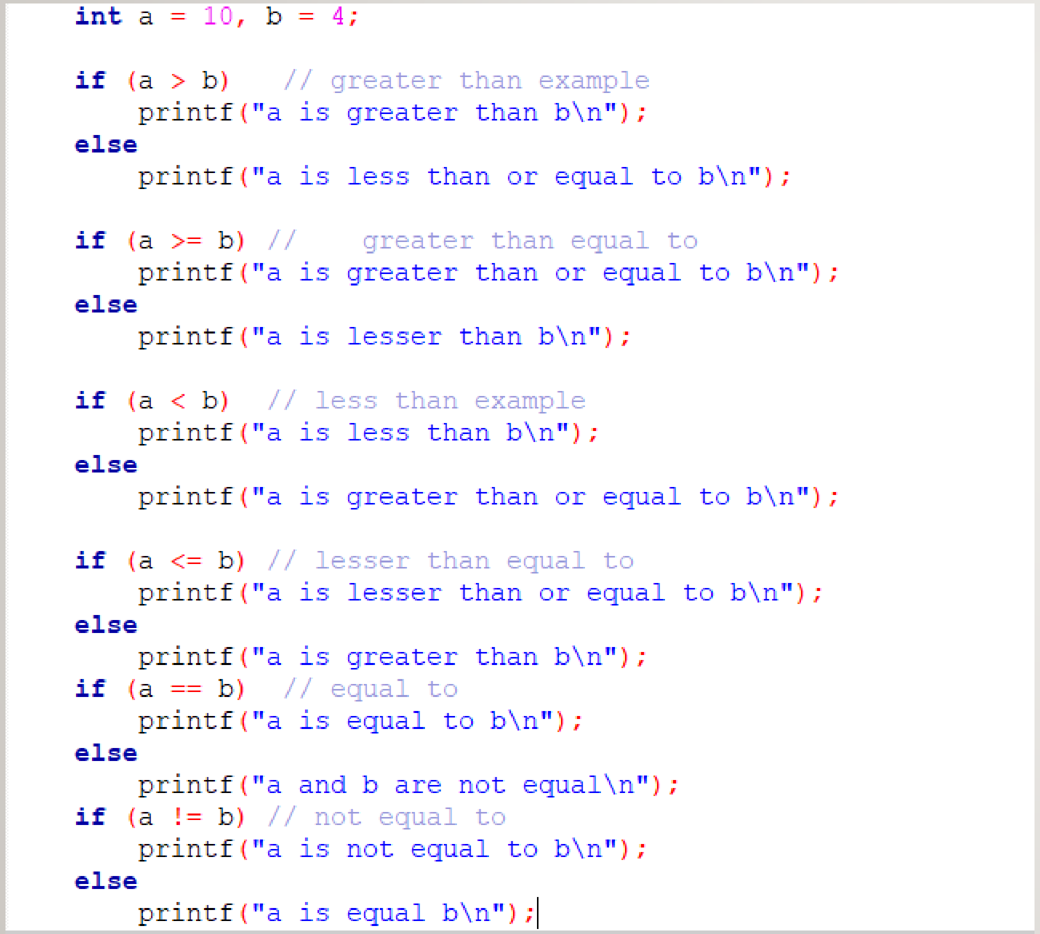
Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes tru	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

C OPERATORS IN C

C operators in c.

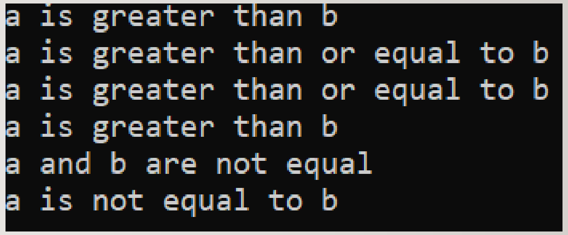
Relational Operators

CODE



```
    int a = 10, b = 4;

    if (a > b)    // greater than example
        printf("a is greater than b\n");
    else
        printf("a is less than or equal to b\n");

    if (a >= b) //     greater than equal to
        printf("a is greater than or equal to b\n");
    else
        printf("a is lesser than b\n");

    if (a < b)   // less than example
        printf("a is less than b\n");
    else
        printf("a is greater than or equal to b\n");

    if (a <= b)  // lesser than equal to
        printf("a is lesser than or equal to b\n");
    else
        printf("a is greater than b\n");
    if (a == b)   // equal to
        printf("a is equal to b\n");
    else
        printf("a and b are not equal\n");
    if (a != b)  // not equal to
        printf("a is not equal to b\n");
    else
        printf("a is equal b\n");
```

OUTPUT



```
a is greater than b
a is greater than or equal to b
a is greater than or equal to b
a is greater than b
a and b are not equal
a is not equal to b
```

DESCRIPTION

- In this example we create variable a and b operation happened in Relational   operator.

C operators in c.

Logical Operators

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

C operators in c.

Bitwise Operators

➢ A bitwise operator works on each bit of data. Bitwise operators are used in bit level programming, Assume variable A holds 60 and variable B holds 13.

Operator	Description	Example
&	Binary AND Operator.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator.	(~A) will give -61 which is 1100 0011
<<	Binary Left Shift Operator	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator.	A >> 2 will give 15 which is 0000 1111

C operators in c.

Bitwise Operators

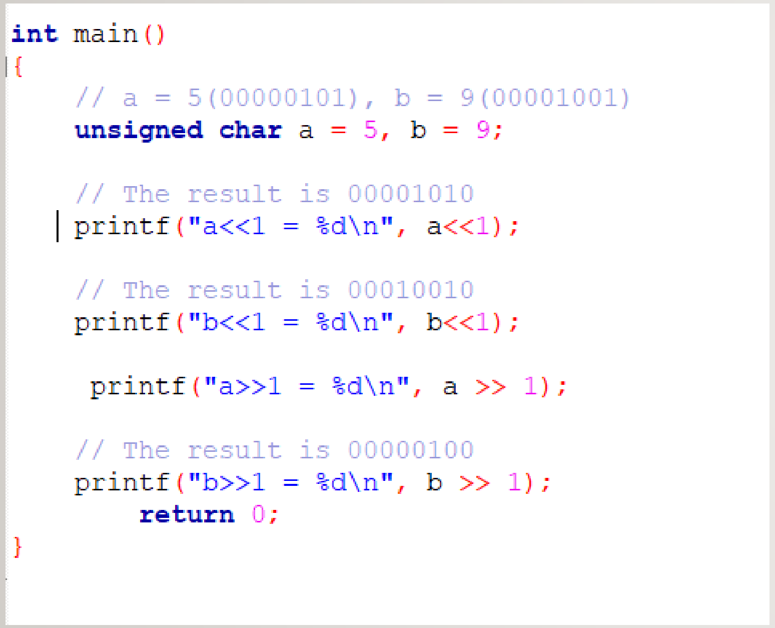
- For right shifting using (>>):
  - Positive and Negative numbers differs also in bitwise operations.
  - the positive numbers :0 is added to the positive numbers from left.
  - negative number :1 is added from left.
  - The left shift and right shift operators should not be used for negative numbers. The result of is undefined behavior if any of the operands is a negative number. For example results of both -1 << 1 and 1 << -1 is undefined.
  - but in some compilers doing a such operation on negative, numbers causes un expected behavior.
  - right shifting an integer "**x**" with an integer "**y**" denoted as '**(x>>y)**' is equivalent to dividing x with 2^y.

- For left shifts using (<<):
  - the bits of the first operand, the second operand decides the number of places to shift. Or in other words left shifting an integer "**x**" with an integer "**y**" denoted as '**(x<<y)**' is equivalent to multiplying **x** with **2^y** (2 raised to power y).

C operators in c.

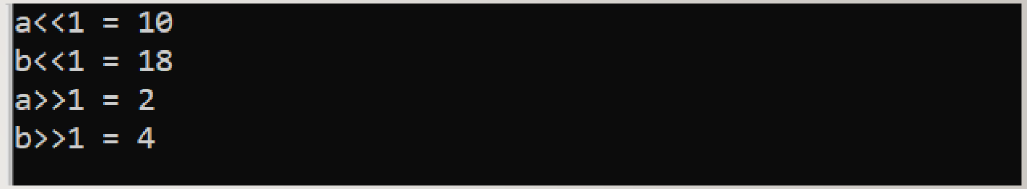
Bitwise Operators

CODE



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

    // The result is 00001010
    printf("a<<1 = %d\n", a<<1);

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

     printf("a>>1 = %d\n", a >> 1);

    // The result is 00000100
    printf("b>>1 = %d\n", b >> 1);
        return 0;
}
```

OUTPUT



```
a<<1 = 10
b<<1 = 18
a>>1 = 2
b>>1 = 4
```

DESCRIPTION

- In this example we create variable a and b operation happened in Bitwise operator.

C OPERATORS IN C

C operators in c.

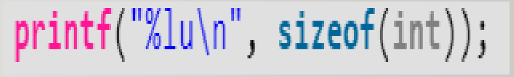
Other Operators

- Other Operators:
  - Comma operators:

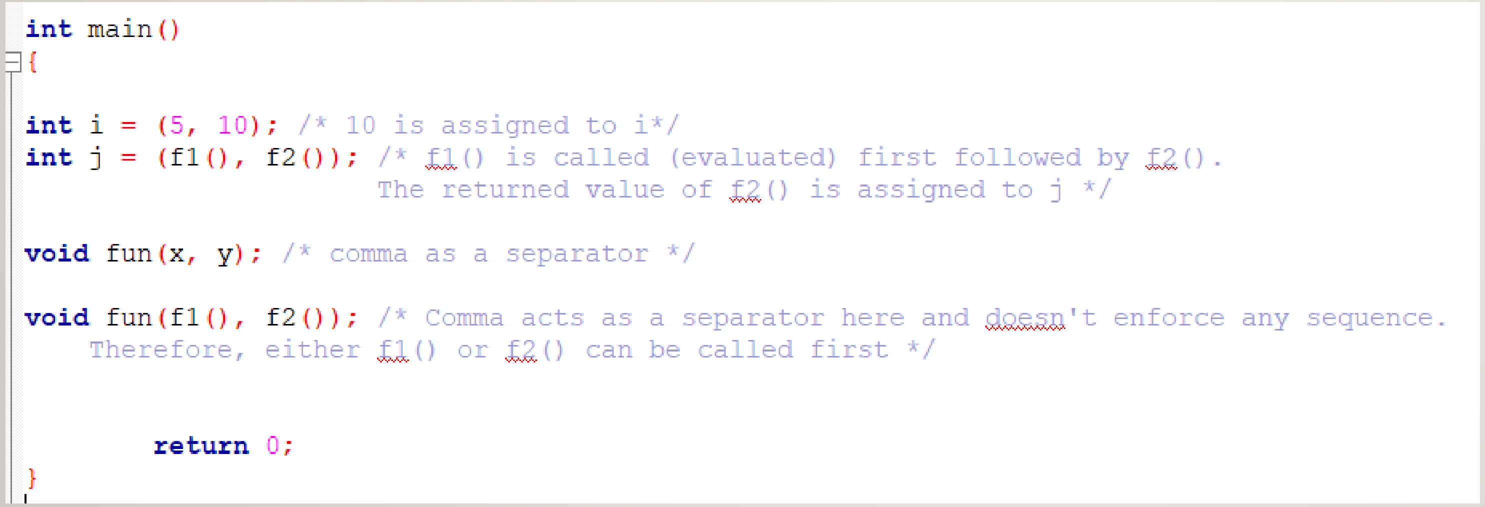
    used to link related expressions together

  - size of operator:

    It is a unary operator which is used in finding the size required for a certain data type



```c
printf("%lu\n", sizeof(int));
```



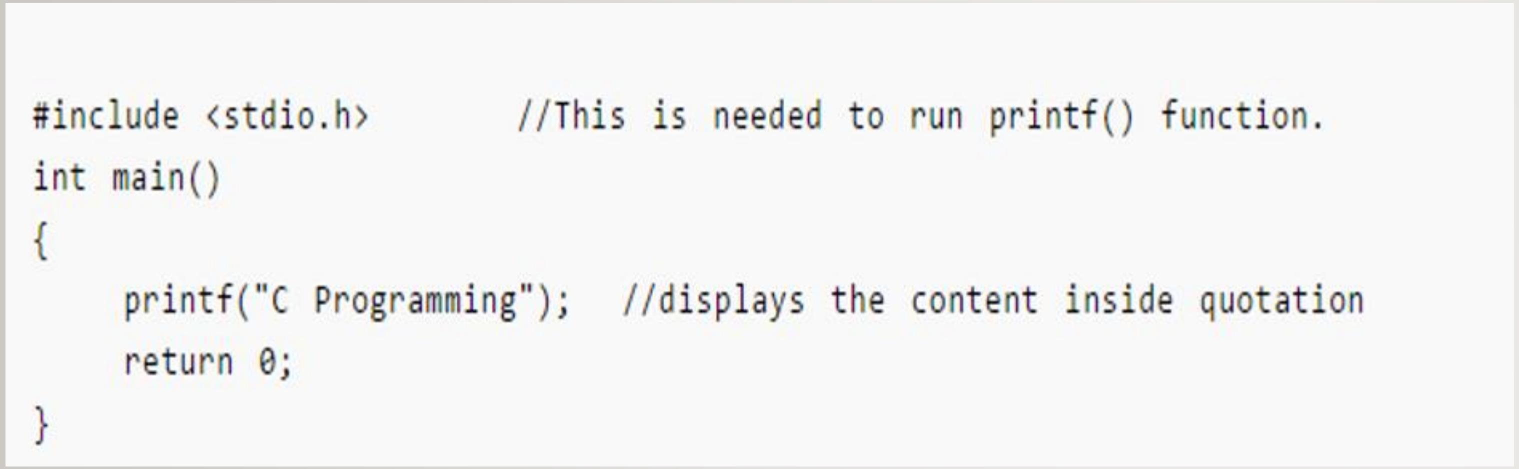
```c
int main()
{

int i = (5, 10); /* 10 is assigned to i*/
int j = (f1(), f2()); /* f1() is called (evaluated) first followed by f2().
                      The returned value of f2() is assigned to j */

void fun(x, y); /* comma as a separator */

void fun(f1(), f2()); /* Comma acts as a separator here and doesn't enforce any sequence.
    Therefore, either f1() or f2() can be called first */


        return 0;
}
```

C INPUT /OUTPUT

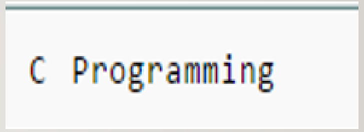
C INPUT /OUTPUT

CODE

OUTPUT



```
#include <stdio.h>       //This is needed to run printf() function.
int main()
{
    printf("C Programming");  //displays the content inside quotation
    return 0;
}
```



```
C Programming
```

C INPUT /OUTPUT

I/O of integers

CODE

OUTPUT

```c
#include<stdio.h>
int main()
{
     int c;
     printf("Enter a number\n");
     scanf("%d",&c);
     printf("Number=%d",c);
     return 0;
}
```

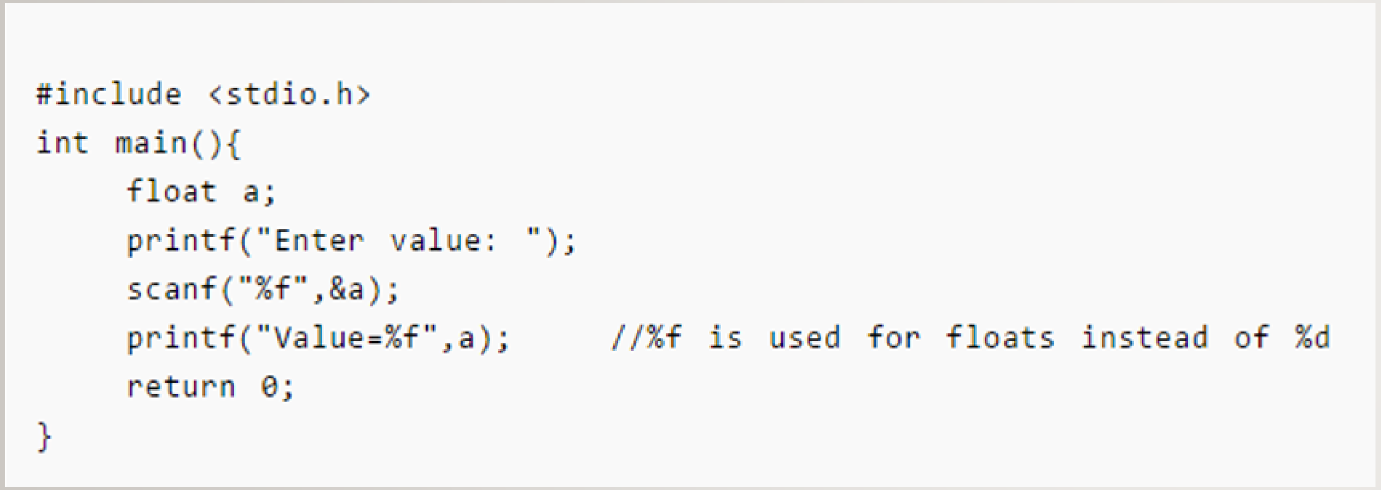
```
Enter a number
4
Number=4
```

C INPUT /OUTPUT

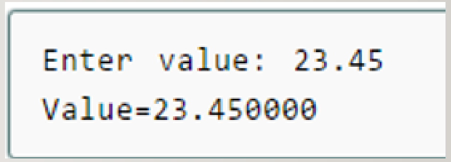
I/O of integers

CODE

OUTPUT



```
#include <stdio.h>
int main(){
     float a;
     printf("Enter value: ");
     scanf("%f",&a);
     printf("Value=%f",a);     //%f is used for floats instead of %d
     return 0;
}
```



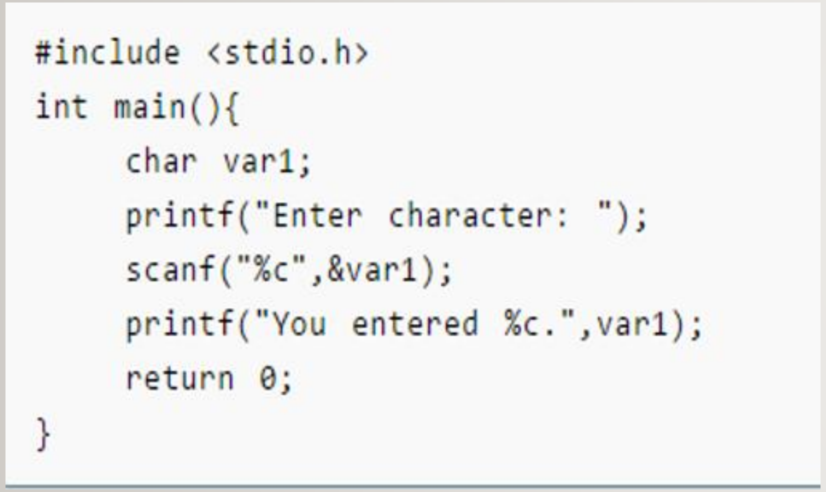
```
Enter value: 23.45
Value=23.450000
```

C INPUT /OUTPUT

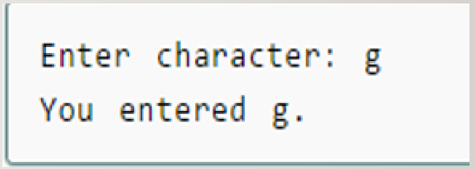
I/O of characters and ASCII code

CODE

OUTPUT



```c
#include <stdio.h>
int main(){
    char var1;
    printf("Enter character: ");
    scanf("%c",&var1);
    printf("You entered %c.",var1);
    return 0;
}
```



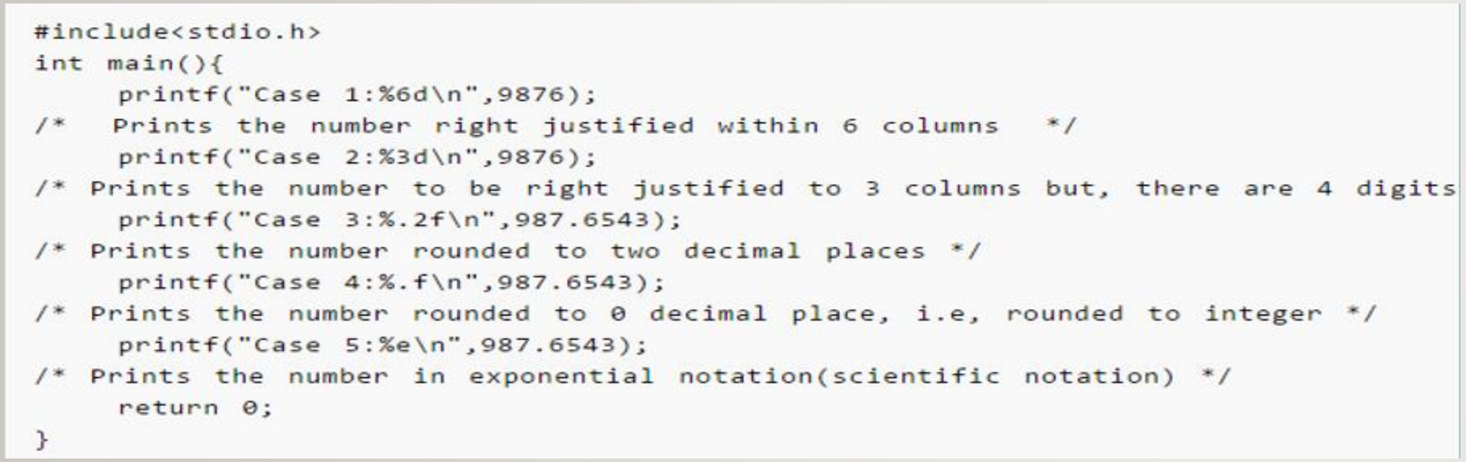
```
Enter character: g
You entered g.
```

C INPUT /OUTPUT

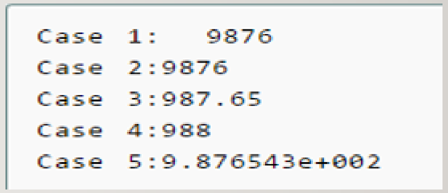
Validation in output for integers and floats

CODE

OUTPUT



```
#include<stdio.h>
int main(){
     printf("Case 1:%6d\n",9876);
/*  Prints the number right justified within 6 columns   */
     printf("Case 2:%3d\n",9876);
/* Prints the number to be right justified to 3 columns but, there are 4 digits
     printf("Case 3:%.2f\n",987.6543);
/* Prints the number rounded to two decimal places */
     printf("Case 4:%.f\n",987.6543);
/* Prints the number rounded to 0 decimal place, i.e, rounded to integer */
     printf("Case 5:%e\n",987.6543);
/* Prints the number in exponential notation(scientific notation) */
     return 0;
}
```

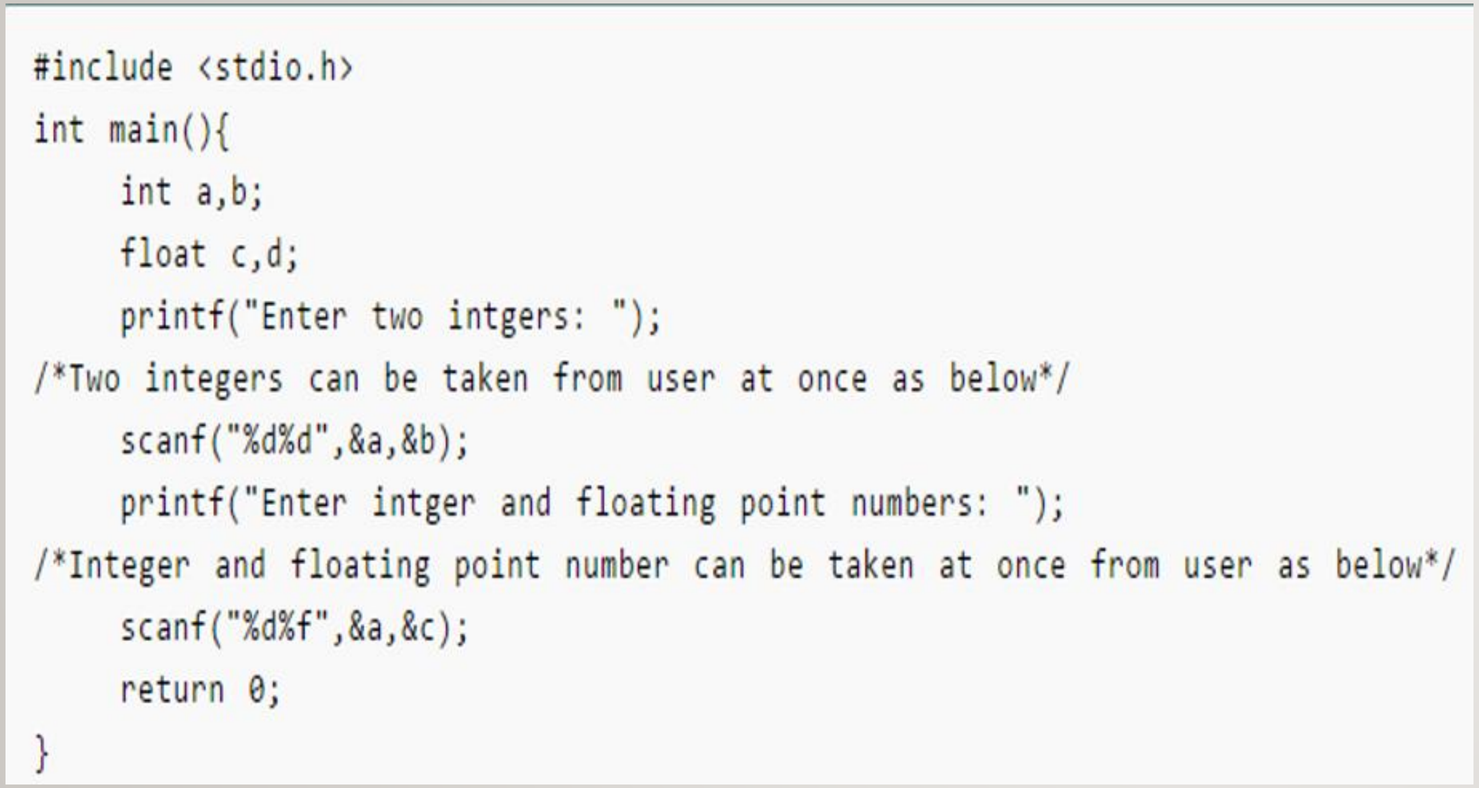


```
Case 1:  9876
Case 2:9876
Case 3:987.65
Case 4:988
Case 5:9.876543e+002
```

C INPUT /OUTPUT

Validation in output for integers and floats

CODE



```
#include <stdio.h>
int main(){
    int a,b;
    float c,d;
    printf("Enter two intgers: ");
/*Two integers can be taken from user at once as below*/
    scanf("%d%d",&a,&b);
    printf("Enter intger and floating point numbers: ");
/*Integer and floating point number can be taken at once from user as below*/
    scanf("%d%f",&a,&c);
    return 0;
}
```

C INPUT /OUTPUT

Lab #1

- Implement a program that takes 2 input numbers from the user, and prints the following:
  - First Number
  - Second Number
  - Addition Result
  - Subtraction Result
  - Multiplication Result
  - Division Result
  - Remainder Result
- Use proper displaying statement for each displayed value.