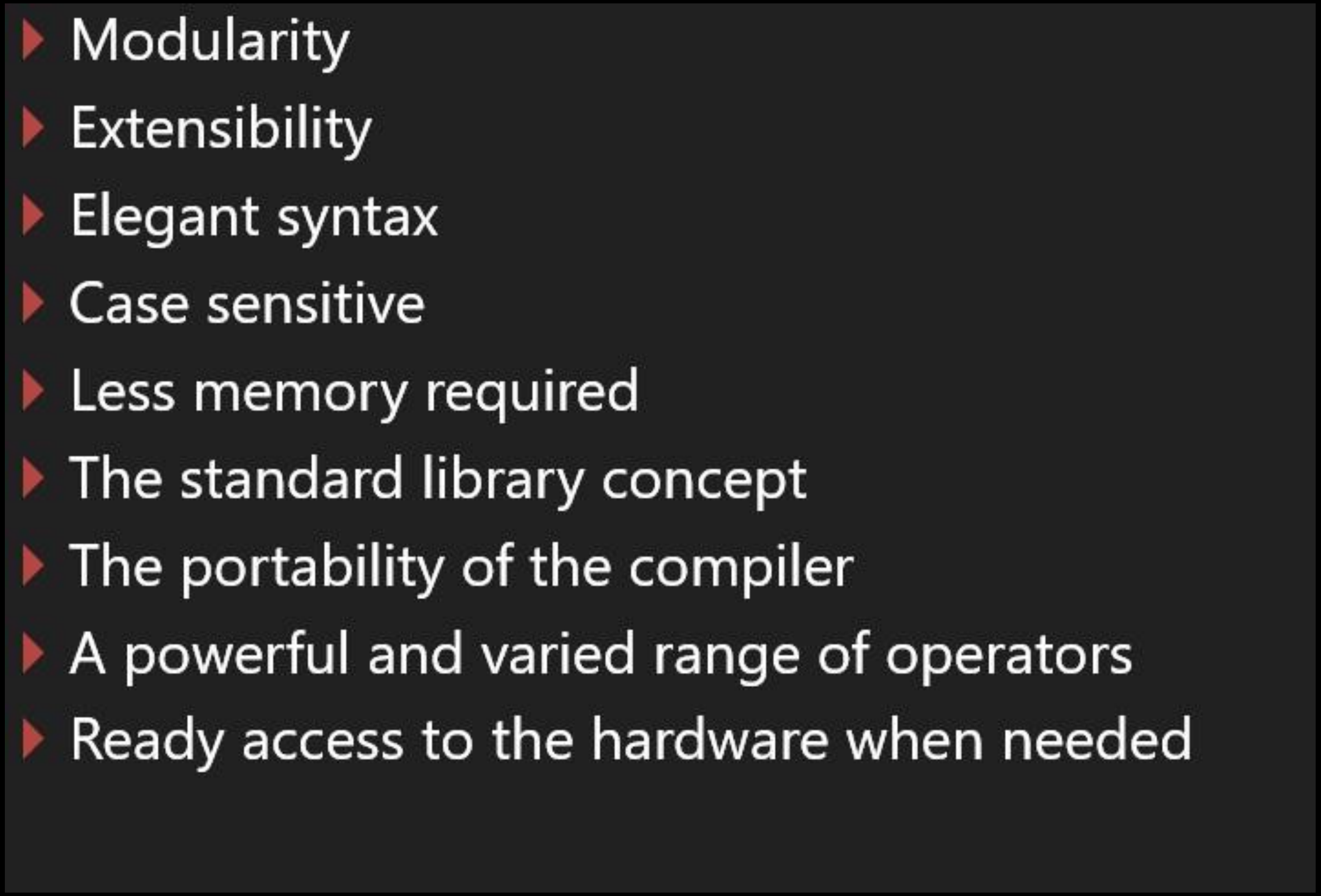
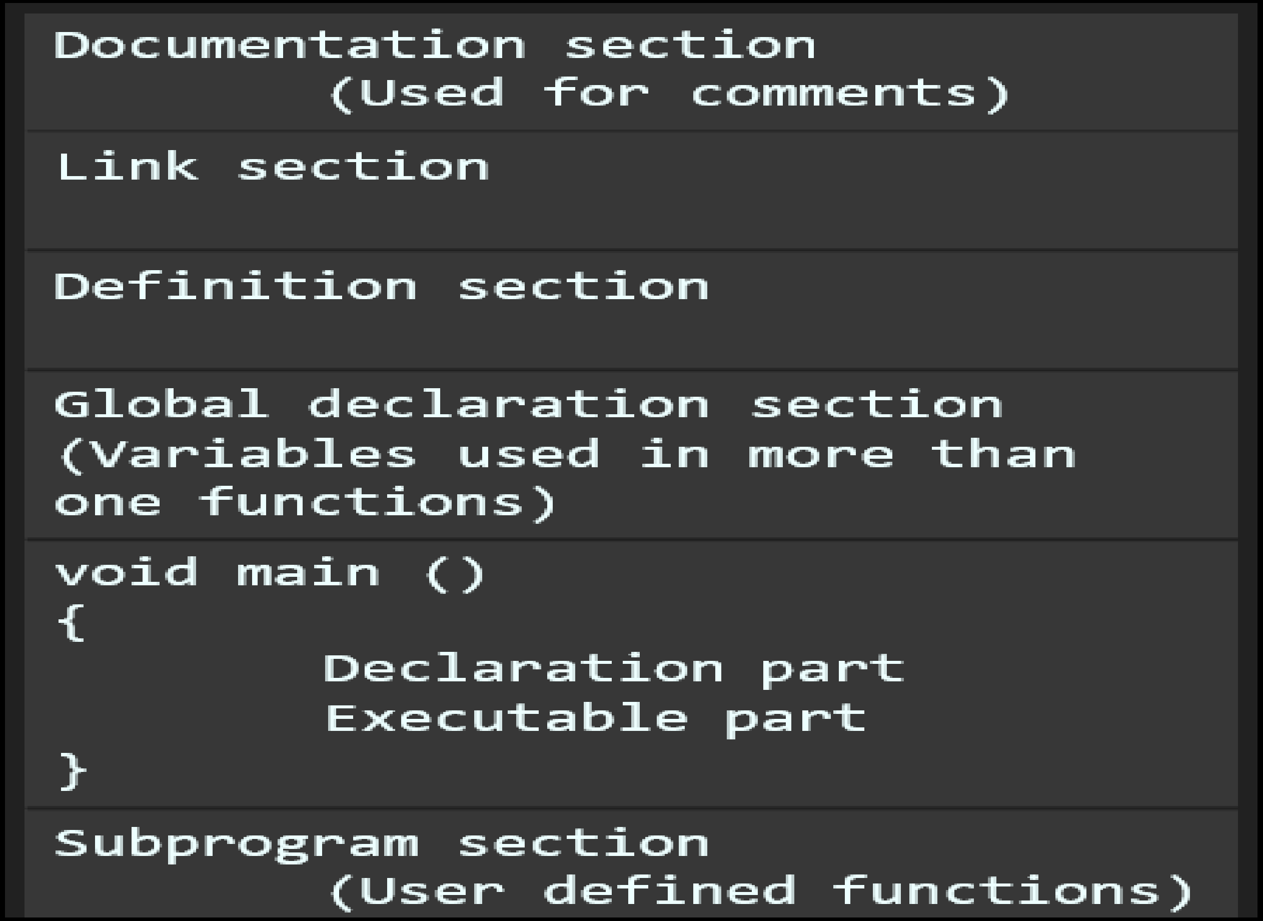
Unit 1

Features of C Language



- Modularity
- Extensibility
- Elegant syntax
- Case sensitive
- Less memory required
- The standard library concept
- The portability of the compiler
- A powerful and varied range of operators
- Ready access to the hardware when needed

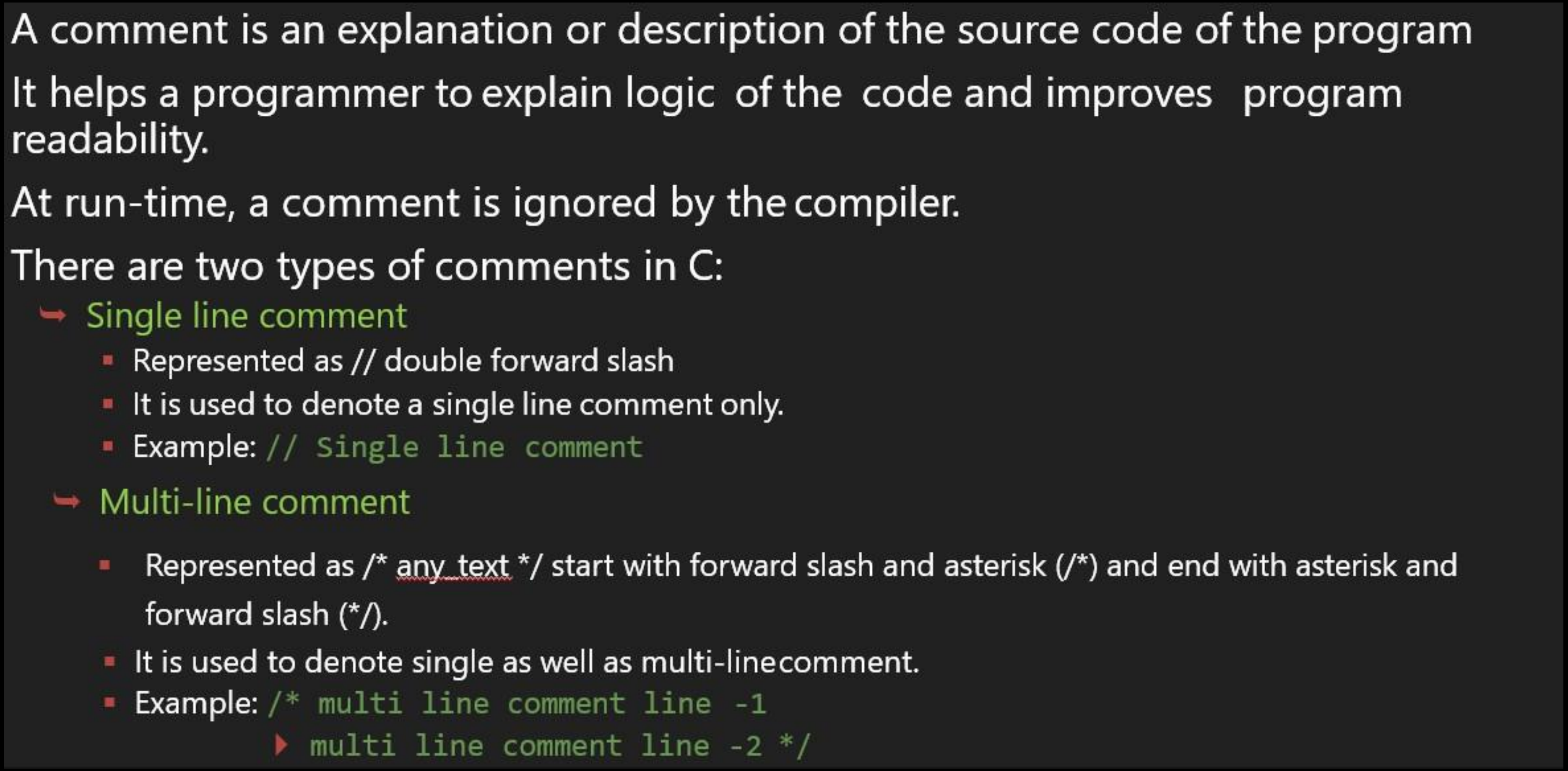
Structure of C Program



```
Documentation section
        (Used for comments)
Link section

Definition section

Global declaration section
(Variables used in more than
one functions)
void main ()
{
        Declaration part
        Executable part
}
Subprogram section
        (User defined functions)
```

```
//Author,name,date
//library files
#include<stdio.h>
#include<math.h>
#include<string.h>
#include<stdlib.h>
//declaration constants, global
variables,userdefined functions
void main()
{
   Declaration part;
  int roll_no;





}
User defined function()
{


}
```

Comments



A comment is an explanation or description of the source code of the program

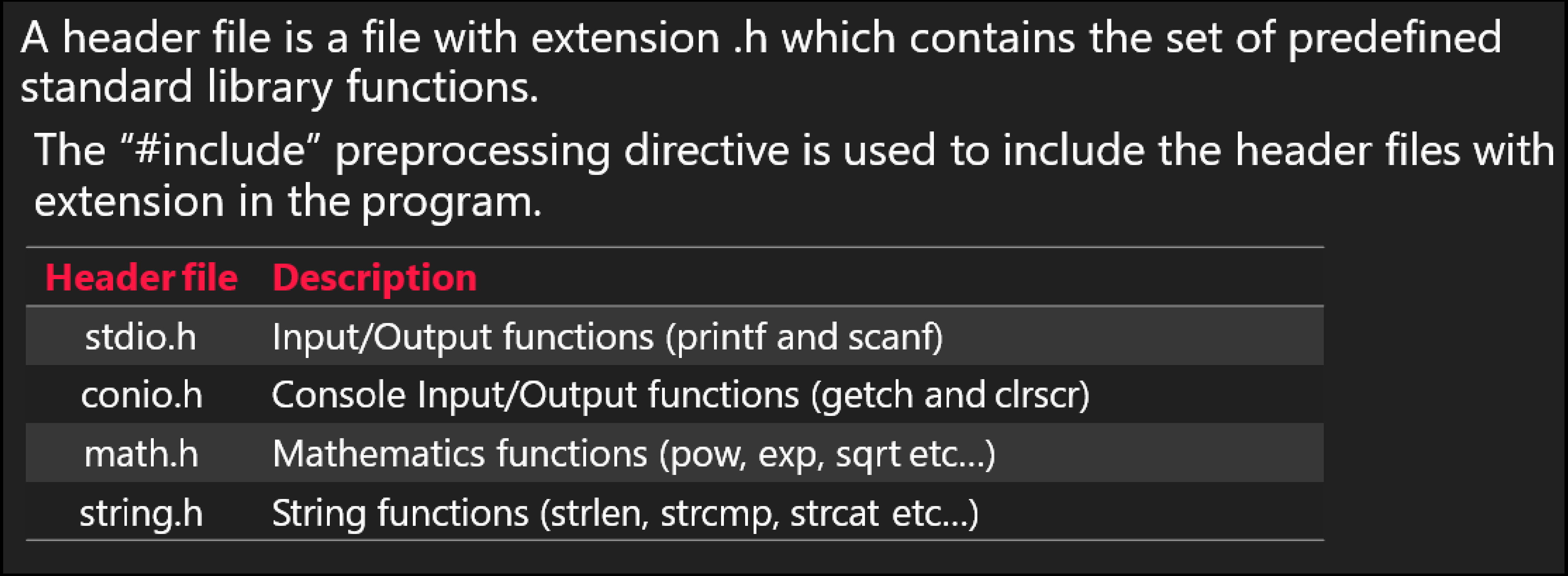
It helps a programmer to explain logic of the code and improves program readability.

At run-time, a comment is ignored by the compiler.

There are two types of comments in C:

- Single line comment
  - Represented as // double forward slash
  - It is used to denote a single line comment only.
  - Example: `// Single line comment`
- Multi-line comment
  - Represented as /* any_text */ start with forward slash and asterisk (/*) and end with asterisk and forward slash (*/).
  - It is used to denote single as well as multi-linecomment.
  - Example: `/* multi line comment line -1`
    - `multi line comment line -2 */`

Header Files



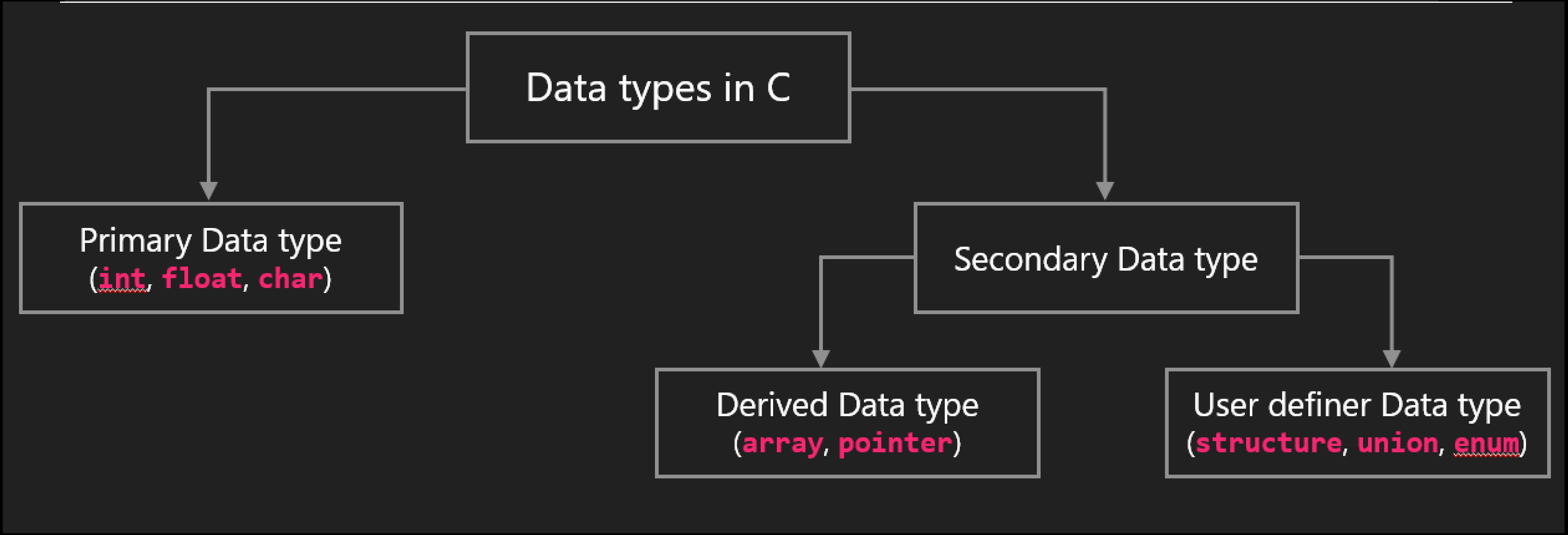
A header file is a file with extension .h which contains the set of predefined standard library functions.

The “#include” preprocessing directive is used to include the header files with extension in the program.

Header file	Description
stdio.h	Input/Output functions (printf and scanf)
conio.h	Console Input/Output functions (getch and clrscr)
math.h	Mathematics functions (pow, exp, sqrt etc…)
string.h	String functions (strlen, strcmp, strcat etc…)

Data Types in C

Data types are defined as the data storage format that a variable can store a data. It determines the type and size of data associated with variables.



Data types in C

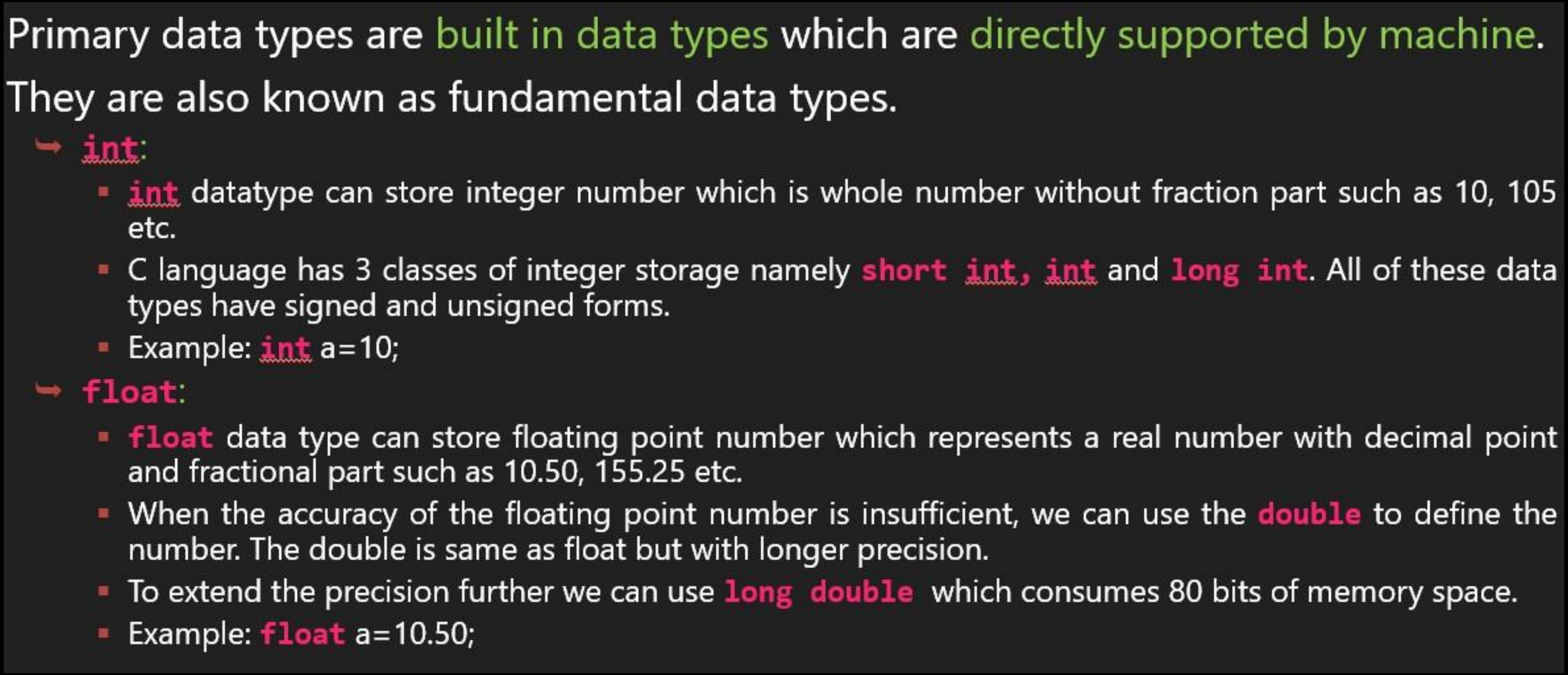
Primary Data type (int, float, char)

Secondary Data type

Derived Data type (array, pointer)

User definer Data type (structure, union, enum)

Primary Data Type

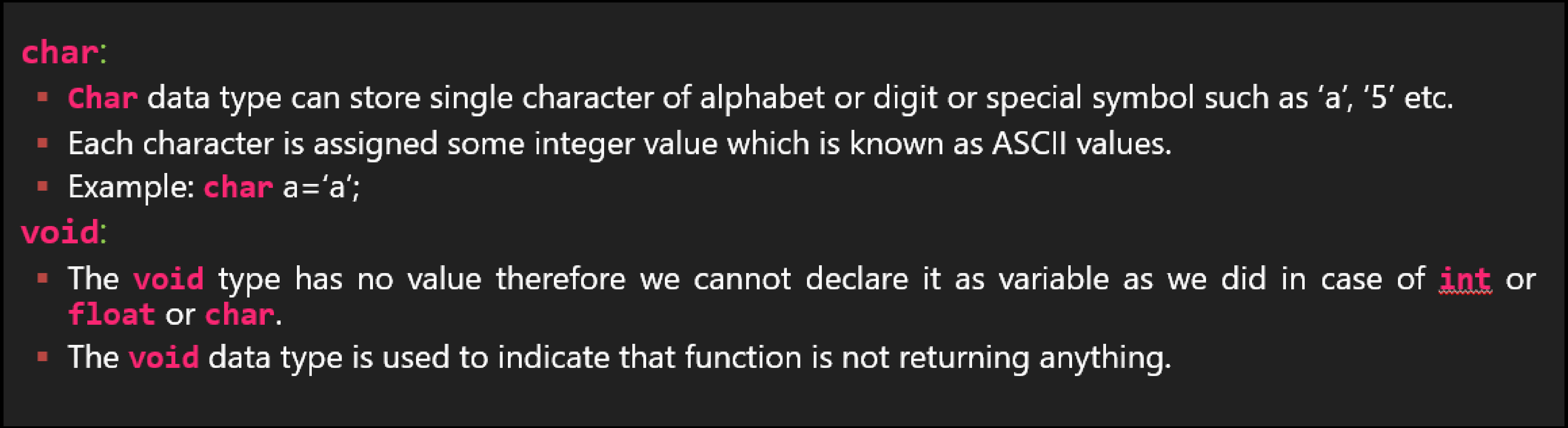


Primary data types are built in data types which are directly supported by machine.

They are also known as fundamental data types.

- int:
  - int datatype can store integer number which is whole number without fraction part such as 10, 105 etc.
  - C language has 3 classes of integer storage namely short int, int and long int. All of these data types have signed and unsigned forms.
  - Example: int a=10;
- float:
  - float data type can store floating point number which represents a real number with decimal point and fractional part such as 10.50, 155.25 etc.
  - When the accuracy of the floating point number is insufficient, we can use the double to define the number. The double is same as float but with longer precision.
  - To extend the precision further we can use long double which consumes 80 bits of memory space.
  - Example: float a=10.50;

Primary Data types



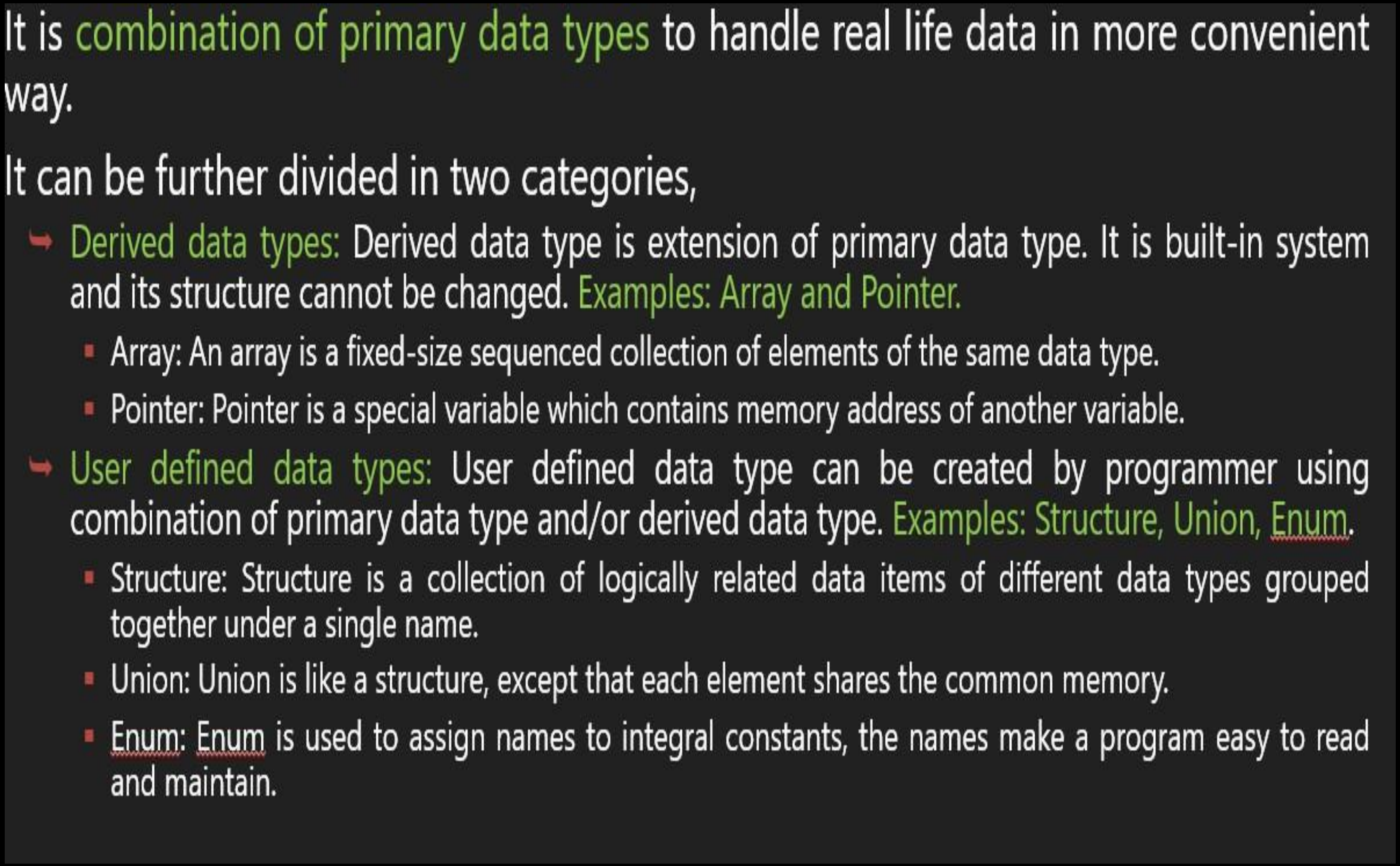
char:

- Char data type can store single character of alphabet or digit or special symbol such as ‘a’, ‘5’ etc.
- Each character is assigned some integer value which is known as ASCII values.
- Example: char a=‘a‘;

void:

- The void type has no value therefore we cannot declare it as variable as we did in case of int or float or char.
- The void data type is used to indicate that function is not returning anything.

Secondary Data types



It is combination of primary data types to handle real life data in more convenient way.

It can be further divided in two categories,

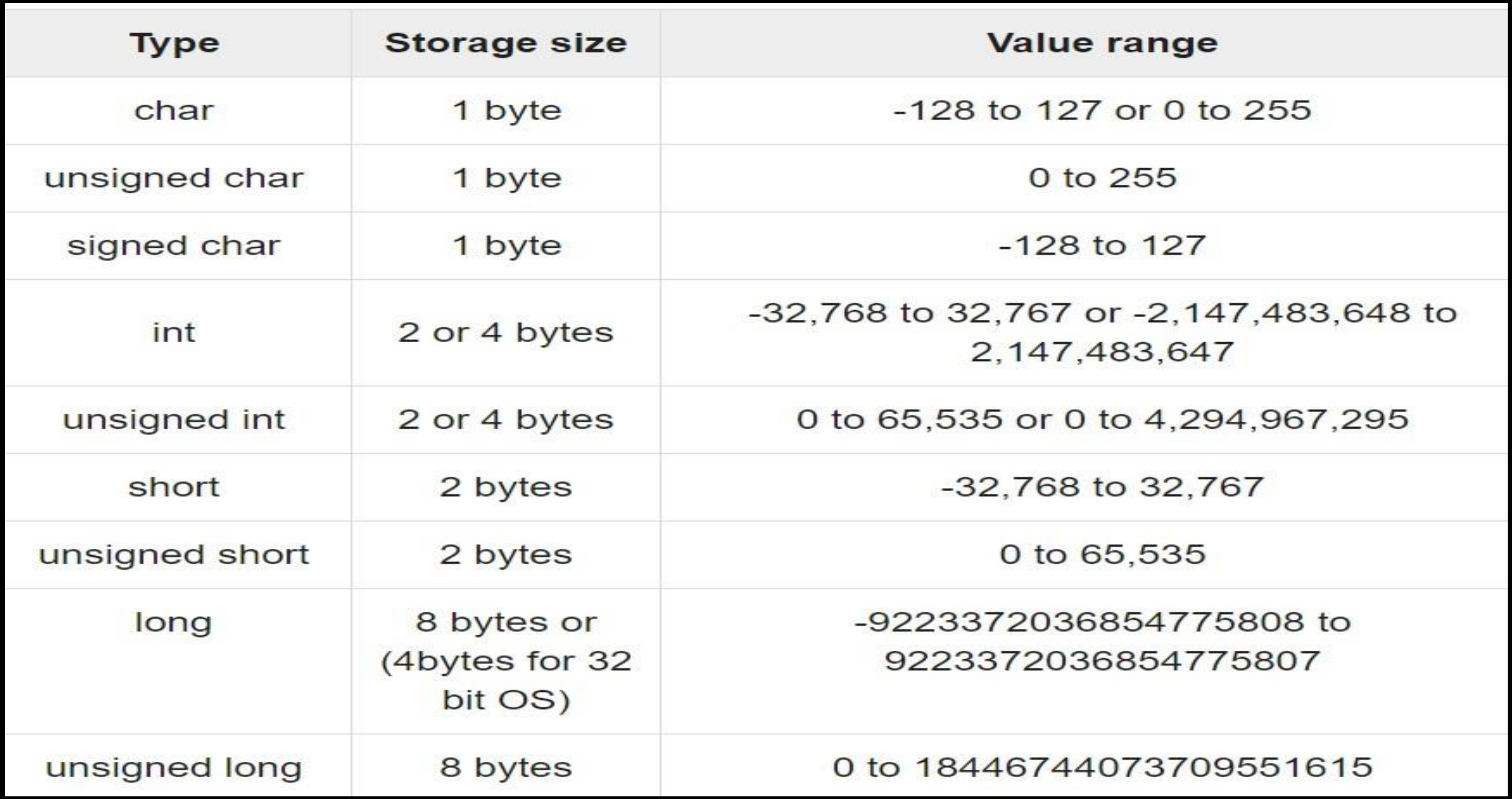
- Derived data types: Derived data type is extension of primary data type. It is built-in system and its structure cannot be changed. Examples: Array and Pointer.
    - Array: An array is a fixed-size sequenced collection of elements of the same data type.
    - Pointer: Pointer is a special variable which contains memory address of another variable.
- User defined data types: User defined data type can be created by programmer using combination of primary data type and/or derived data type. Examples: Structure, Union, Enum.
    - Structure: Structure is a collection of logically related data items of different data types grouped together under a single name.
    - Union: Union is like a structure, except that each element shares the common memory.
    - Enum: Enum is used to assign names to integral constants, the names make a program easy to read and maintain.

```
int rollno=101;
int rollno[26]
```

Data types-int

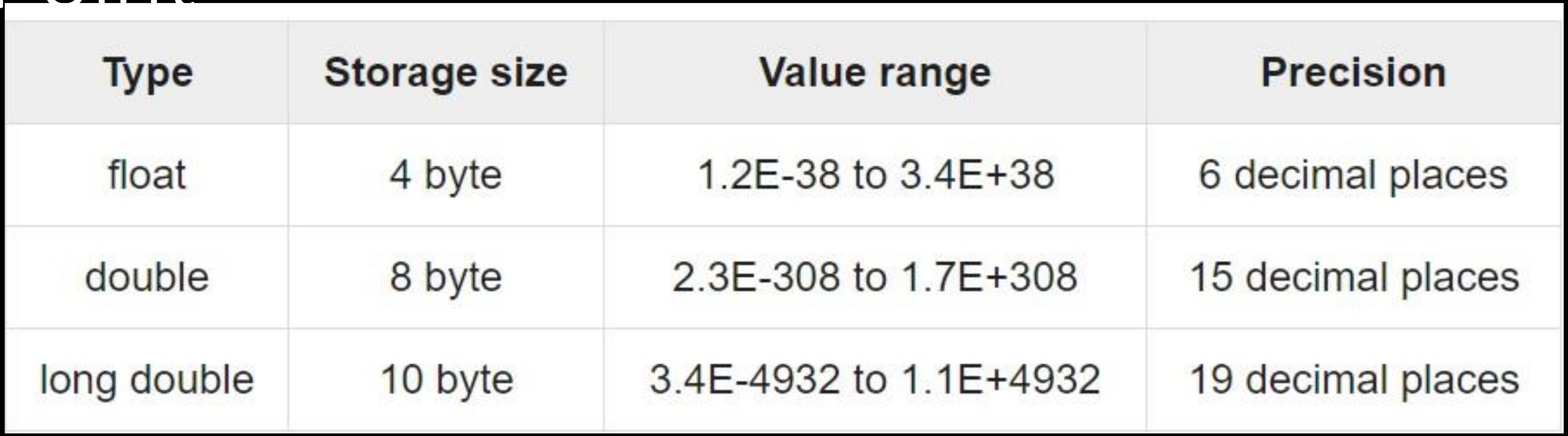
- In C, the **int** data type occupies **2 bytes (16 bits)** of memory to store an integer value.
- Signed and unsigned int
- Int,short,long int a;
- **int** or **signed int** data type denotes a 16 – bit signed integer, which can hold any value between -32,768 (-2 ^{15}) and 32,767 (2 ^{15} -1). **unsigned int** data type denotes a 16 – bit integer and does not use a bit to store the sign. Hence, it can hold only **positive** values between 0 and 65535 (2 ^{16} - 1).
- The OS architecture (i.e., either 16 – bit, 32 – bit or 64 – bit) plays an important role in determining the memory occupied by a numeric data type. In a 16 – bit OS, 2 bytes (16 bits) of memory is allocated to an **int** data type.
- Similarly, **4 bytes** (32 bits) of memory is allocated in a 32 – bit OS, and **8 bytes** (64 bits) of memory is allocated in a 64 – bit OS for an int data type.

Data types-Integer and Character



Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Data types-Floating Point



Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Variables in C

- A variable is a name given to a storage area that our programs can manipulate.
- Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

```
int marks=100;
marks++;
 int _marks;
```

- The name of a variable can be composed of letters, digits, and the underscore character.
- It must begin with either a letter or an underscore.
- Upper and lowercase letters are distinct because C is case-sensitive.

Identifiers in C

- “Identifiers” or “symbols” are the names we give for variables, types, functions, and labels in your program.
- Identifier names must differ in spelling and case from any keywords.
- We cannot use keywords as identifiers; they are reserved for special use.
- A special kind of identifier, called a statement label, can be used in goto statements.
- The first character of an identifier name must be a non-digit.

Variable Definition/Declaration in C

- A variable definition specifies a data type and contains a list of one or more variables of that type as follows −

```
type variable_list;

int marks, roll_number, total_marks;
float percentage, salary;
```

Variable Initialization

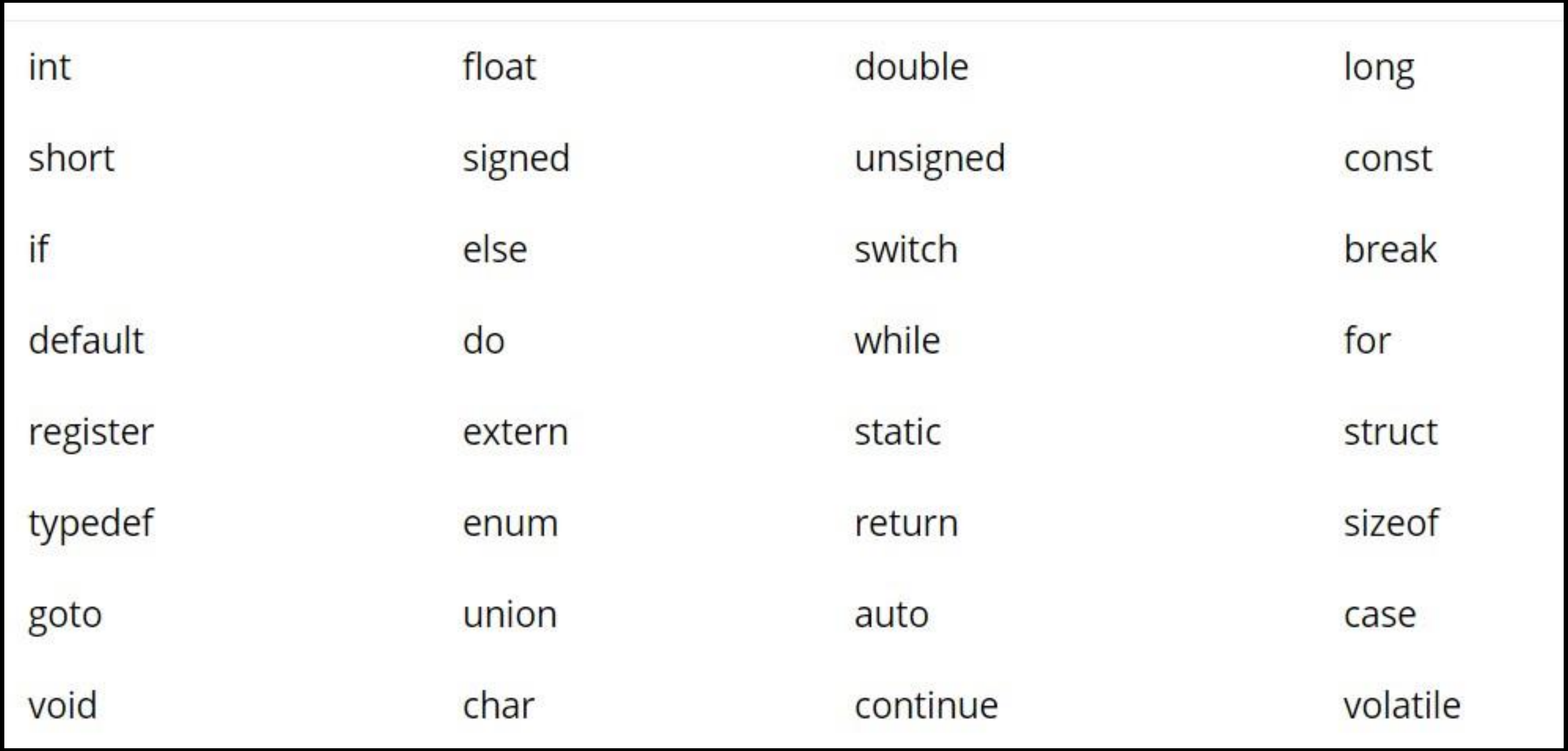
```
float salary=85000.50;
Total_marks=100;
float class_size=50.0;
```

Keywords

- Keywords are the system defined identifiers.
- All keywords have fixed meanings that can not change.
- White spaces are not allowed in keywords.
- Keyword may not be used as user defined identifier.
- It is strongly recommended that keywords should be in lower case letters.

Keywords

- **There are totally 32 keywords in a C programming.**



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

Constants in C

Constants are fixed values that the program may not change during its execution.

These fixed values are also called **literals**.

Constants can be of any of the basic data types like *an integer constant, a floating constant, a character constant, or a string literal*.

Integer Constants:

- An integer constant can be a decimal, octal, or hexadecimal constant.
- A prefix specifies the base : 0x or 0X for hexadecimal, 0 for octal, and decimal constants are without prefix.
- Defining Constants
- There are two ways in C to define constants −
  - Using **#define** preprocessor.
  - Using **const** keyword.

```
85 /* decimal */
 0213 /* octal */
0x4b /* hexadecimal */
30 /* int */
30u /* unsigned int */
 30l /* long */
30ul /* unsigned long */
3.14 /*float constant*/
‘y’ /*character constant*/
```

Defining Constants

The #define Preprocessor

Syntax-#define identifier value

Example

```
#include <stdio.h>
#define LENGTH 10
#define WIDTH  5
int main() {
   int area;
    area = LENGTH * WIDTH;
   printf("value of area : %d", area);
     return 0;
}
```

The const Keyword

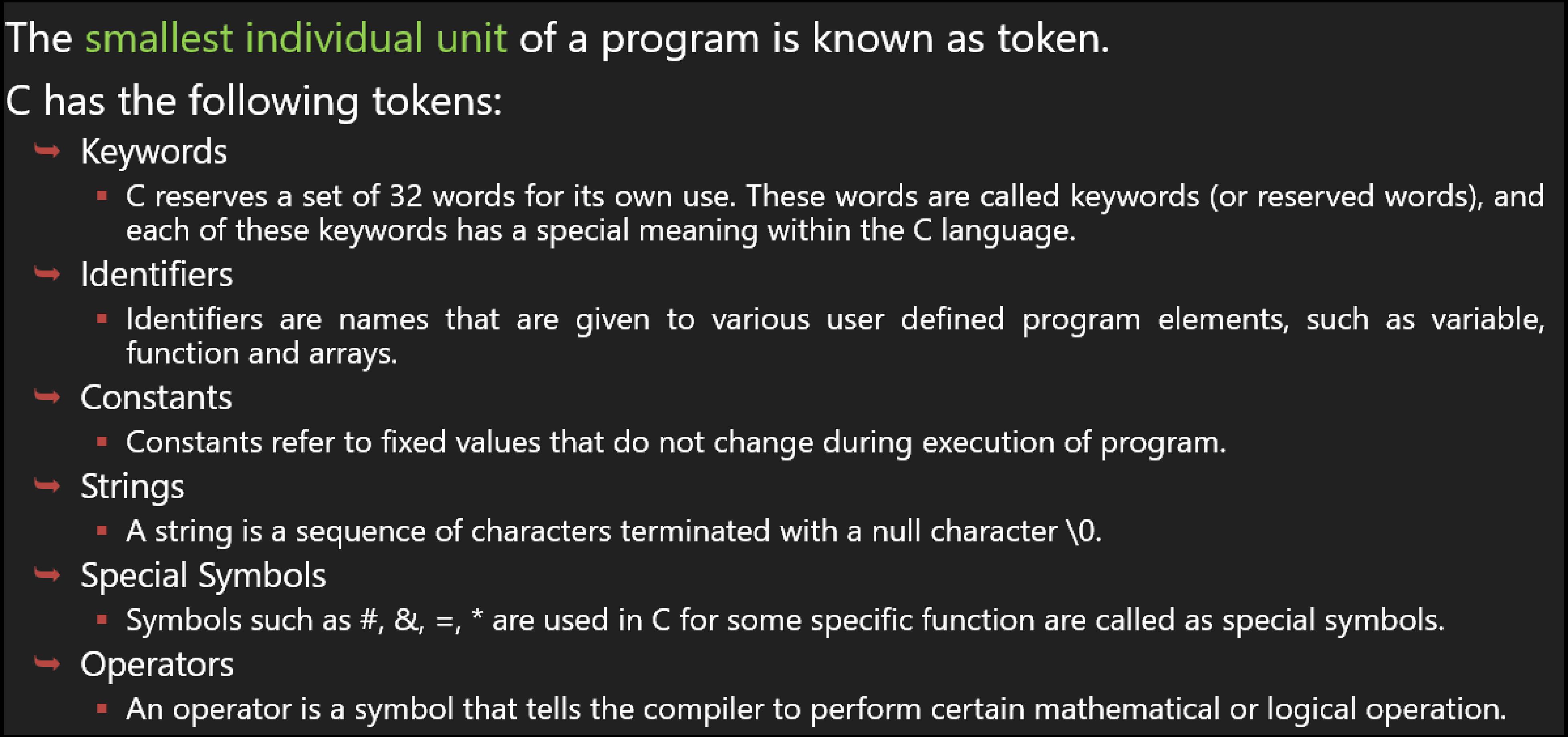
Syntax –

const type variable = value;

Example

```
#include <stdio.h>
int main() {
   const int  LENGTH = 10;
   const int  WIDTH = 5;
   int area;
    area = LENGTH * WIDTH;
   printf("value of area : %d",
area);
     return 0;
}
```

Tokens

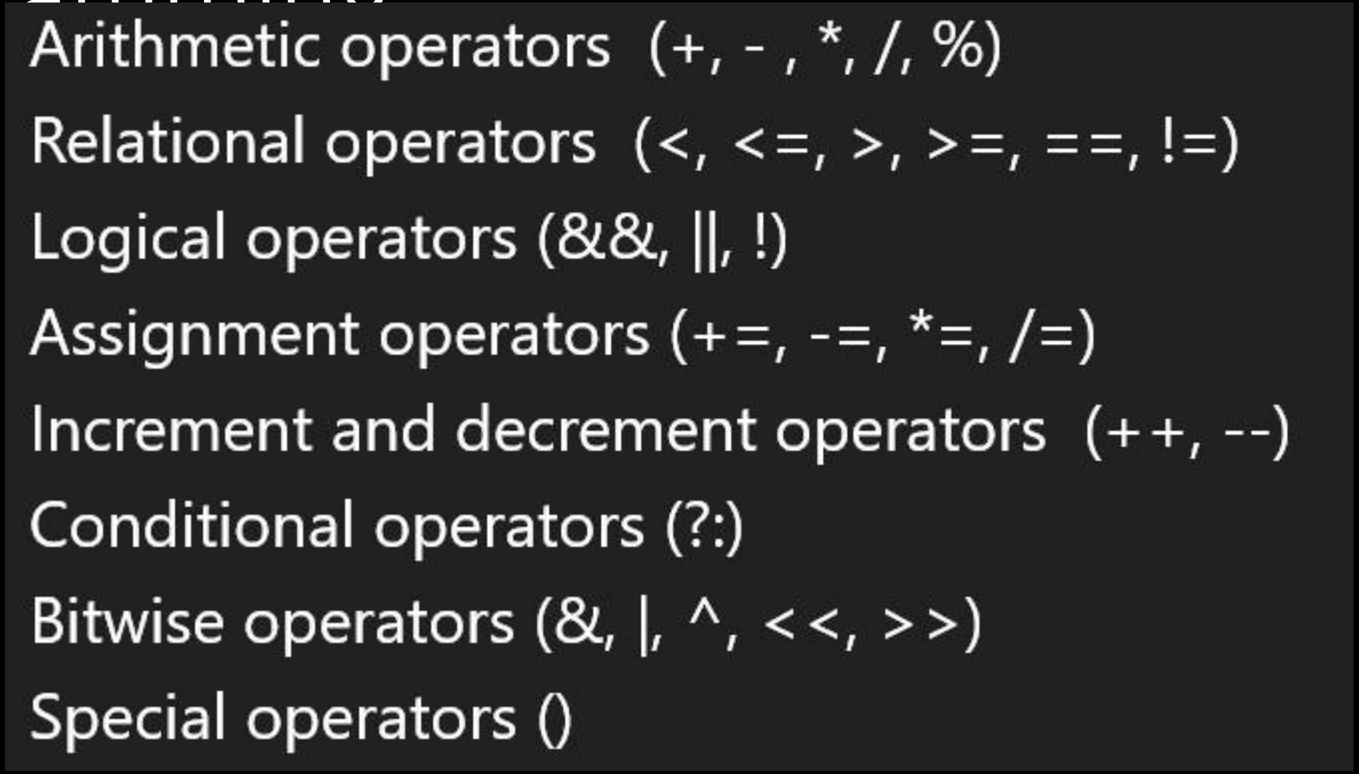


The smallest individual unit of a program is known as token.

C has the following tokens:

- Keywords
  - C reserves a set of 32 words for its own use. These words are called keywords (or reserved words), and each of these keywords has a special meaning within the C language.
- Identifiers
  - Identifiers are names that are given to various user defined program elements, such as variable, function and arrays.
- Constants
  - Constants refer to fixed values that do not change during execution of program.
- Strings
  - A string is a sequence of characters terminated with a null character \0.
- Special Symbols
  - Symbols such as #, &, =, * are used in C for some specific function are called as special symbols.
- Operators
  - An operator is a symbol that tells the compiler to perform certain mathematical or logical operation.

Operators C Programming



Arithmetic operators (+, - , *, /, %)

Relational operators (<, <=, >, >=, ==, !=)

Logical operators (&&, ||, !)

Assignment operators (+=, -=, *=, /=)

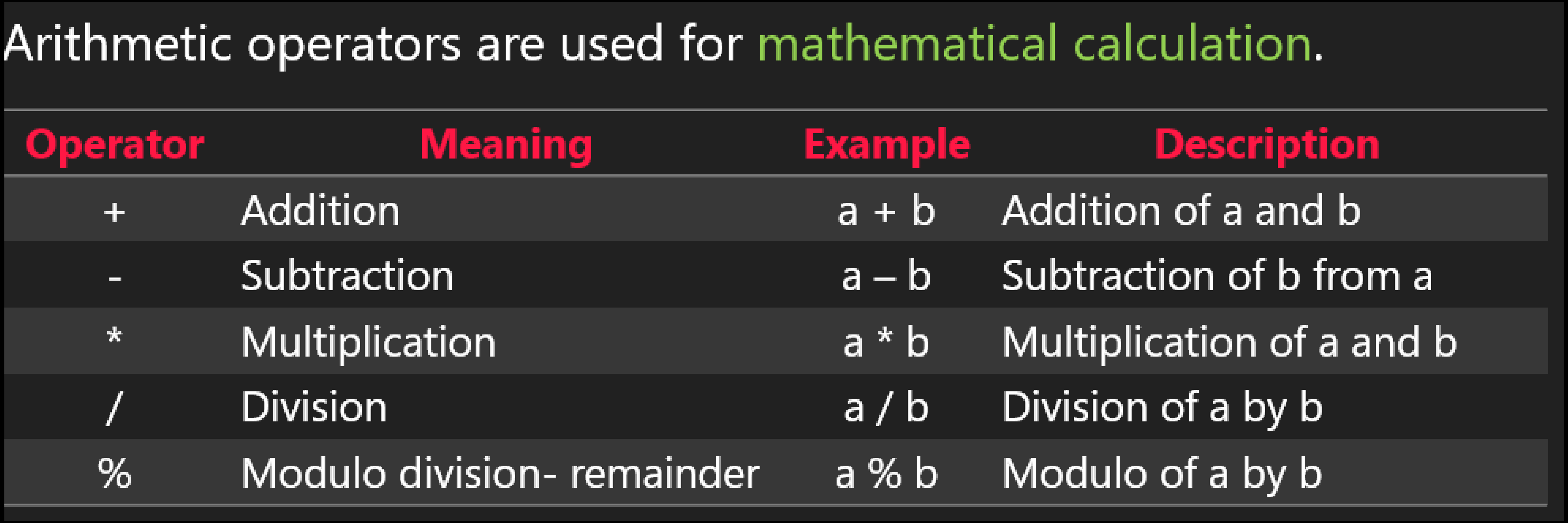
Increment and decrement operators (++, --)

Conditional operators (?:)

Bitwise operators (&, |, ^, <<, >>)

Special operators ()

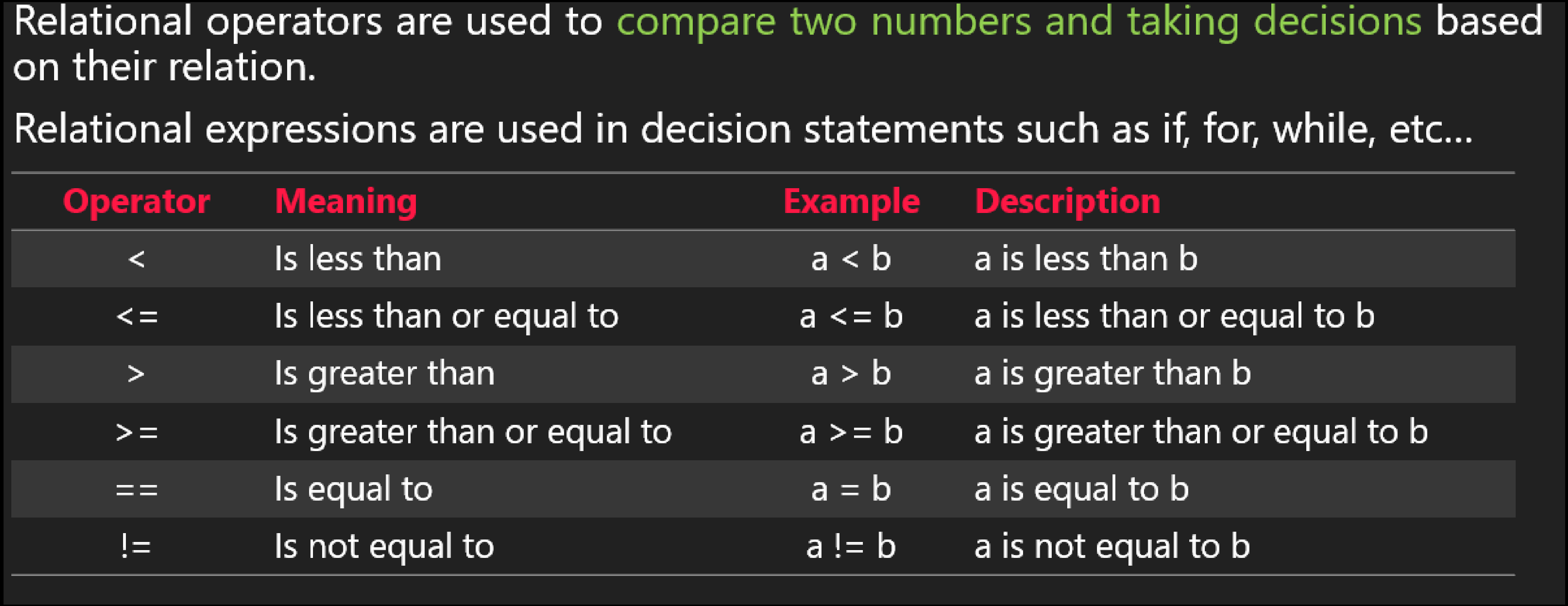
Arithmetic Operators



Arithmetic operators are used for mathematical calculation.

Operator	Meaning	Example	Description
+	Addition	a + b	Addition of a and b
-	Subtraction	a – b	Subtraction of b from a
*	Multiplication	a * b	Multiplication of a and b
/	Division	a / b	Division of a by b
%	Modulo division- remainder	a % b	Modulo of a by b

Relational Operators

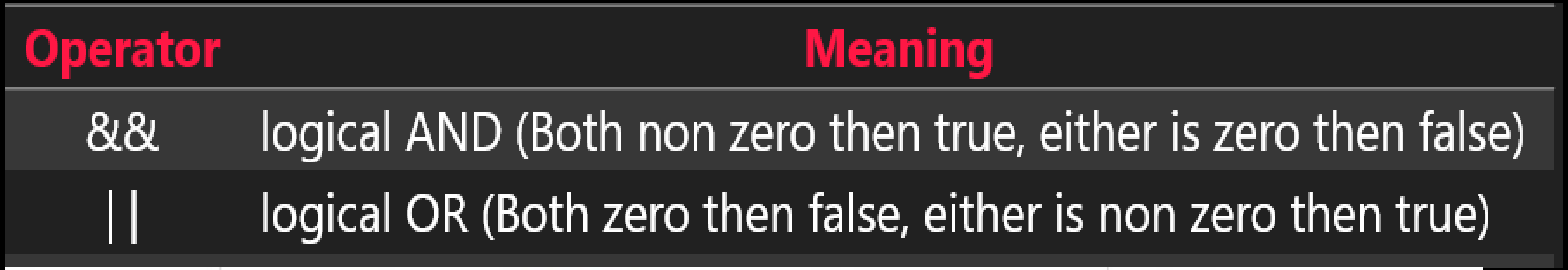


Relational operators are used to compare two numbers and taking decisions based on their relation.

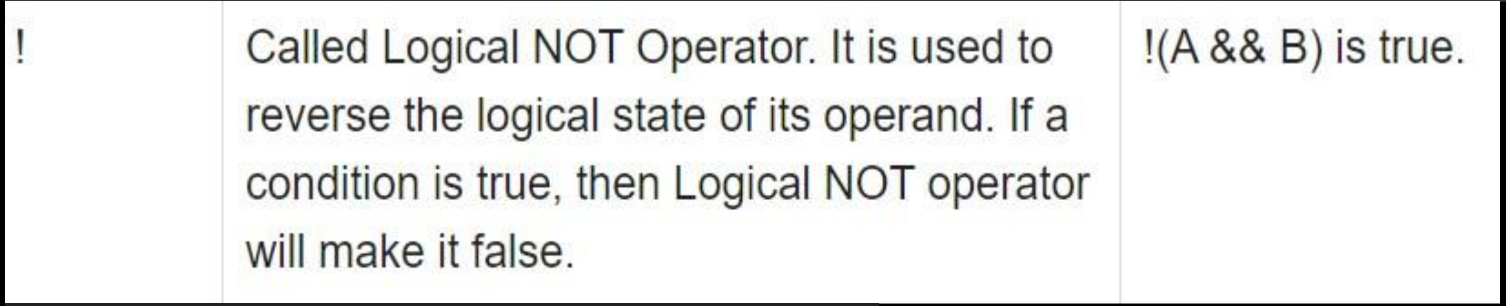
Relational expressions are used in decision statements such as if, for, while, etc…

Operator	Meaning	Example	Description
<	Is less than	a < b	a is less than b
<=	Is less than or equal to	a <= b	a is less than or equal to b
>	Is greater than	a > b	a is greater than b
>=	Is greater than or equal to	a >= b	a is greater than or equal to b
==	Is equal to	a = b	a is equal to b
!=	Is not equal to	a != b	a is not equal to b

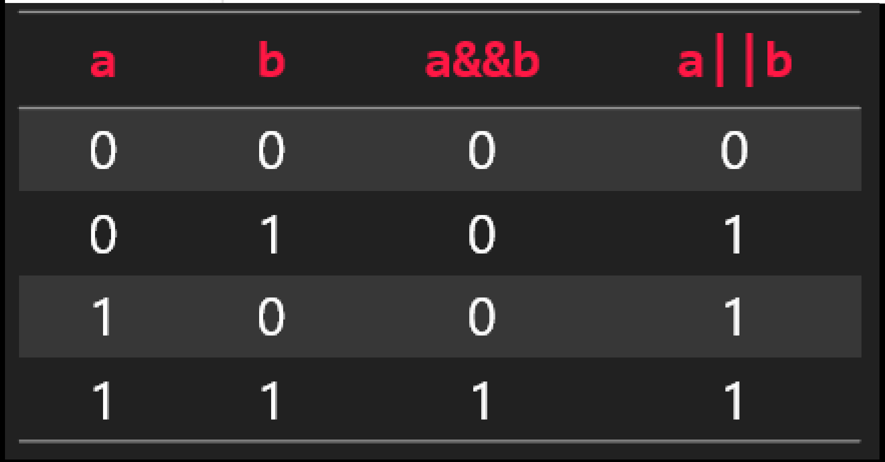
Logical Operators



Operator	Meaning
&&	logical AND (Both non zero then true, either is zero then false)
||	logical OR (Both zero then false, either is non zero then true)

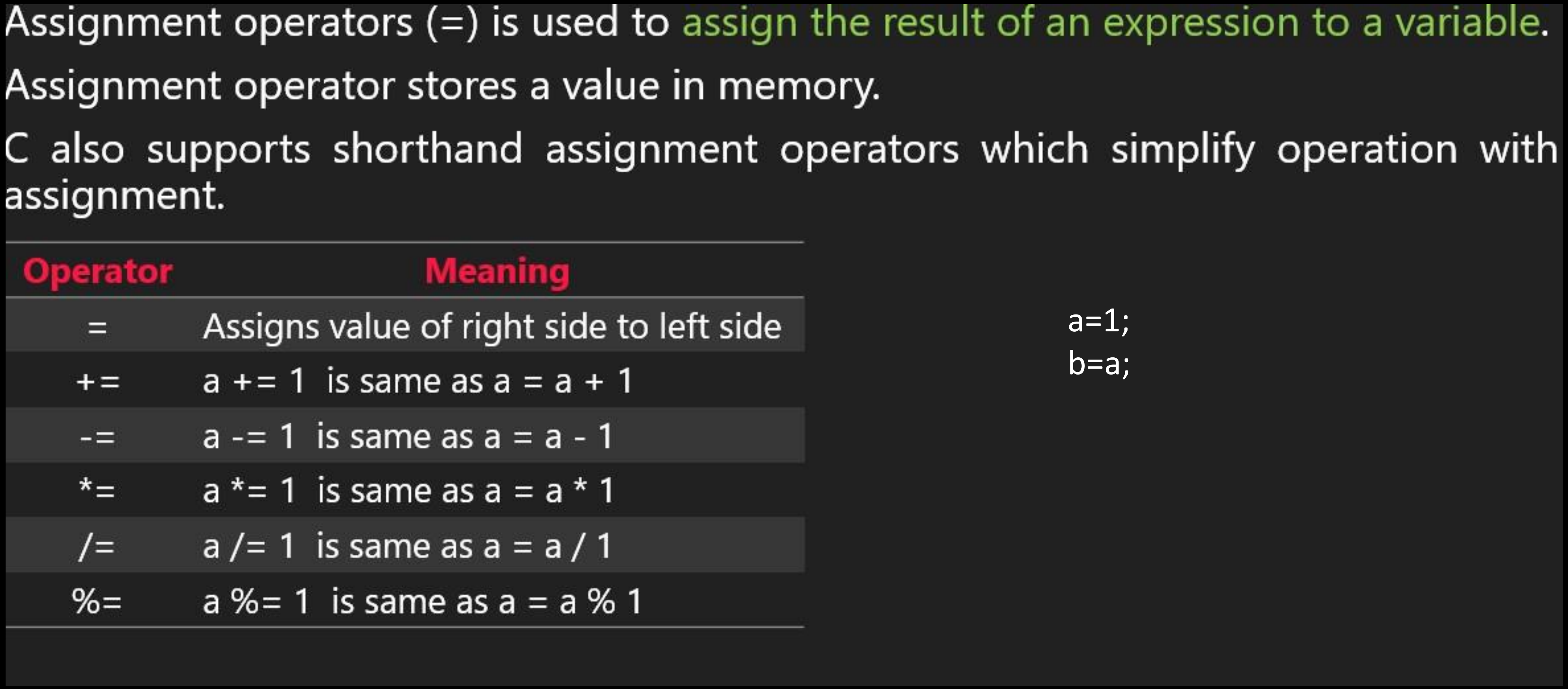


!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.



a	b	a&&b	a||b
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Assignment Operator



Assignment operators (=) is used to assign the result of an expression to a variable.

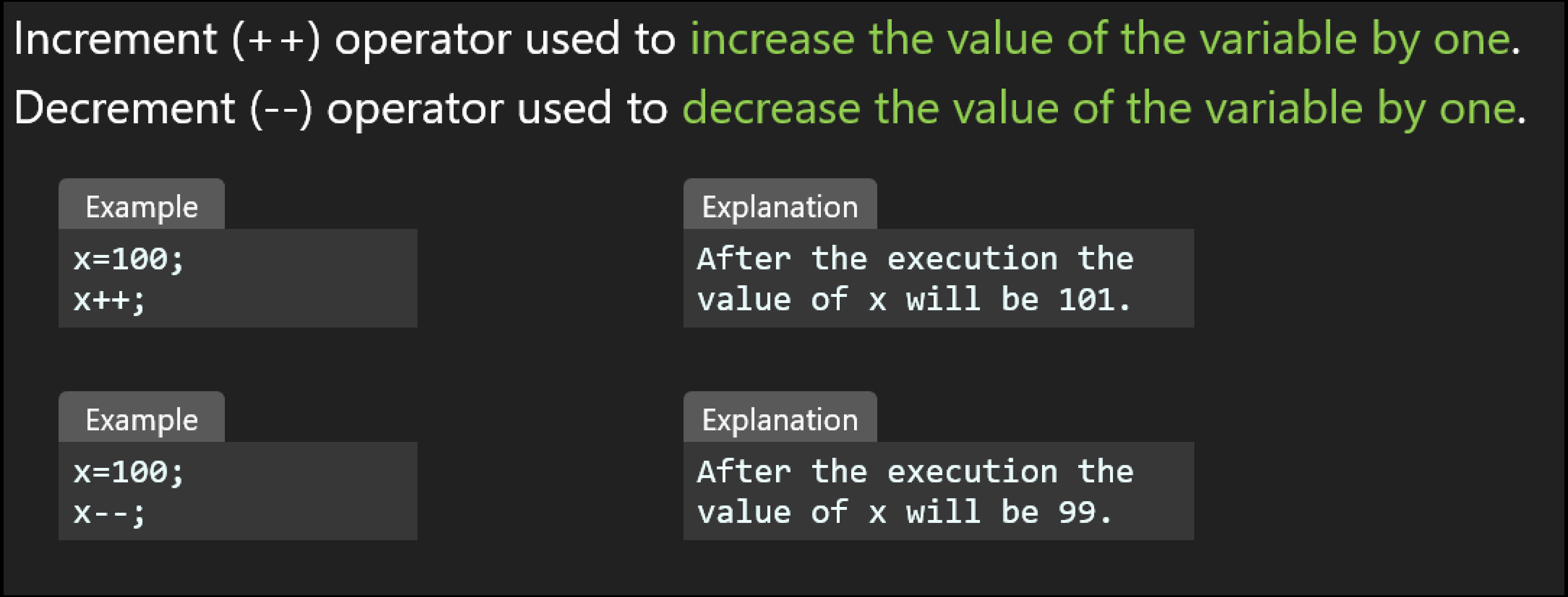
Assignment operator stores a value in memory.

C also supports shorthand assignment operators which simplify operation with assignment.

Operator	Meaning
=	Assigns value of right side to left side
+=	a += 1 is same as a = a + 1
-=	a -= 1 is same as a = a - 1
*=	a *= 1 is same as a = a * 1
/=	a /= 1 is same as a = a / 1
%=	a %= 1 is same as a = a % 1

```
a=1;
b=a;
```

Increment Decrement Operator



Increment (++) operator used to increase the value of the variable by one.

Decrement (--) operator used to decrease the value of the variable by one.

Example

```
x=100;
x++;
```

Explanation

```
After the execution the
value of x will be 101.
```

Example

```
x=100;
x--;
```

Explanation

```
After the execution the
value of x will be 99.
```

Increment Decrement Operator

- Pre increment operator

x=10;

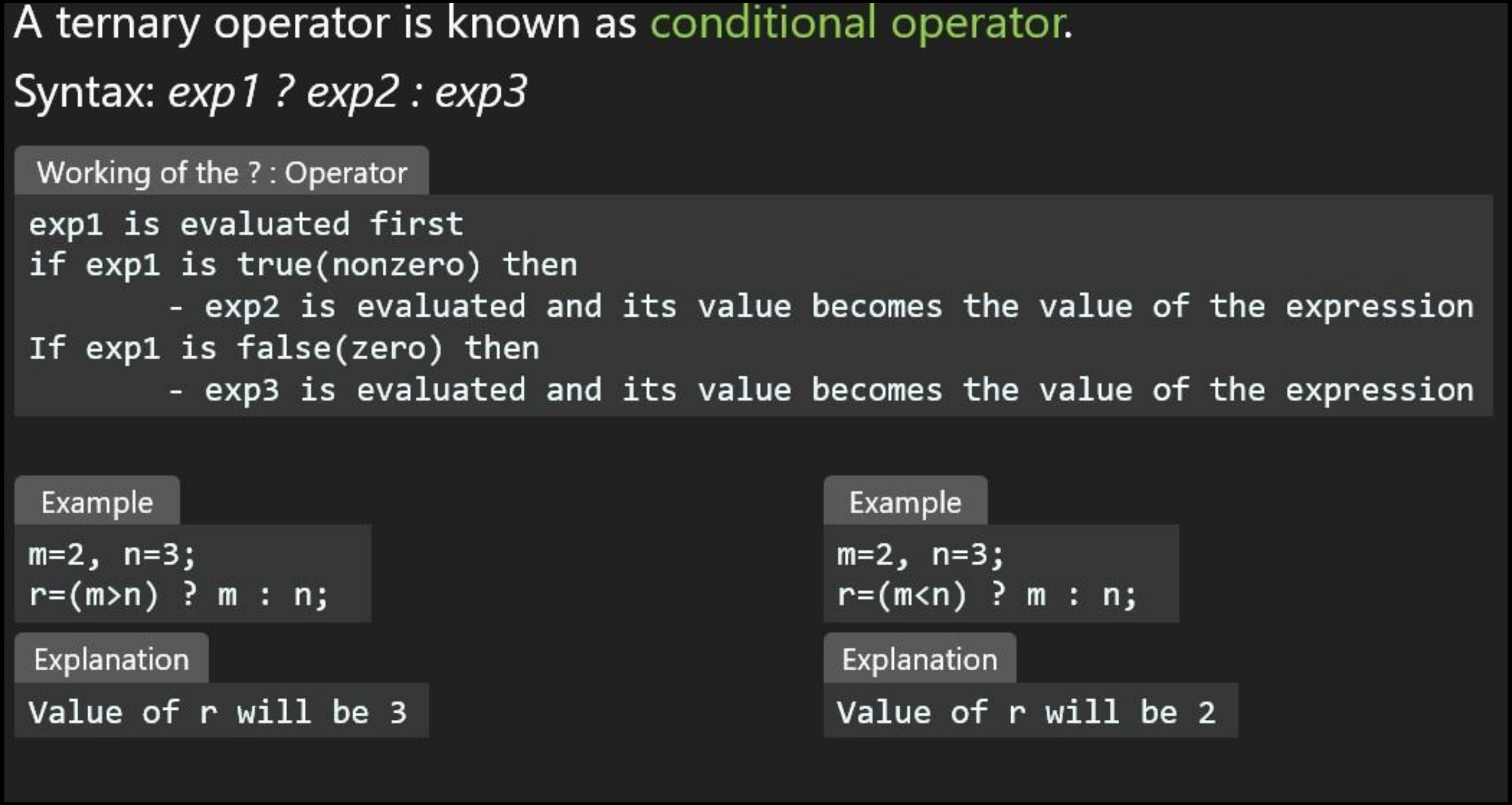
p=++x;

Post Increment Operator

x=10;

P=x++;

Conditional Operators



A ternary operator is known as conditional operator.

Syntax: *exp1 ? exp2 : exp3*

Working of the ? : Operator

```
exp1 is evaluated first
if exp1 is true(nonzero) then
       - exp2 is evaluated and its value becomes the value of the expression
If exp1 is false(zero) then
       - exp3 is evaluated and its value becomes the value of the expression
```

Example

```
m=2, n=3;
r=(m>n) ? m : n;
```

Explanation

```
Value of r will be 3
```

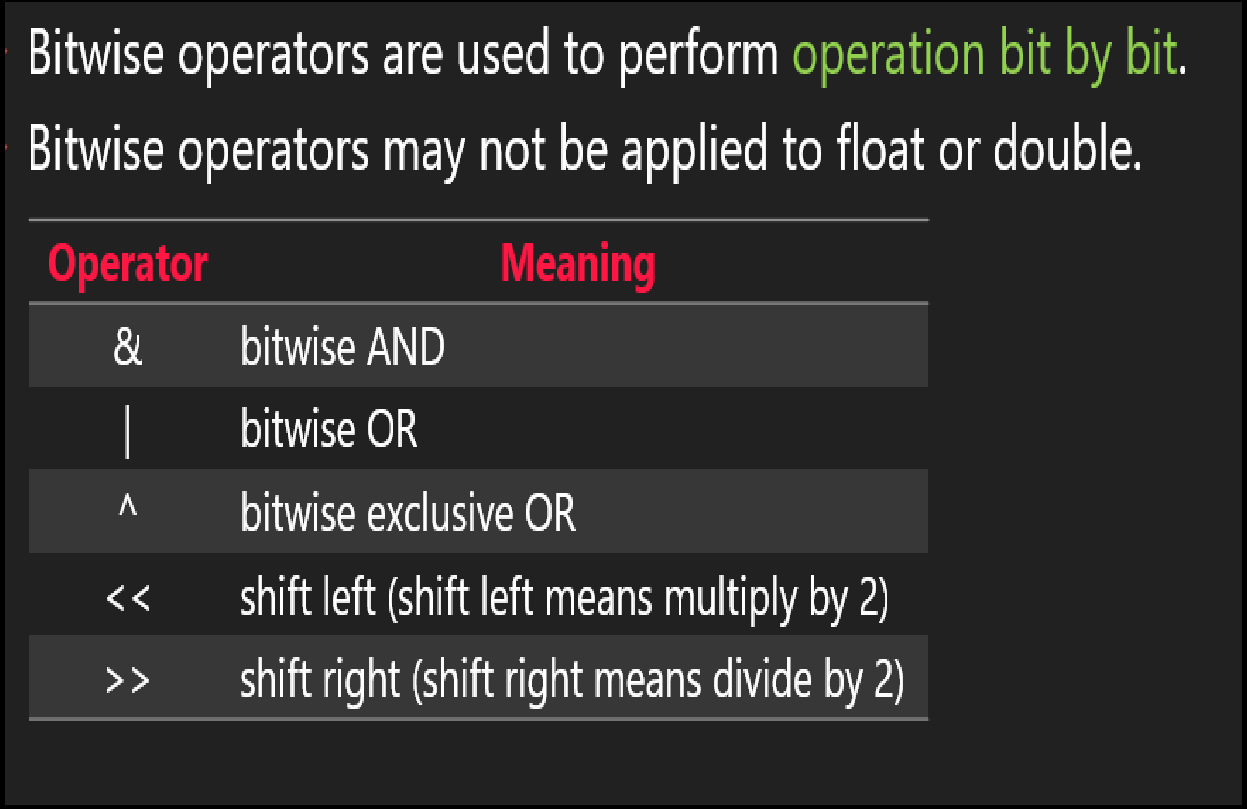
Example

```
m=2, n=3;
r=(m<n) ? m : n;
```

Explanation

```
Value of r will be 2
```

Bitwise Operators



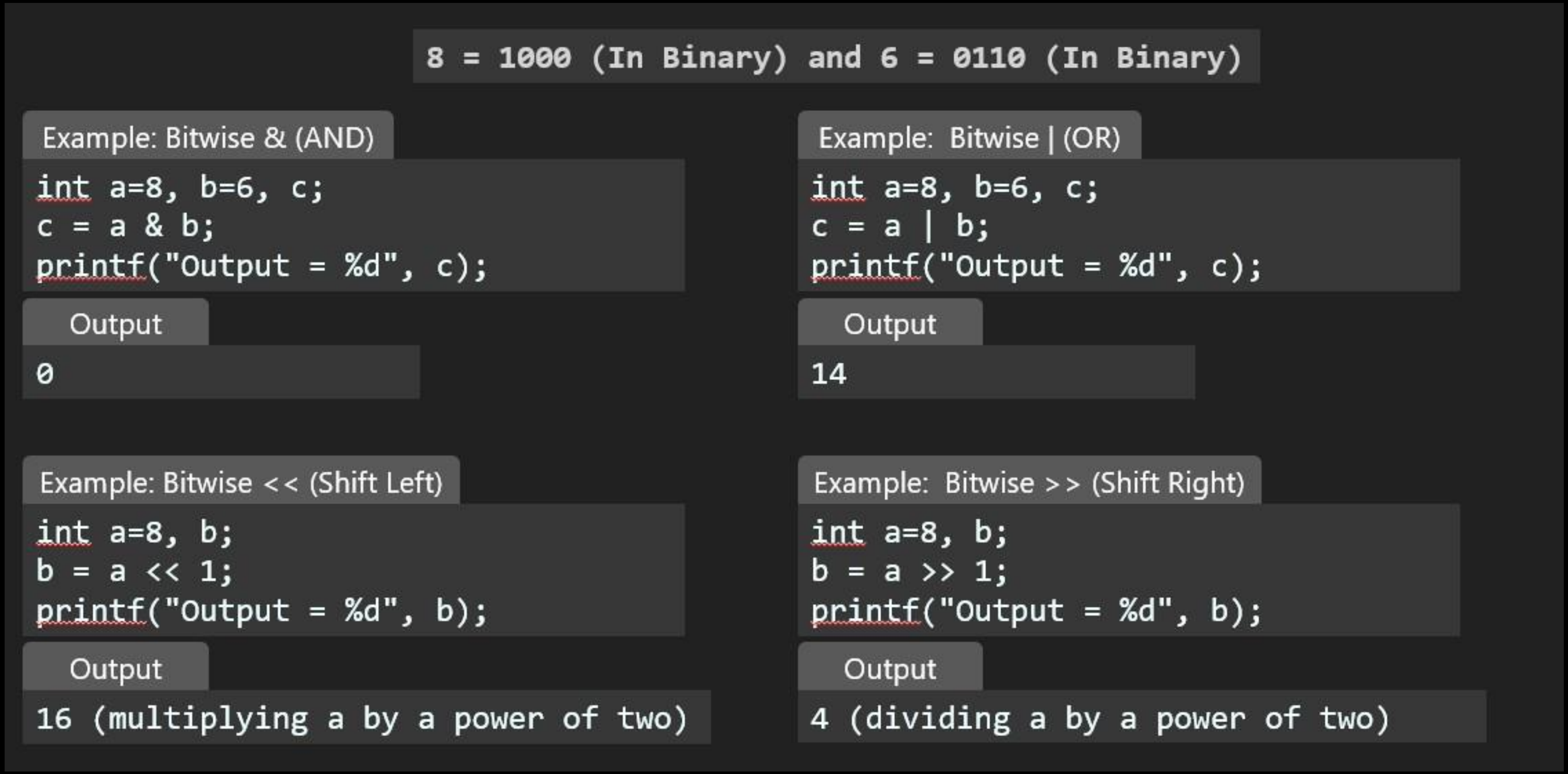
Bitwise operators are used to perform operation bit by bit.

Bitwise operators may not be applied to float or double.

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left (shift left means multiply by 2)
>>	shift right (shift right means divide by 2)

A	b	A&b	A|b	A^b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Bitwise Operators



```
8 = 1000 (In Binary) and 6 = 0110 (In Binary)
```

Example: Bitwise & (AND)

```c
int a=8, b=6, c;
c = a & b;
printf("Output = %d", c);
```

Output

```
0
```

Example: Bitwise | (OR)

```c
int a=8, b=6, c;
c = a | b;
printf("Output = %d", c);
```

Output

```
14
```

Example: Bitwise << (Shift Left)

```c
int a=8, b;
b = a << 1;
printf("Output = %d", b);
```

Output

```
16 (multiplying a by a power of two)
```

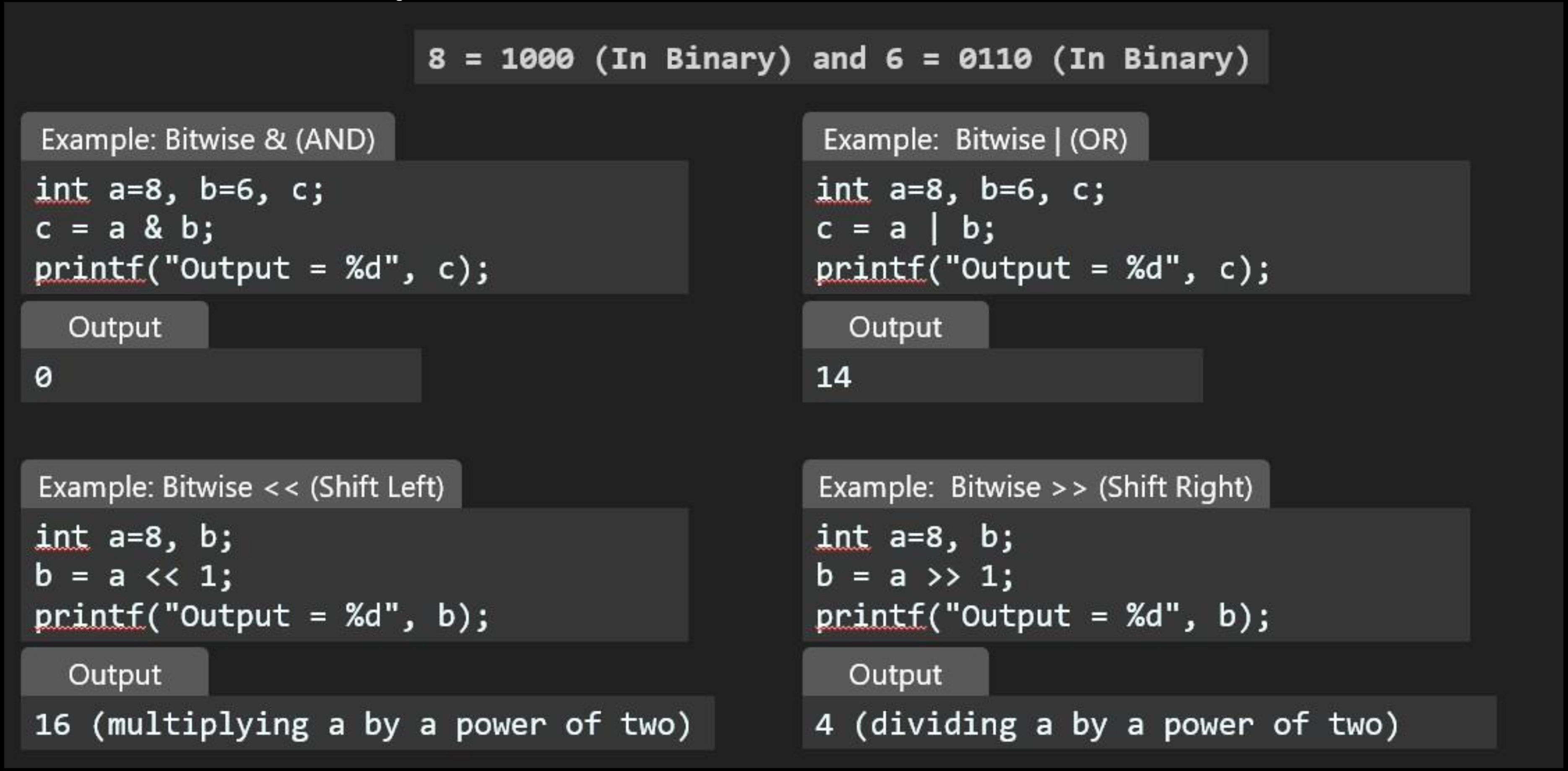
Example: Bitwise >> (Shift Right)

```c
int a=8, b;
b = a >> 1;
printf("Output = %d", b);
```

Output

```
4 (dividing a by a power of two)
```

Bitwise Operators



```
8 = 1000 (In Binary) and 6 = 0110 (In Binary)
```

Example: Bitwise & (AND)

```
int a=8, b=6, c;
c = a & b;
printf("Output = %d", c);
```

Output

```
0
```

Example: Bitwise | (OR)

```
int a=8, b=6, c;
c = a | b;
printf("Output = %d", c);
```

Output

```
14
```

Example: Bitwise << (Shift Left)

```
int a=8, b;
b = a << 1;
printf("Output = %d", b);
```

Output

```
16 (multiplying a by a power of two)
```

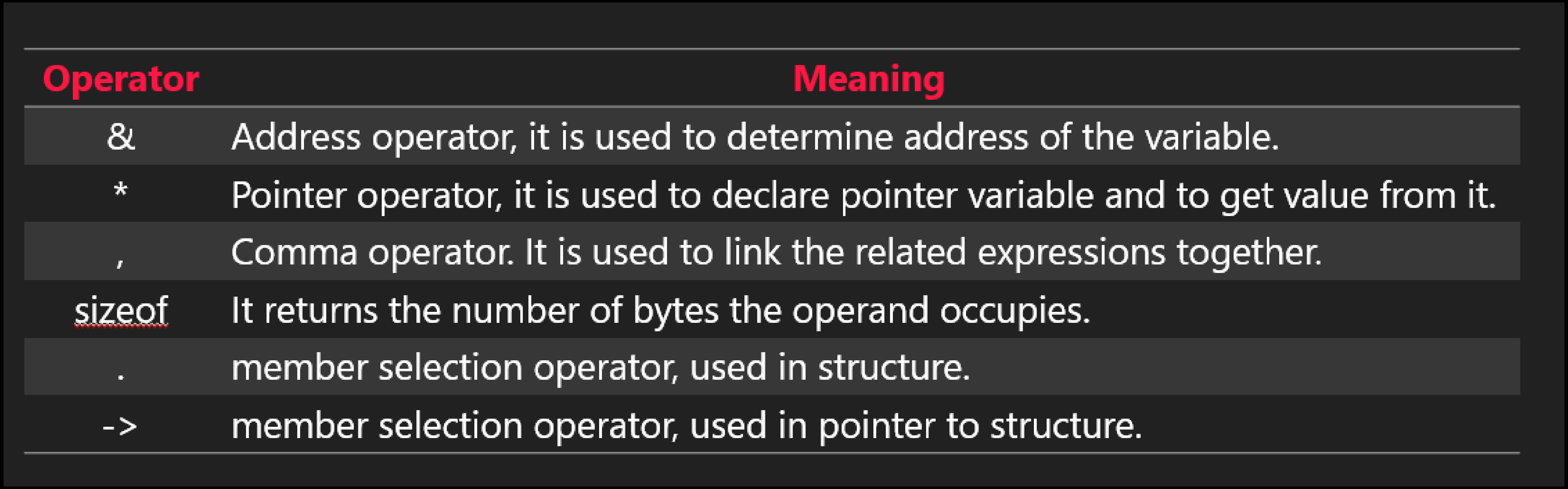
Example: Bitwise >> (Shift Right)

```
int a=8, b;
b = a >> 1;
printf("Output = %d", b);
```

Output

```
4 (dividing a by a power of two)
```

Special Operators

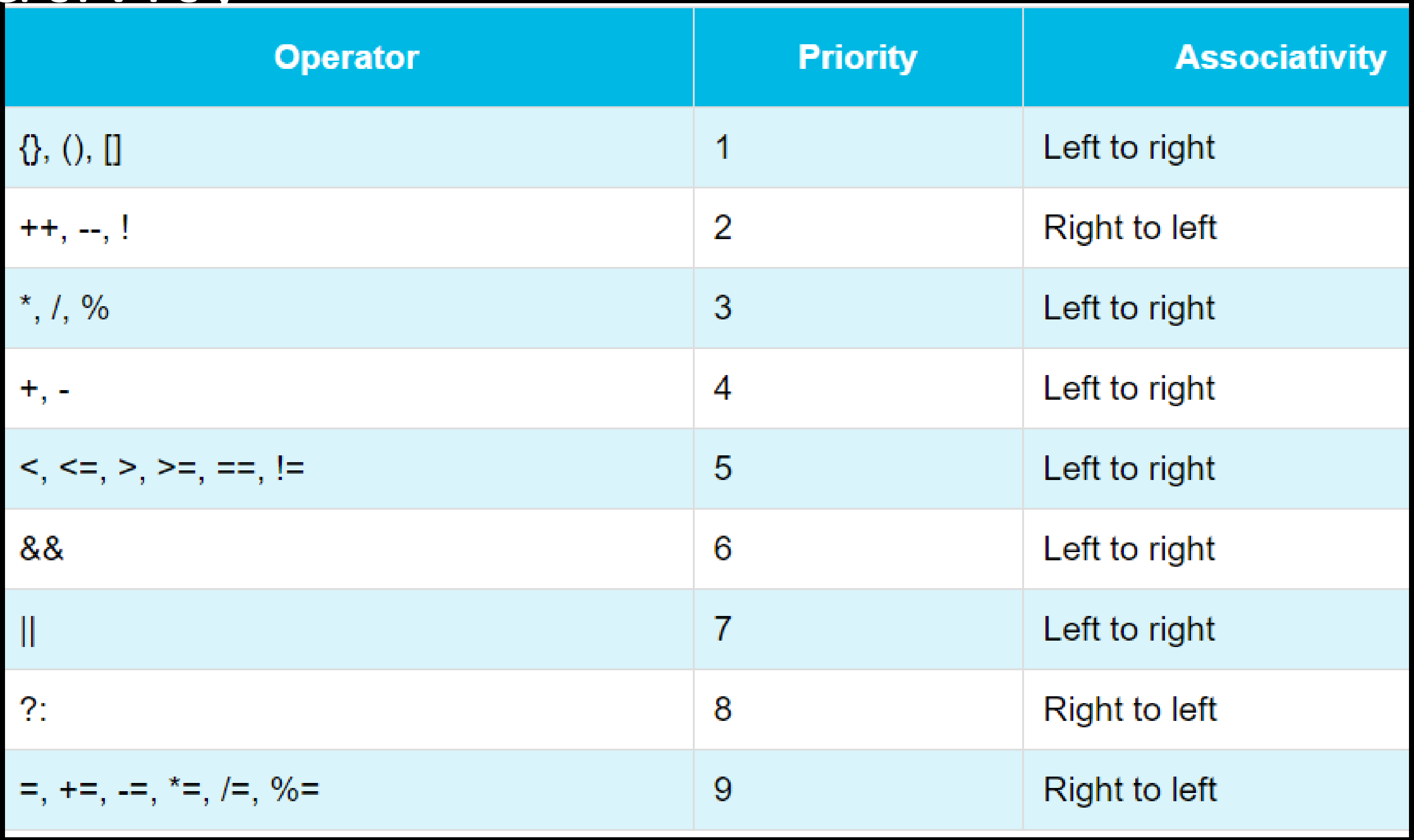


Operator	Meaning
&	Address operator, it is used to determine address of the variable.
*	Pointer operator, it is used to declare pointer variable and to get value from it.
,	Comma operator. It is used to link the related expressions together.
sizeof	It returns the number of bytes the operand occupies.
.	member selection operator, used in structure.
->	member selection operator, used in pointer to structure.

Evaluation of Expression in C

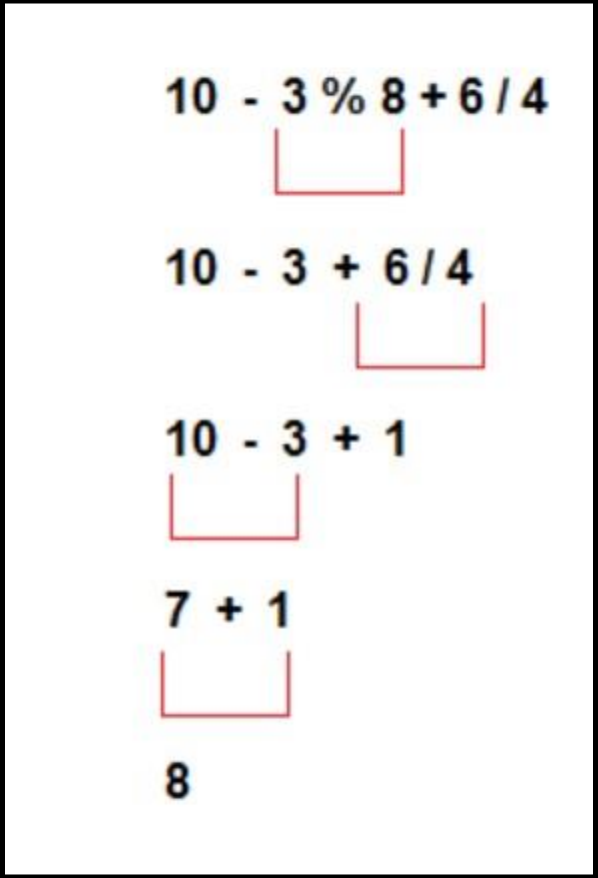
- An expression is a sequence of operands and operators that reduces to a single value. Example-12+21
- An expression is a combination of variables ,constants and operators written according to the syntax of C language.
- Evaluation of expression depends on priority and associativity.
- Priority-This represents the evaluation of expression starts from "which" operator.
- **Associativity-**It represents which operator should be evaluated first if an expression is containing more than one operator with same priority.

Operators-Priority and Associativity



Operator	Priority	Associativity
{}, (), []	1	Left to right
++, --, !	2	Right to left
*, /, %	3	Left to right
+, -	4	Left to right
<, <=, >, >=, ==, !=	5	Left to right
&&	6	Left to right
||	7	Left to right
?:	8	Right to left
=, +=, -=, *=, /=, %=	9	Right to left

Evaluation of Expression in C



10 - 3 % 8 + 6 / 4

10 - 3 + 6 / 4

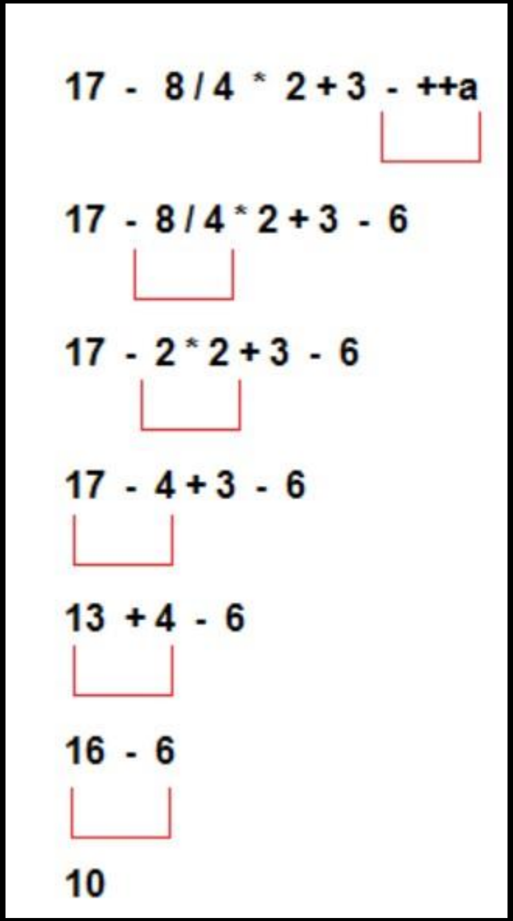
10 - 3 + 1

7 + 1

8

11-4*3+(4-1)/2
11-4*3+3/2
11-12+3/2
11-12+1
-1+1
0

Evaluation of Expression in C



```
17 - 8 / 4 * 2 + 3 - ++a

17 - 8 / 4 * 2 + 3 - 6

17 - 2 * 2 + 3 - 6

17 - 4 + 3 - 6

13 + 4 - 6

16 - 6

10
```

Type Conversion/Type Casting

- The **type conversion in C**- Converting one type of data, type to other to perform arithmetic/logical operation.
- The conversion is done only between those datatypes wherein the conversion is possible ex – char to int and vice versa.
- Types of Type conversion
  - Implicit type conversion
  - Explicit type conversion

Implicit Type Conversion

- This type of conversion is performed by the compiler when necessary without any commands by the programmer. It is also called **"Automatic Type Conversion"**.
- The compiler usually performs this type of conversion when a particular expression contains more than one data type.

Implicit Type Conversion

- Rules

1. `char` or `short` type operands will be converted to `int` during an operation and the outcome data type will also be `int`.
2. If an operand of type `long double` is present in the expression, then the corresponding operand will also be converted to `long double` same for the `double` data type.
3. If an operand of `float` type is present then the corresponding operand in the expression will also be converted to `float` type and the final result will also be `float` type.
4. If an operand of `unsigned long int` is present then the other operand will be converted to `unsigned long int` and the final result will also be `unsigned long int`.
5. If an operand of `long int` is present then the other operand will be converted to `long int` and the final result will also be `long int`.
6. If an operand of `unsigned int` is present then the other operand will be converted to `unsigned int` and the final result will also be `unsigned int`.

Implicit Type Conversion

- Example-Rule 2

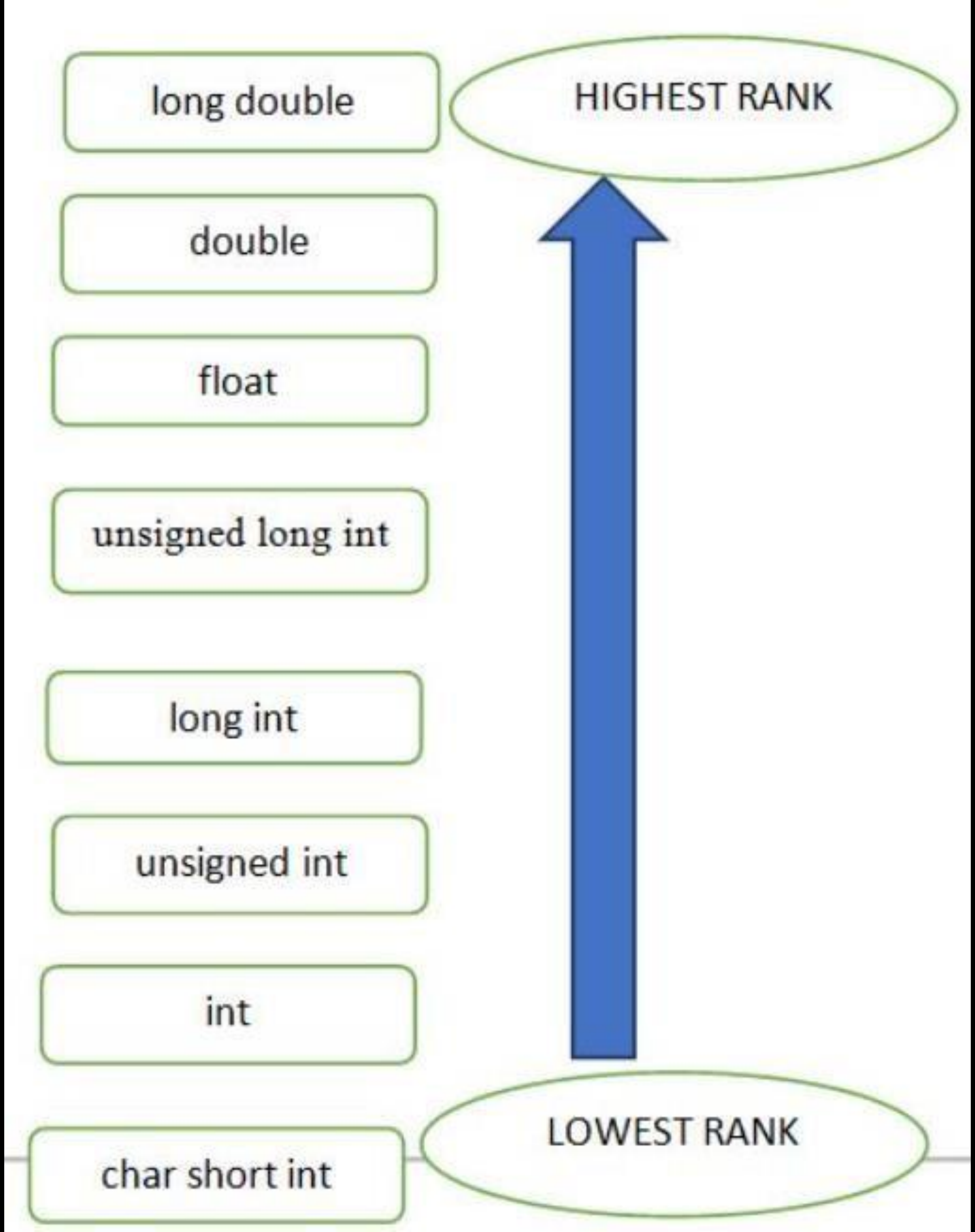
```
int a = 20;
double b = 20.5;
a + b;
```

Example-Rule 1

```
char ch='a';
 int a =13;
 a + c;
```

Example 3- Rule 6

```
char ch='A';
 unsigned int a =60;
 a * b;
```



long double

HIGHEST RANK

double

float

unsigned long int

long int

unsigned int

int

LOWEST RANK

char short int

Explicit Type Conversion

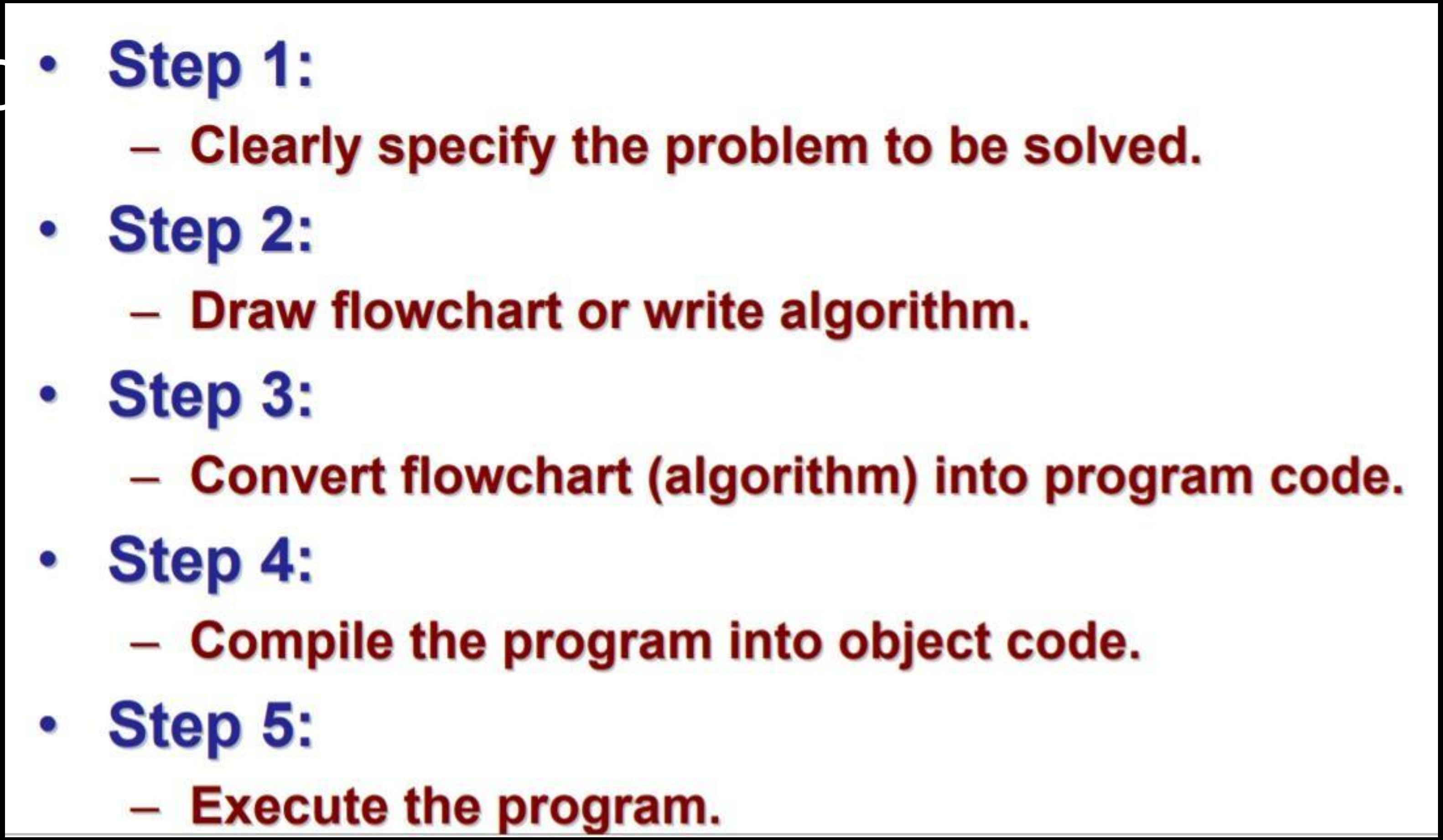
- The process of converting one data type data into another data type according to user requirement is called explicit type conversion.
- This is done by programmer .

Syntax:
type(expression)

Explicit type Conversion

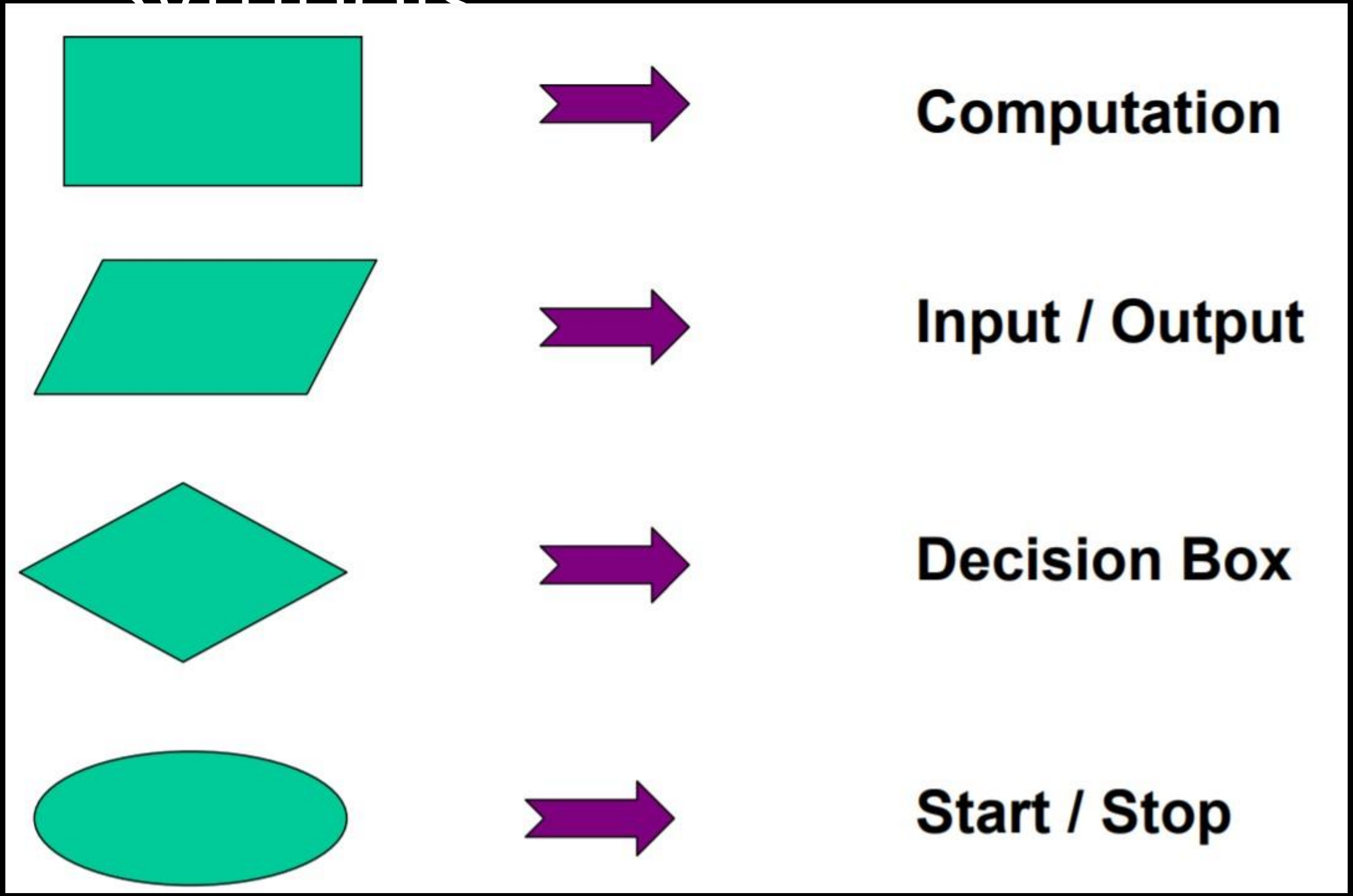
```
#include<stdio.h>
int main()
{
        float a = 1.2;
        //int b  = a; //Compiler will throw an error for this
        int b = (int)a + 1; //type casting-explicit type casting
        printf("Value of a is %f\n", a);
        printf("Value of b is %d\n",b);
        return 0;
}
```

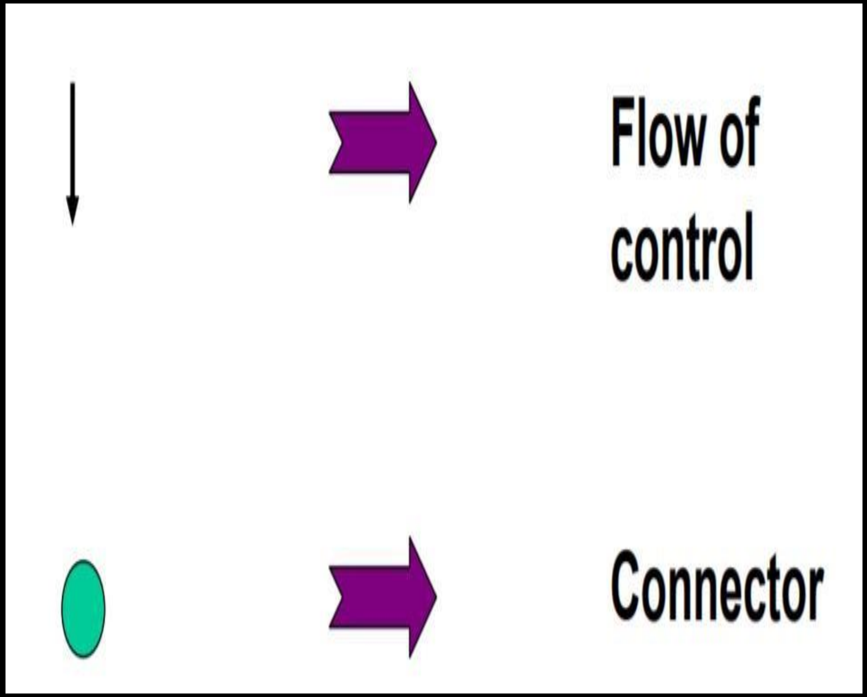
Problem Solving Approach-Flowchart and



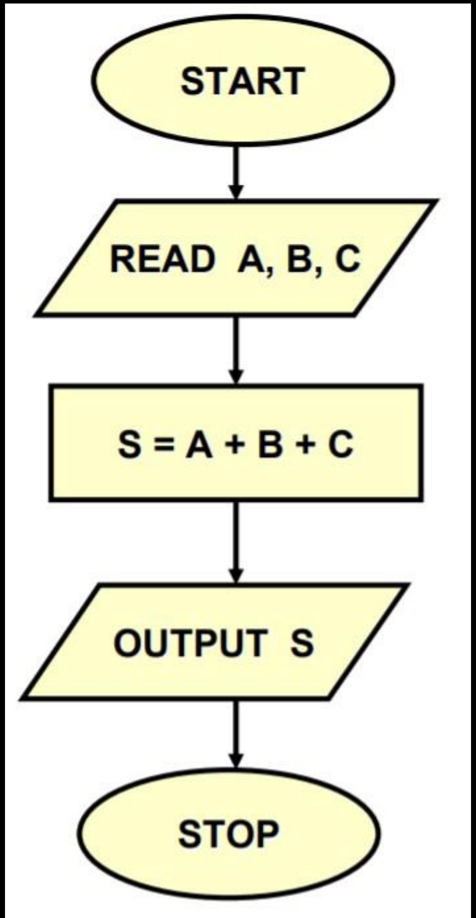
- **Step 1:**
  - **Clearly specify the problem to be solved.**
- **Step 2:**
  - **Draw flowchart or write algorithm.**
- **Step 3:**
  - **Convert flowchart (algorithm) into program code.**
- **Step 4:**
  - **Compile the program into object code.**
- **Step 5:**
  - **Execute the program.**

Flow chart Basic Symbols

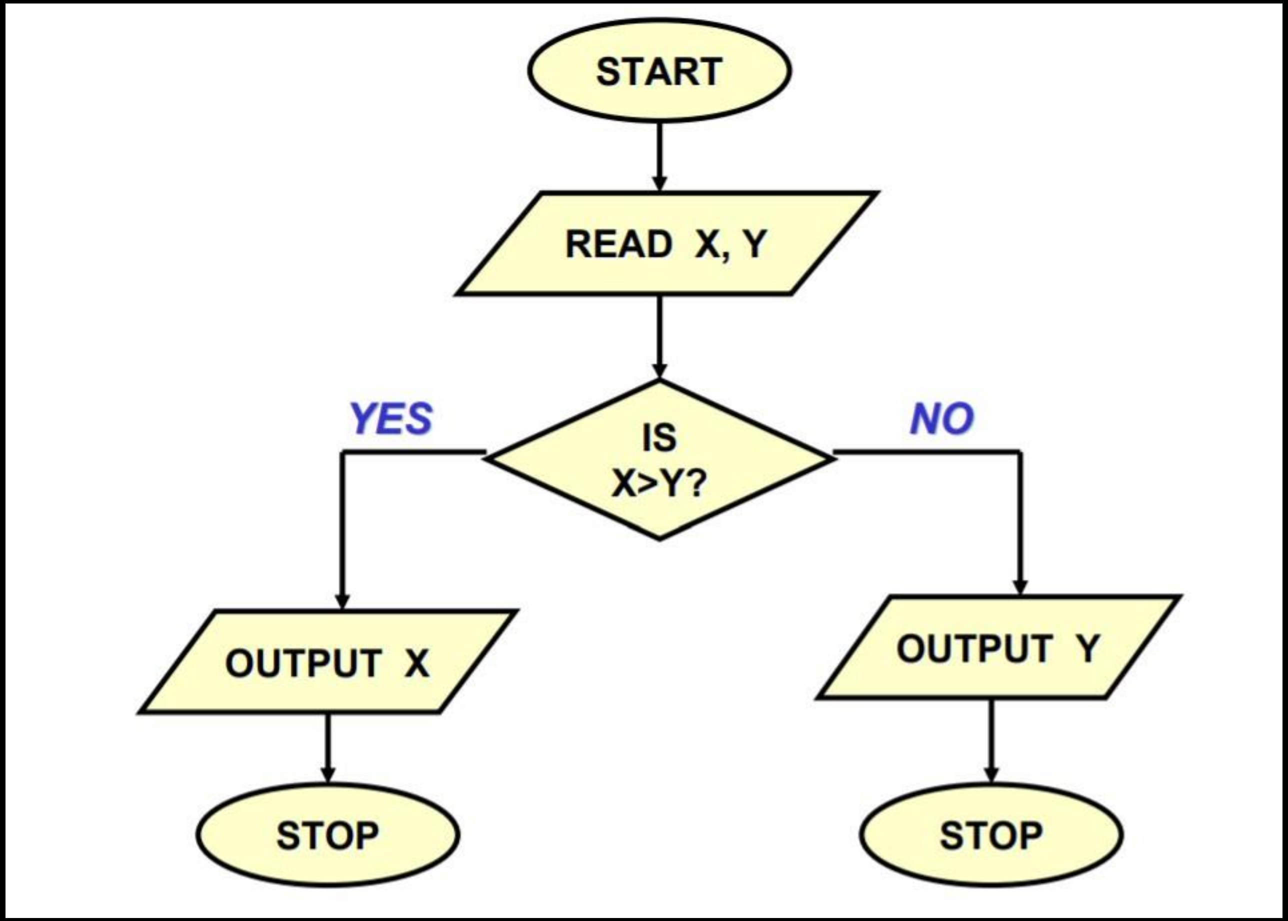




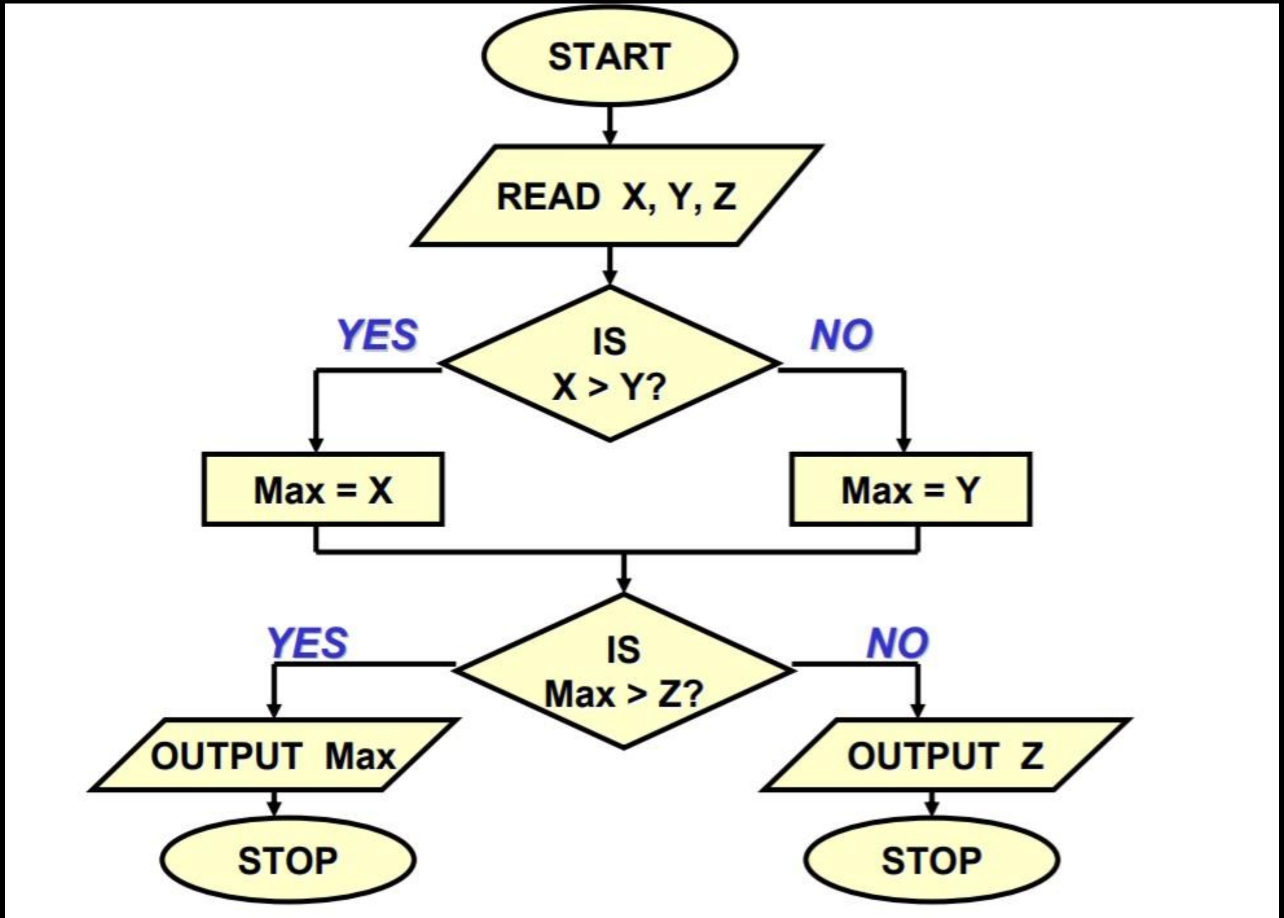
Addition of three numbers



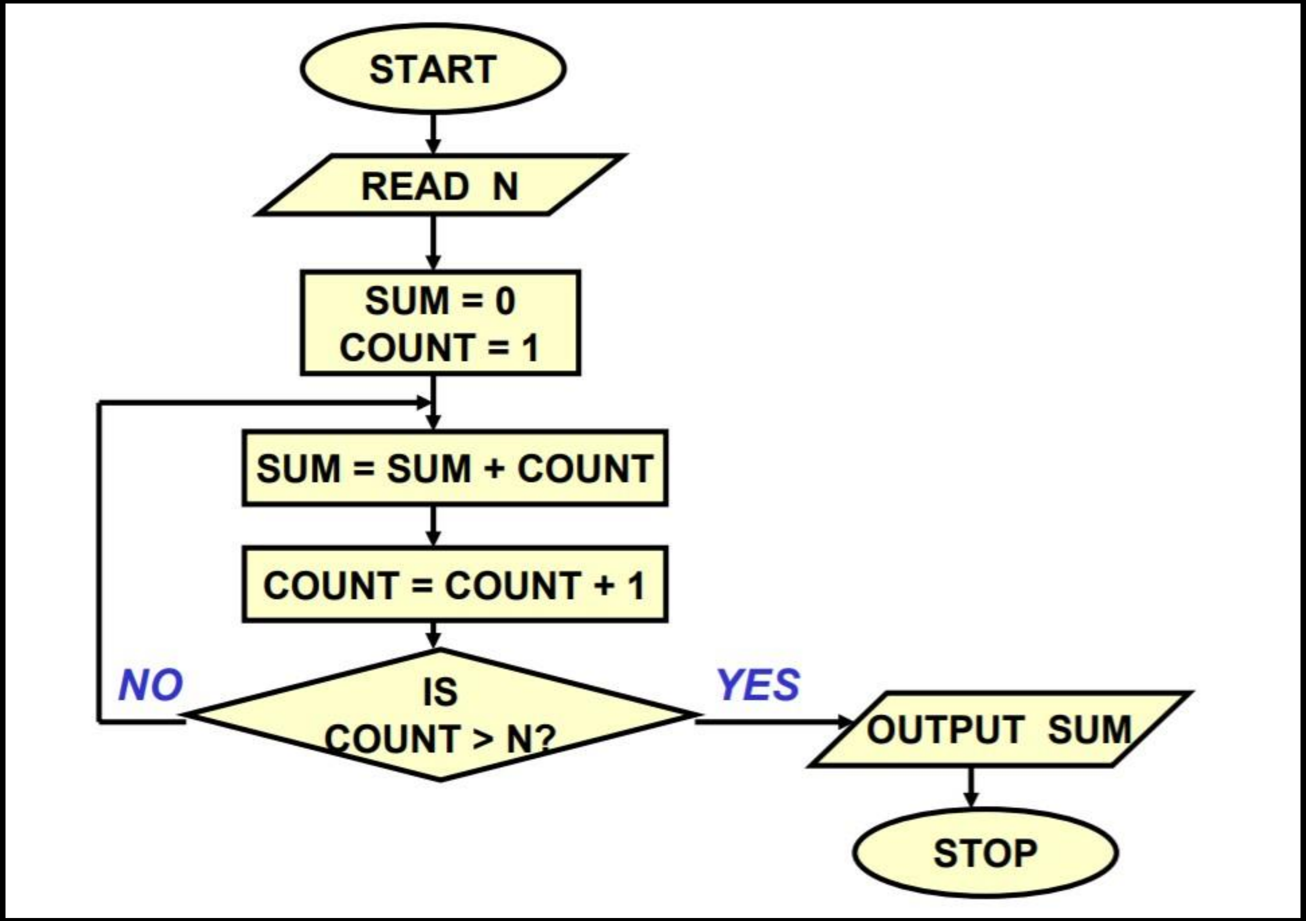
Larger of Two numbers



Largest of three numbers



Sum of first N natural numbers



Algorithm for finding factorial of a number

- Step 1: Start
- Step 2: Read a number as N for finding its factorial.
- Step 3: Fact=1,Count=1
- Step 4: Fact=fact*count
- Step 5: Count=count+1
- Step 6: Check if count <= N then goto step 4 else step 7
- Step 7: Print factorial value in Fact variable
- Step 8: Stop