Language

Radha Ranpara.

Introduction of computer languages

- Natural languages, such as English, are ambiguous, fuzzily structured and have large (and changing) vocabularies. Computers have no common sense, so computer languages must be very precise they have relatively few, exactly defined, rules for composition of programs, and strictly controlled vocabularies in which unknown words must be defined before they can be used. It is a major goal of research in Artificial Intelligence to find out how to make computers understand natural languages, and the more we learn, the harder it seems to be!

- [Computer languages can be divided into two groups: high-level languages and low-level languages.] High-level languages are designed to be easier to use, more abstract, and more portable than low-level languages. Syntactically correct programs in some languages are then compiled to low-level language and executed by the computer.[Most modern software is written in a high-level language, compiled into object code, and then translated into machine instructions.]

- [The description of a programming language is usually split into the two components of syntax (form) and semantics (meaning). Some languages are defined by a specification document (for example, the C programming language is specified by an ISO Standard), while other languages, such as Perl 5 and earlier, have a dominant implementation that is used as a reference.]

-

Introduction of programming concept

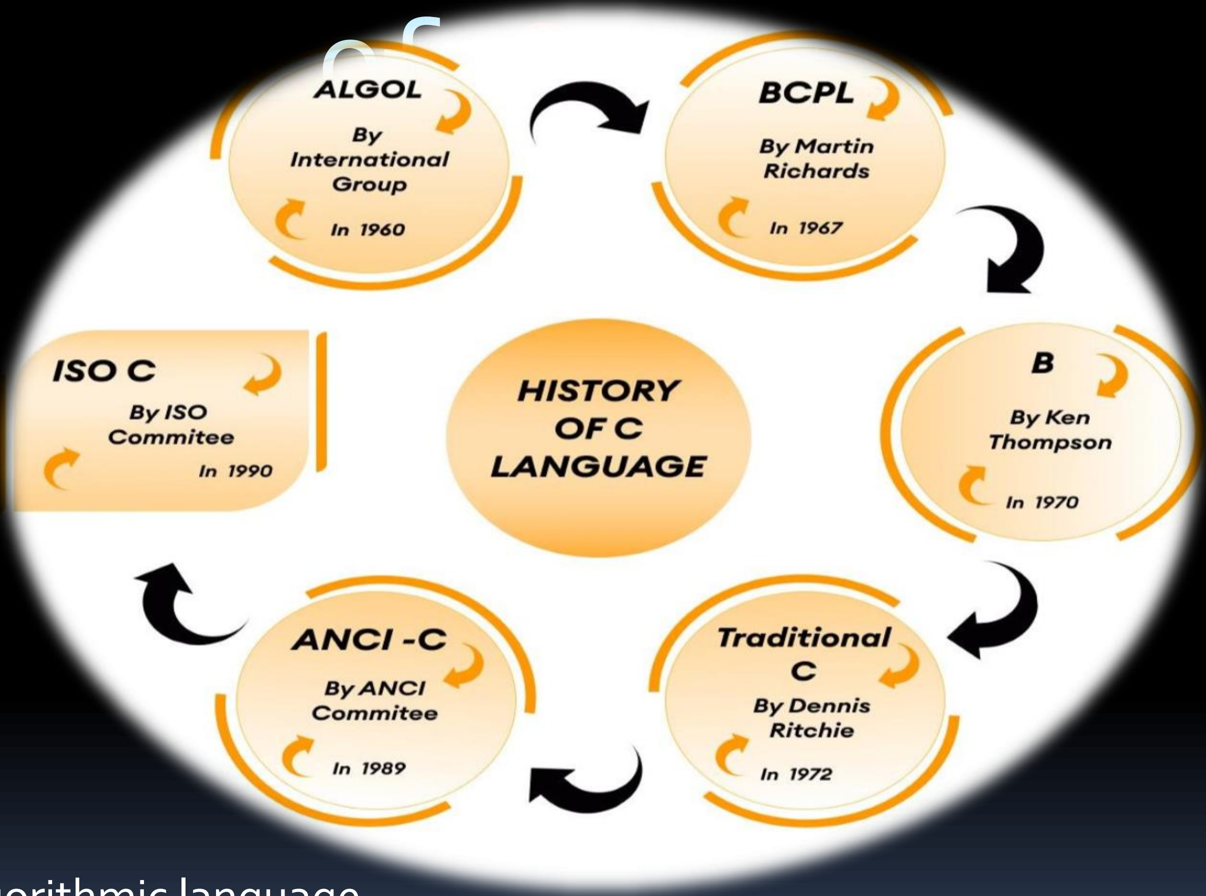
- Computer programs are collections of [instructions that tell a computer how to interact with the user, interact with the computer hardware and process data.] The first programmable computers required the programmers to write explicit instructions to directly manipulate the hardware of the computer. This "machine language" was very tedious to write by hand since even simple tasks such as printing some output on the screen require 10 or 20 machine language commands. Machine language is often referred to as a "low level language" since the code directly manipulates the hardware of the computer.

- By contrast, higher level languages such as "C", C++, Pascal, Cobol, Fortran, ADA and Java are called "compiled languages". In a compiled language, the programmer writes more general instructions and a compiler (a special piece of software) automatically translates these high level instructions into machine language. The machine language is then executed by the computer. A large portion of software in use today is programmed in this fashion.

- Think about some of the different ways that people use computers. In school, students use computers for tasks such as writing papers, searching for articles, sending email, and participating in online classes. At work, people use computers to analyze data, make presentations, conduct business transactions, communicate with customers and co workers control machines in manufacturing facilities, and do many other things. At home, people use computers for tasks such as paying bills, shopping online, communicating with friends and family, and playing computer games. And don't forget that cell phones, iPods, BlackBerries, car navigation systems, and many other devices are computers too. The uses of computers are almost limitless in our everyday lives.

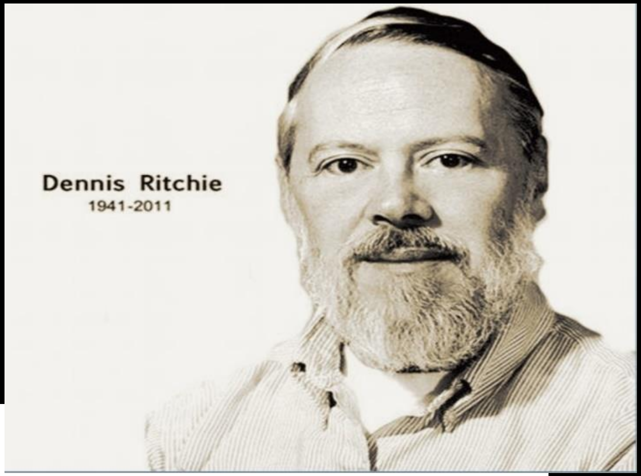
- Computer programming is really just about solving problems. It turns out that a large number of the problems you encounter in the real world are really just special cases of a more general problem.

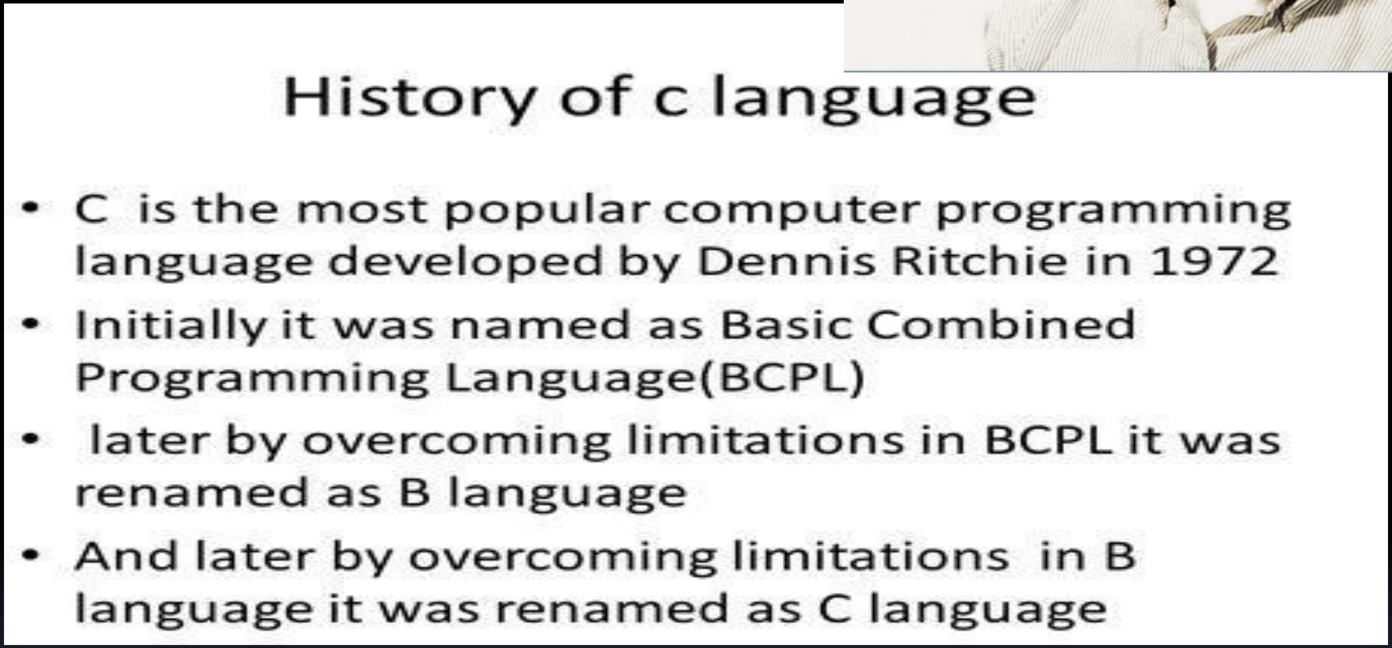
History of



ALGOL: algorithmic language.
BCPL : Basic Combined Programming Language.
ANCI : American National Standards Institute.
ISO : International Organization for Standardization.

- C was developed by Dennis ritchie at AT & T’s Bell laboratories in 1972.
- In 1960, the first programming language ALGOL(algorithmic language) is developed by International group.it gives the concept of structured programming.
- In 1967,Martin Richards developed a language called BCPL(Basic Combined Programming Language)primarily for writing system software.
- In 1970,Ken Thompson created a language using many feactures of BCPL and called it simply B. both BCPL and B were “typeless” system programming languages.





History of c language

- C is the most popular computer programming language developed by Dennis Ritchie in 1972
- Initially it was named as Basic Combined Programming Language(BCPL)
- later by overcoming limitations in BCPL it was renamed as B language
- And later by overcoming limitations in B language it was renamed as C language

Importance of C Language

- C is called as a robust language, which has so many built-in functions and operations, which can be used to write any complex program.
- Generally, we use to call C as a middle level language. Because, the ‘C’ compiler combines the capabilities of an assembly language with the features of a high-level language. Therefore, it is best for writing both system software and business packages.
- ‘C’ Programs are efficient and fast.
- C is highly portable, that is, ‘C’ programs written on one computer can be run on another with little (or) no modification.
- ‘C’ language is best for structured programming, where the user can think of a problem in terms of function modules (or) blocks.
- It has the ability to extend itself.

- Mother language
- System programming language
- Procedure-oriented programming language
- Structured programming language
- Mid-level programming language

1) C as a mother language

- C language is considered as the mother language of all the modern programming languages because **most of the compilers, JVMs, Kernels, etc. are written in C language**, and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.
- It provides the core concepts like the array, strings, functions, file handling, etc. that are being used in many languages like C++, Java, C#, etc.

2) C as a system programming language

- A system programming language is used to create system software. C language is a system programming language because it **can be used to do low-level programming (for example driver and kernel)**. It is generally used to create hardware devices, OS, drivers, kernels, etc. For example, Linux kernel is written in C.
- It can't be used for internet programming like Java, .Net, PHP, etc.

3) C as a procedural language

- A procedure is known as a function, method, routine, subroutine, etc. A procedural language **specifies a series of steps for the program to solve the problem**.
- A procedural language breaks the program into functions, data structures, etc.
- C is a procedural language. In C, variables and function prototypes must be declared before being used.

4) C as a structured programming language

- A structured programming language is a subset of the procedural language. **Structure means to break a program into parts or blocks** so that it may be easy to understand.
- In the C language, we break the program into parts using functions. It makes the program easier to understand and modify.

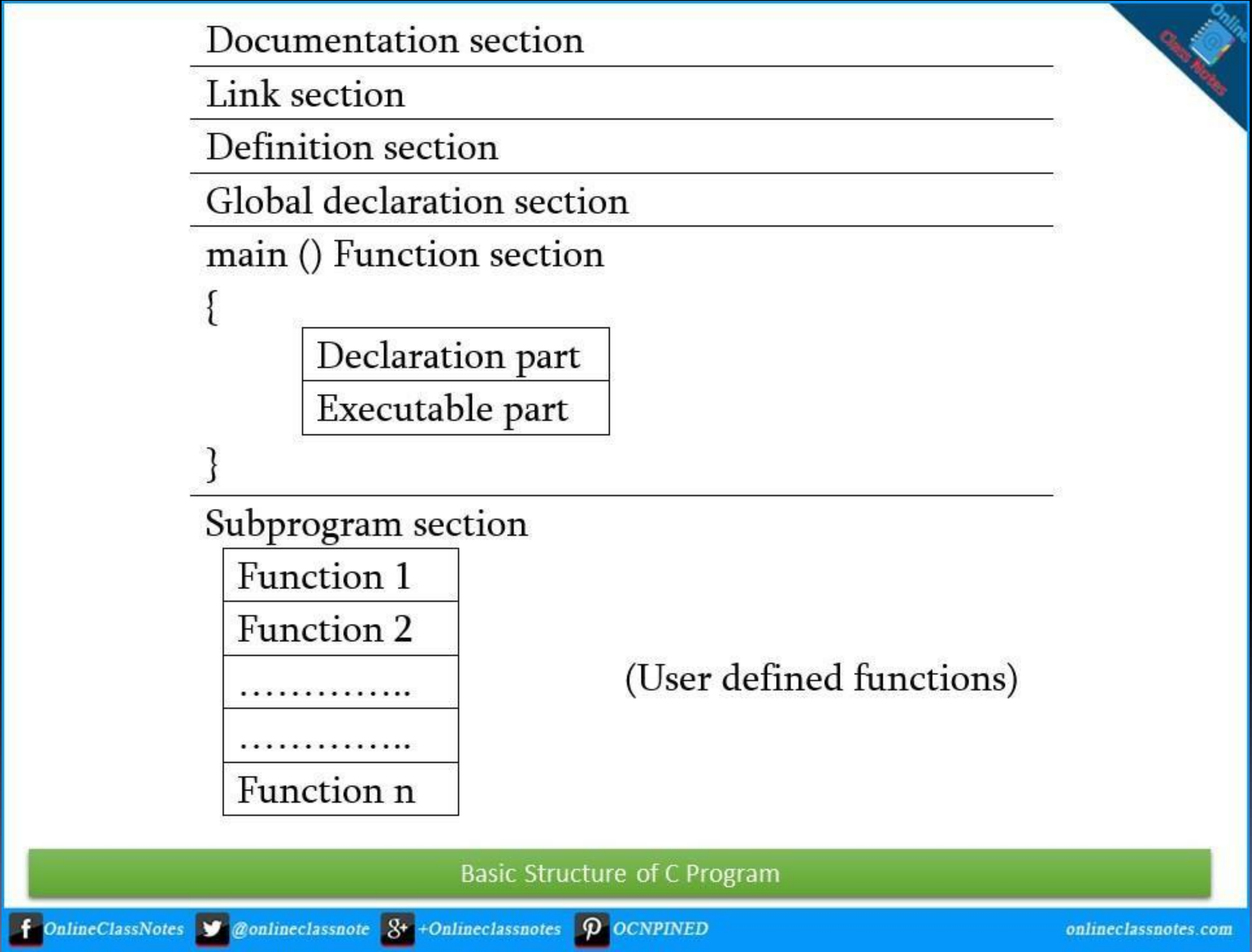
5) C as a mid-level programming language

- C is considered as a middle-level language because it **supports the feature of both low-level and high-level languages**. C language program is converted into assembly code, it supports pointer arithmetic (low-level), but it is machine independent (a feature of high-level).
- A **Low-level language** is specific to one machine, i.e., machine dependent. It is machine dependent, fast to run. But it is not easy to understand.
- A **High-Level language** is not specific to one machine, i.e., machine independent. It is easy to understand.

Features of C Language

- ➢Modularity
- ➢Extensibility
- ➢Elegant syntax
- ➢Case sensitive
- ➢Less memory required
- ➢The standard library concept
- ➢A powerful and varied range of operators

Basic structure of C



Documentation section

Link section

Definition section

Global declaration section

main () Function section

{

Declaration part
Executable part

}

Subprogram section

Function 1
Function 2
…………..
…………..
Function n

(User defined functions)

Basic Structure of C Program

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)

Program

```
// Program for addition of 2 nos

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

void fun();

int a=10;


void main( )
{
Printf ("Value of a inside main function: %d", a);
fun();
}

void fun()
{
    printf("Value of a inside fun function: %d", a);
}
```

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-line comment.
Example: /* multi line comment line -1
multi line comment line -2 */

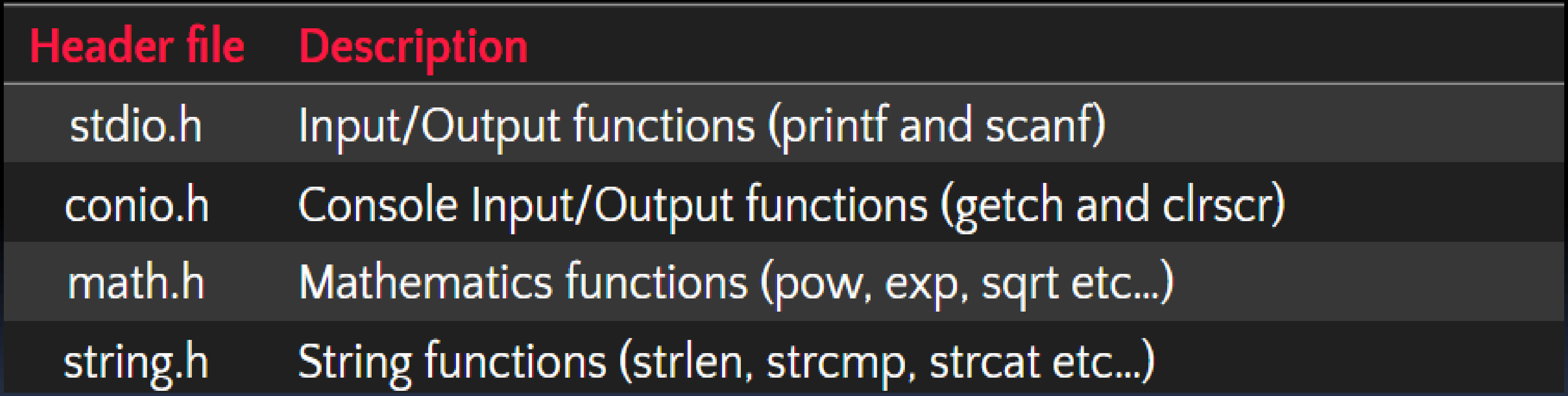
Header files / Link

All header files are included in this section.

- A header file is a file that consists of C declarations that can be used between different files. It helps us in using others' code in our files. A copy of these header files is inserted into your code before compilation.
- #include <stdio.h>

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...)

This function does not take any parameters. Here, getch() **returns the ASCII value of the character read from stdin** . For example, if we give the character 'o' as input, it will return the ASCII value of 'o', which is 49. Now, in C / C++, we can directly convert a character to an integer.

Definition

- This section is not necessary for all programs. But whenever we need to declare some symbolic or numeric constants. A preprocessor directive in C is any statement that begins with the "#" symbol. For example we need to declare a constant PI which is having a fix value of 3.14 so we can declare it as following
- **#define PI 3.14**
- With the help of this , the value of PI is defined 3.14 and would never be changed in the program by any means.
- define statement **does not** ends with a semicolon.

Global Declaration

- This section includes all global variables, function declarations, and static variables. The variables declared in this section can be used anywhere in the program. They're accessible to all the functions of the program. Hence, they are called global variables.
- int age(int current);
- We've declared our age function, which takes one integer argument and returns an integer.

- **Main() Function**
- In the structure of a C program, this section contains the main function of the code. The compiler starts execution from the main() function. It can use global variables, static variables, inbuilt functions, and user-defined functions. The return type of the main() function can be void also not necessarily int.
- Void main ()

```
// 1) Helloworld C program (document
section)
#include <stdio.h>
#include <conio.h>      //(link section)
      void main()  //(main function)
       {
       clrscr();
          printf("Hello world");
          getch();
          }
```

Difference between Traditional vs Modern C

	Traditional C	Modern C
Works	Traditional C works under only Unix based operating system and 16 bit computer platform.	Modern C (Turbo C) works under various operating systems and with some modifications it can work under various computer platforms.
Time- complexity	In Traditional C, only one program can run at a time	In Modern C, more than one program can run simultaneously.
Programming Practices	Traditional C programming often relied on certain practices that are now considered less safe or less efficient.	Modern C encourages the use of safer alternatives

Difference between Traditional vs Modern C

	Traditional C	Modern C
Example	traditional C commonly used functions like `` `gets()` `` for input, which is prone to buffer overflows.	Modern C encourages the use of 'fgets()` or functions from the `` `<stdint.h>` `` and `` `<stdbool.h>` `` headers for improved type safety and code readability.

Difference between Traditional vs Modern C

	Traditional C	Modern C
. Memory Management	Traditional C put the burden of memory management on the programmer, requiring explicit allocation and deallocation of memory using functions like `malloc()` and `free()`.	Modern C introduced new features like automatic storage duration (`auto` variables), variable-length arrays (VLAs), and enhanced support for dynamic memory allocation with functions like `calloc()` and `realloc()`. Additionally, modern C provides a more robust memory model, including concepts like `restrict` and atomic operations.

C character set

- Every language has some basic elements. Before understanding programming, it is must to know the basic elements of c language.
- “the character set is the basic elements of any language and they are used to represent information.”

- The characters that are used in c programs are given below:-

1. Alphabets:

   A,B,C,………….Z

   (uppercase)

   [65 to 90 ascii]

   a,b,c,………z

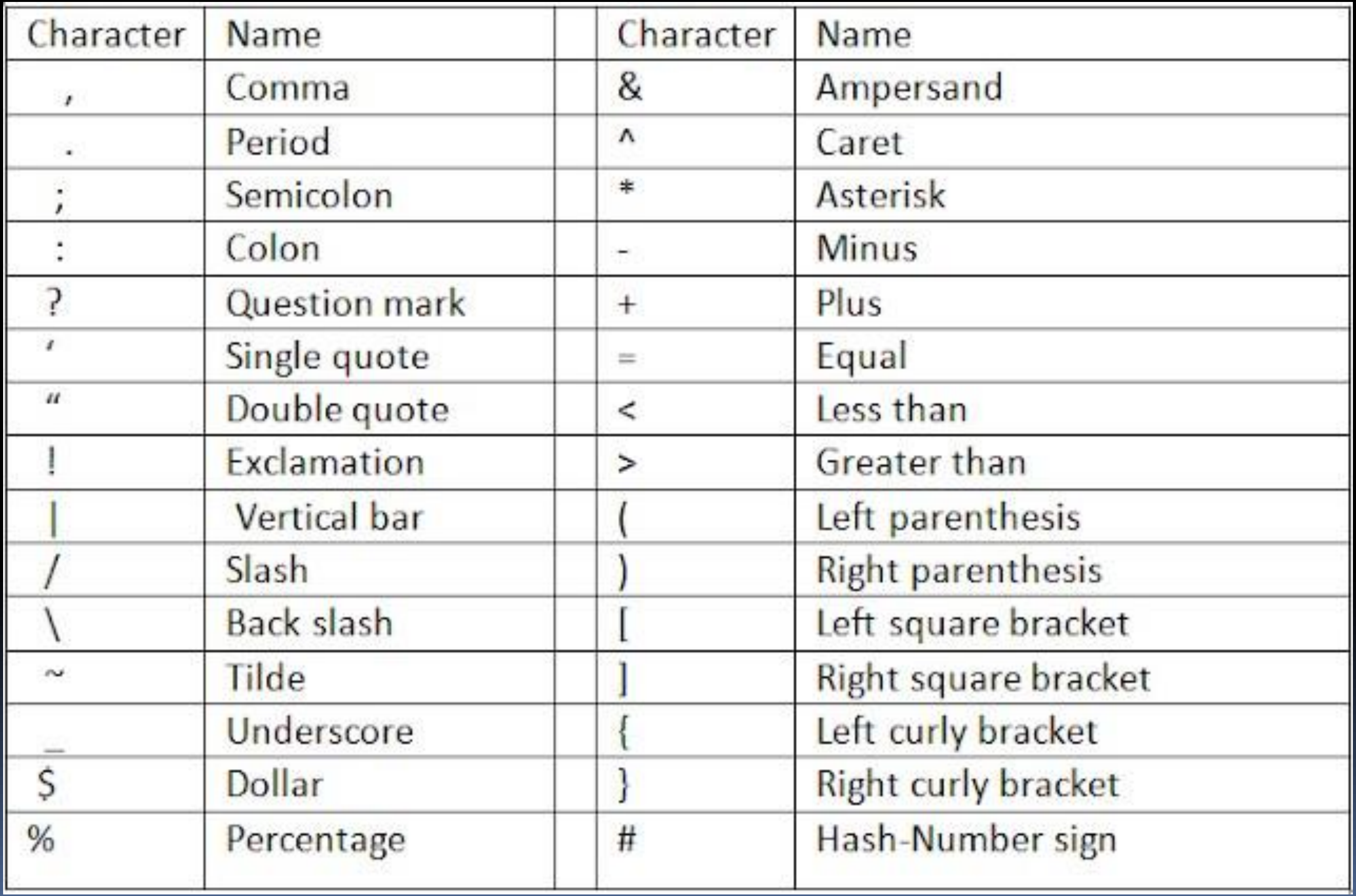
   (lowercase)

   [97 to 122]

2. Digits:

   0,1,2,……9 [48 to 57 ASCII]

3. Special character:-



Character	Name		Character	Name
,	Comma		&	Ampersand
.	Period		^	Caret
;	Semicolon		*	Asterisk
:	Colon		-	Minus
?	Question mark		+	Plus
‘	Single quote		=	Equal
“	Double quote		<	Less than
!	Exclamation		>	Greater than
|	Vertical bar		(	Left parenthesis
/	Slash		)	Right parenthesis
\	Back slash		[	Left square bracket
~	Tilde		]	Right square bracket
_	Underscore		{	Left curly bracket
$	Dollar		}	Right curly bracket
%	Percentage		#	Hash-Number sign

4. White space:

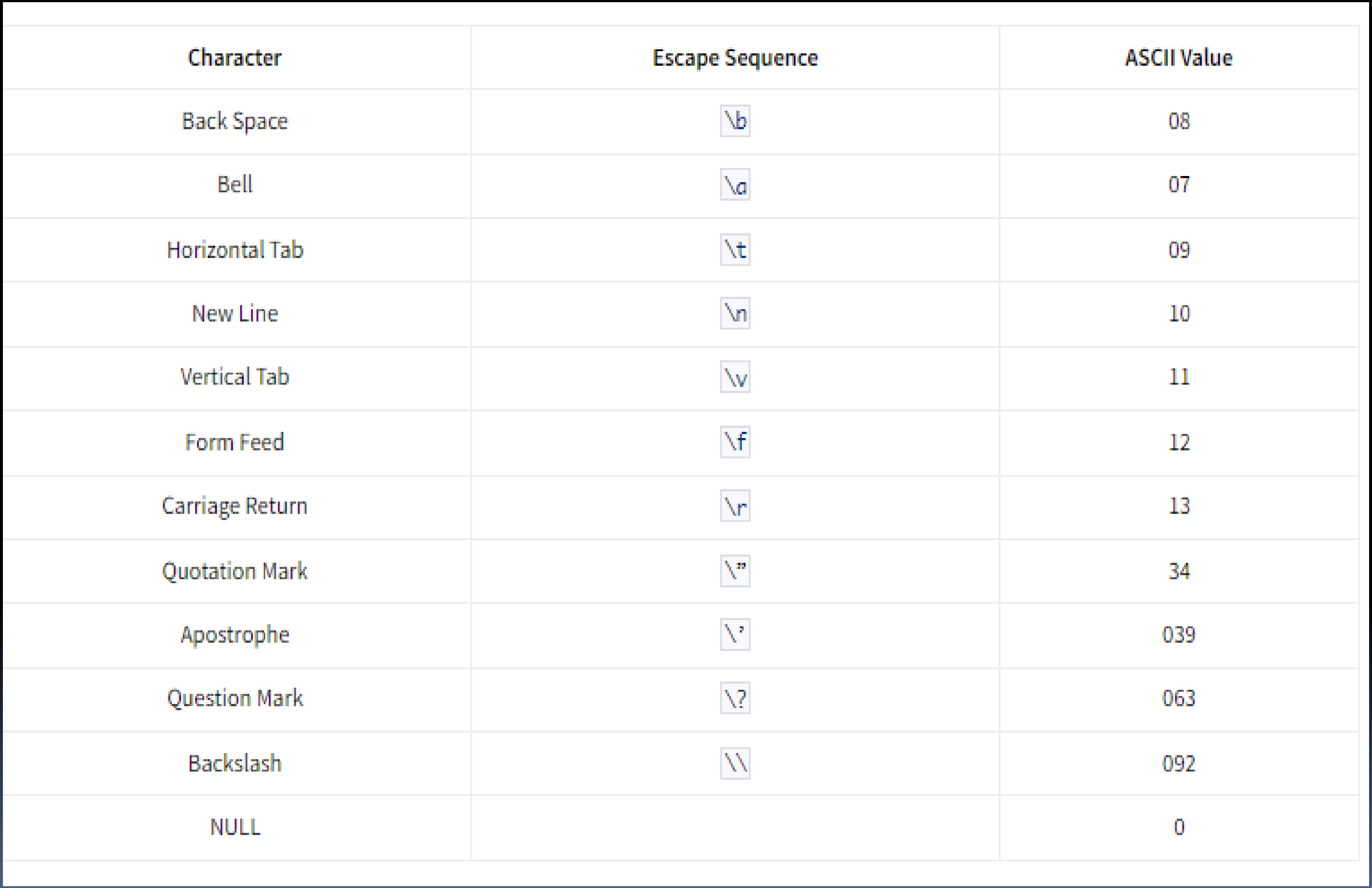
blank space =\b

horizontal tab=\t↔

Carriage return=\r

New line=\n ↓

Form feed=\f

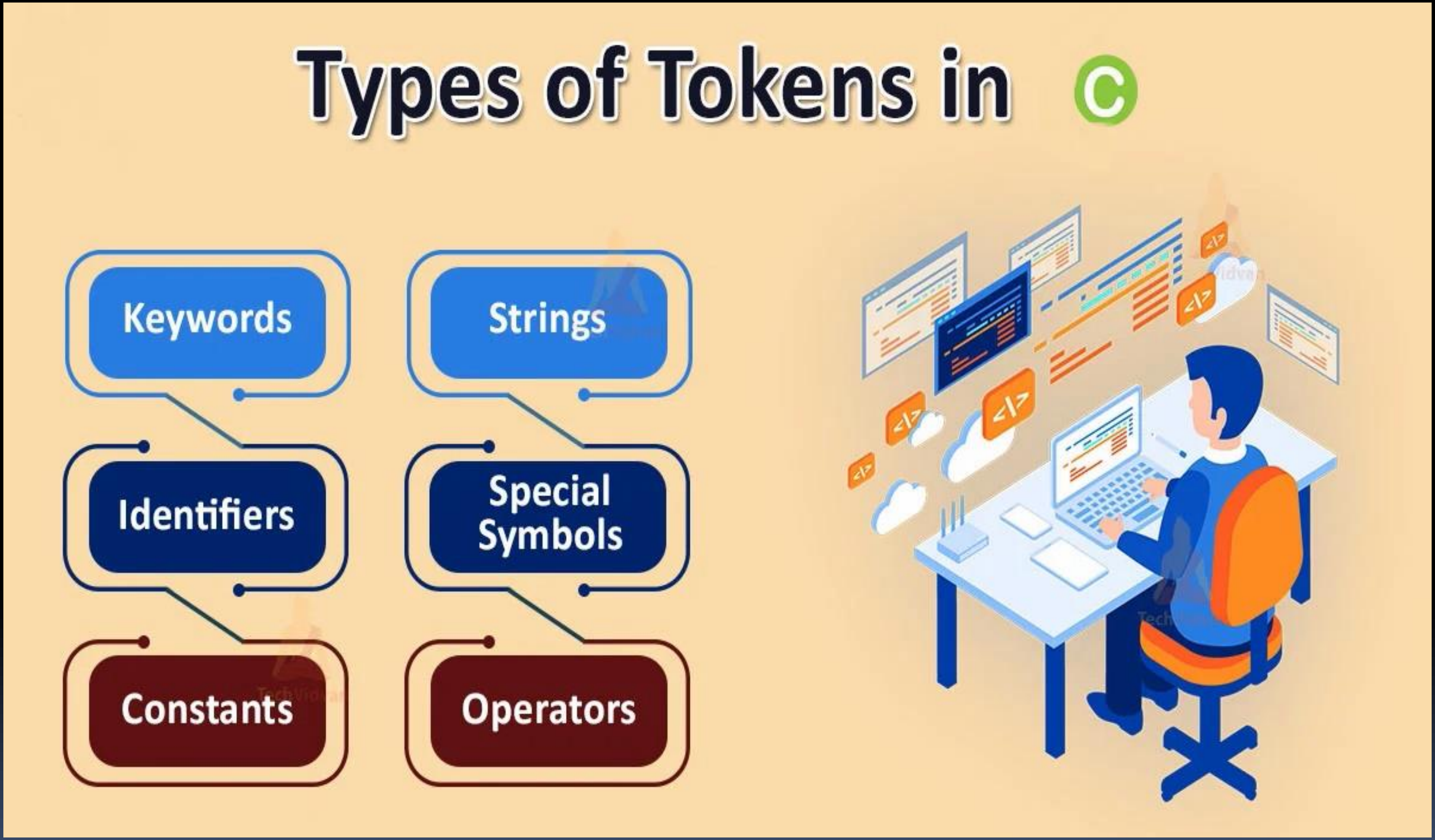


Character	Escape Sequence	ASCII Value
Back Space	`\b`	08
Bell	`\a`	07
Horizontal Tab	`\t`	09
New Line	`\n`	10
Vertical Tab	`\v`	11
Form Feed	`\f`	12
Carriage Return	`\r`	13
Quotation Mark	`\"`	34
Apostrophe	`\'`	039
Question Mark	`\?`	063
Backslash	`\\`	092
NULL		0

C Tokens

- Token is a set of character.
- Tokens in C is the most important element to be used in creating a program in C. We can define the token as the smallest individual element in C. For `example, we cannot create a sentence without using words; similarly, we cannot create a program in C without using tokens in C. Therefore, we can say that tokens in C is the building block or the basic component for creating a program in c language.

There are six types of tokens

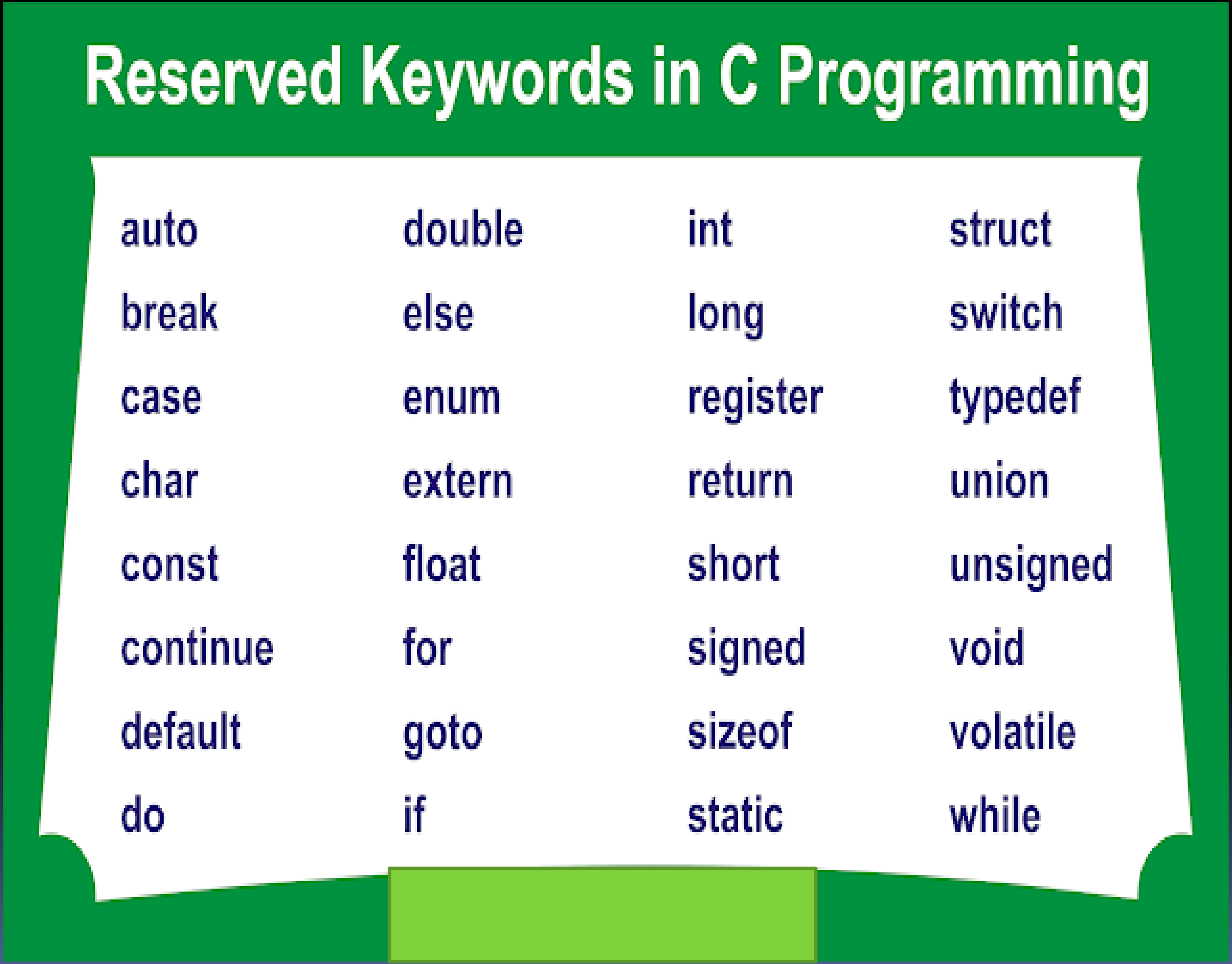


1. Keywords

- Keywords are reserve words that have fixed meanings and these meanings cannot be changed. These words are called keywords (or reserved words), and each of these keywords has a special meaning within the C language.
- All keywords must be written in lowercase.
- There are 32 keywords in C.

1. Keywords

- Keyword can not be used as name such as name like variable, function, user defined datatype.



Reserved Keywords in C Programming

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

2. Identifiers in C

Identifier C are used for naming variables, functions, arrays, structures, etc. Identifiers in C are the user-defined words. It can be composed of uppercase letters, lowercase letters, underscore, or digits, but the starting letter should be either an underscore or an alphabet.

- Identifiers cannot be used as keywords.

Rules for constructing identifiers in C are given below:

- The first character of an identifier should be either an alphabet, number or an symbol [underscore], and then it can be followed by any of the character, digit, or underscore.

- It should not begin with any numerical digit. Or c request that identifier should not start with _. Because it use compiler.

- In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
- Commas or blank spaces cannot be specified within an identifier.
- Keywords cannot be represented as an identifier.
- The length of the identifiers should not be more than 31 characters.
- Identifiers should be written in such a way that it is meaningful, short, and easy to read.

Example of identifier

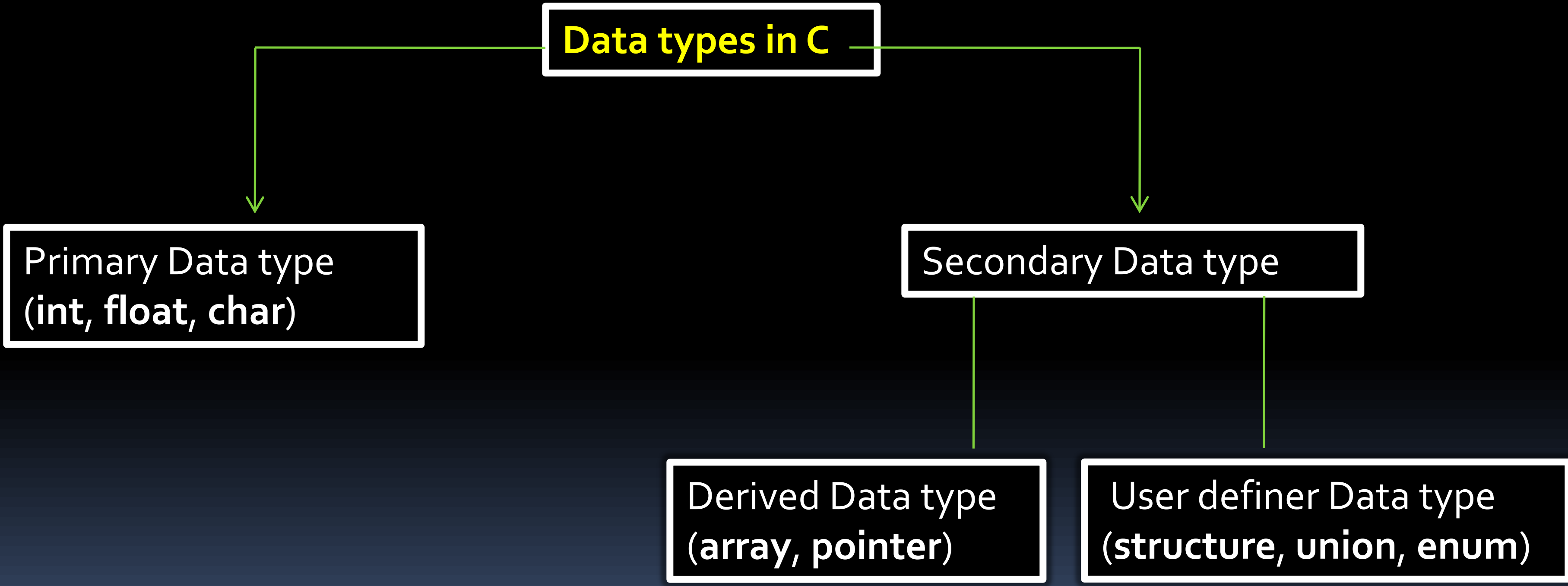
```
int var1, var2;
float Avg;
function sum();
```

- int, float are all keywords.
- var1, var2, Sum, Avg, are the identifiers.

Data Type

❑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 Type (cont…)

- **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 Type

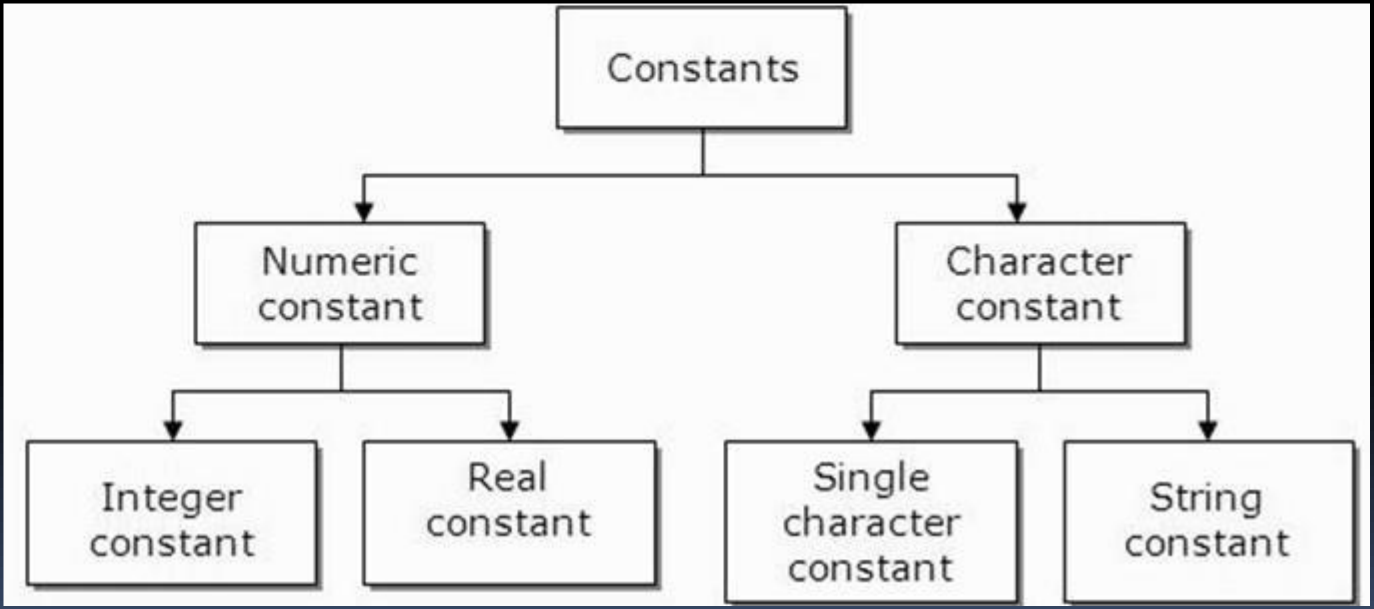
- Secondary data types are not directly supported by the machine.
- It is combination of primary data types to handle real life data in more convenient way.
- It can be further divided in two categories,
  - Derived data types: Derived data type is extension of primary data type. It is built-in system 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.

Secondary Data Type(cont…)

- 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.

Constants

- Constants in C refer to fixed values that do not change during the execution of a program.



- Integer constant:
  - 55,-22,25,60
- Real constant:
  - 21.4,3.56,-0.065,2.0
- Character constant:
  - ‘a’, ‘B’, ‘+’ ,’ ‘, ‘2’
- String constant:
  - “ahemdabad”, ”mca”, ”sarvodaya”, ”ajay”

- **String :**

String is a collection of characters and that can be included in double quotes.

“SARVODAYA”

“ABC”

Special Symbol in C

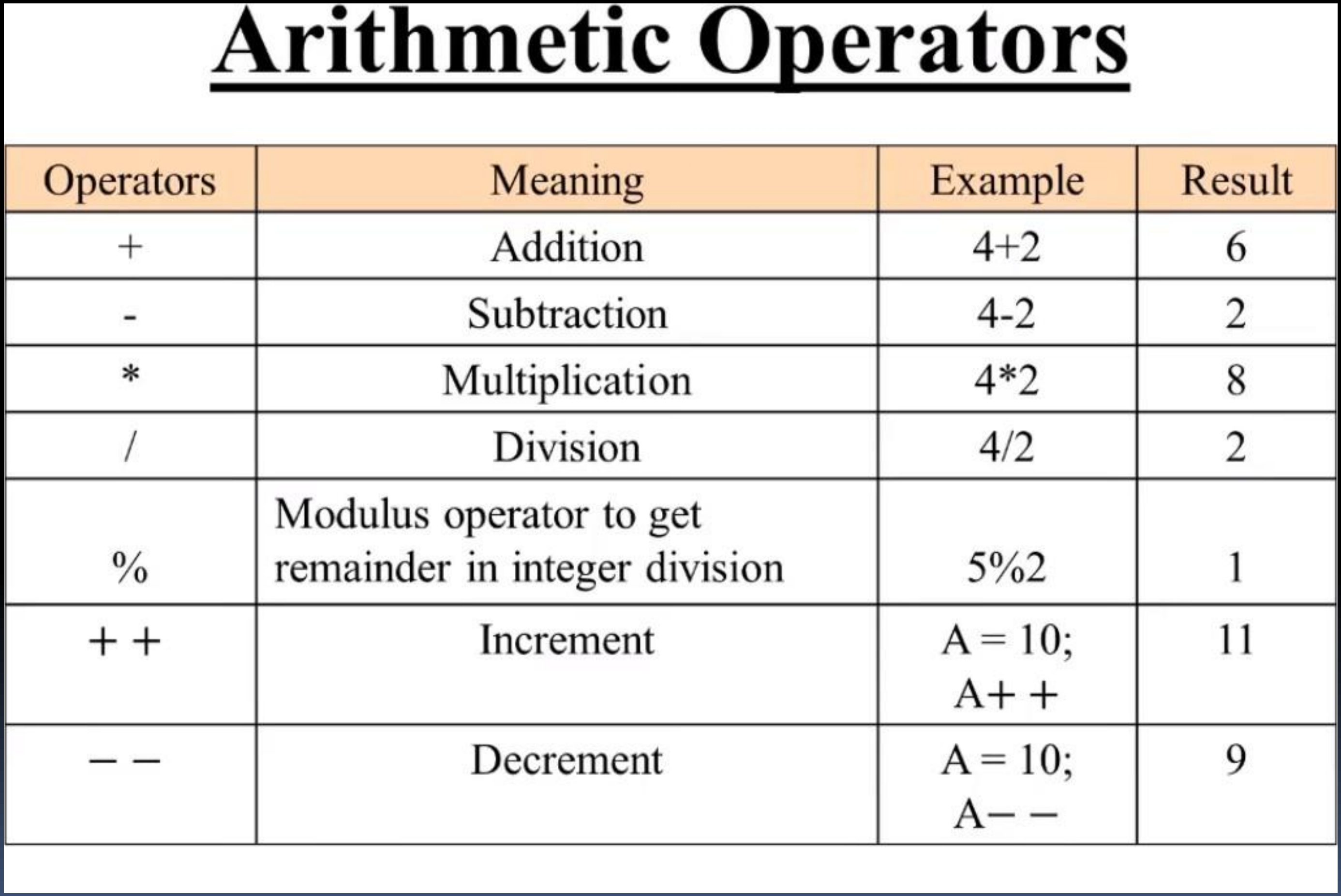
- Some special characters are used in C, and they have a special meaning which cannot be used for another purpose.
- **Square brackets [ ]:** The opening and closing brackets represent the single and multidimensional subscripts.
- **Simple brackets ( ):** It is used in function declaration and function calling. For example, printf() is a pre-defined function.
- **Curly braces { }:** It is used in the opening and closing of the code. It is used in the opening and closing of the loops.

- **Comma (,):** It is used for separating for more than one statement and for example, separating function parameters in a function call, separating the variable when printing the value of more than one variable using a single printf statement.
- **Hash/pre-processor (#):** It is used for pre-processor directive. It basically denotes that we are using the header file.
- **Asterisk (*):** This symbol is used to represent pointers and also used as an operator for multiplication.
- **Tilde (~):** It is used as a destructor to free memory.
- **Period (.):** It is used to access a member of a structure or a union.

Operato rs

- An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators –
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Bitwise Operators
  - Assignment Operators
  - Sizeof & ternary operator

Arithmetic Operators



Arithmetic Operators

Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus operator to get remainder in integer division	5%2	1
+ +	Increment	A = 10; A+ +	11
– –	Decrement	A = 10; A– –	9

```
// Working of arithmetic operators
#include <stdio.h>
#include<conio.h>
void main()
{
    int a = 9,b = 4, c;
     c = a+b;
    printf("a+b = %d \n",c);
    c = a-b;
    printf("a-b = %d \n",c);
    c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
    printf("Remainder is= %d \n",c);
 return 0;
}
```

- Output:

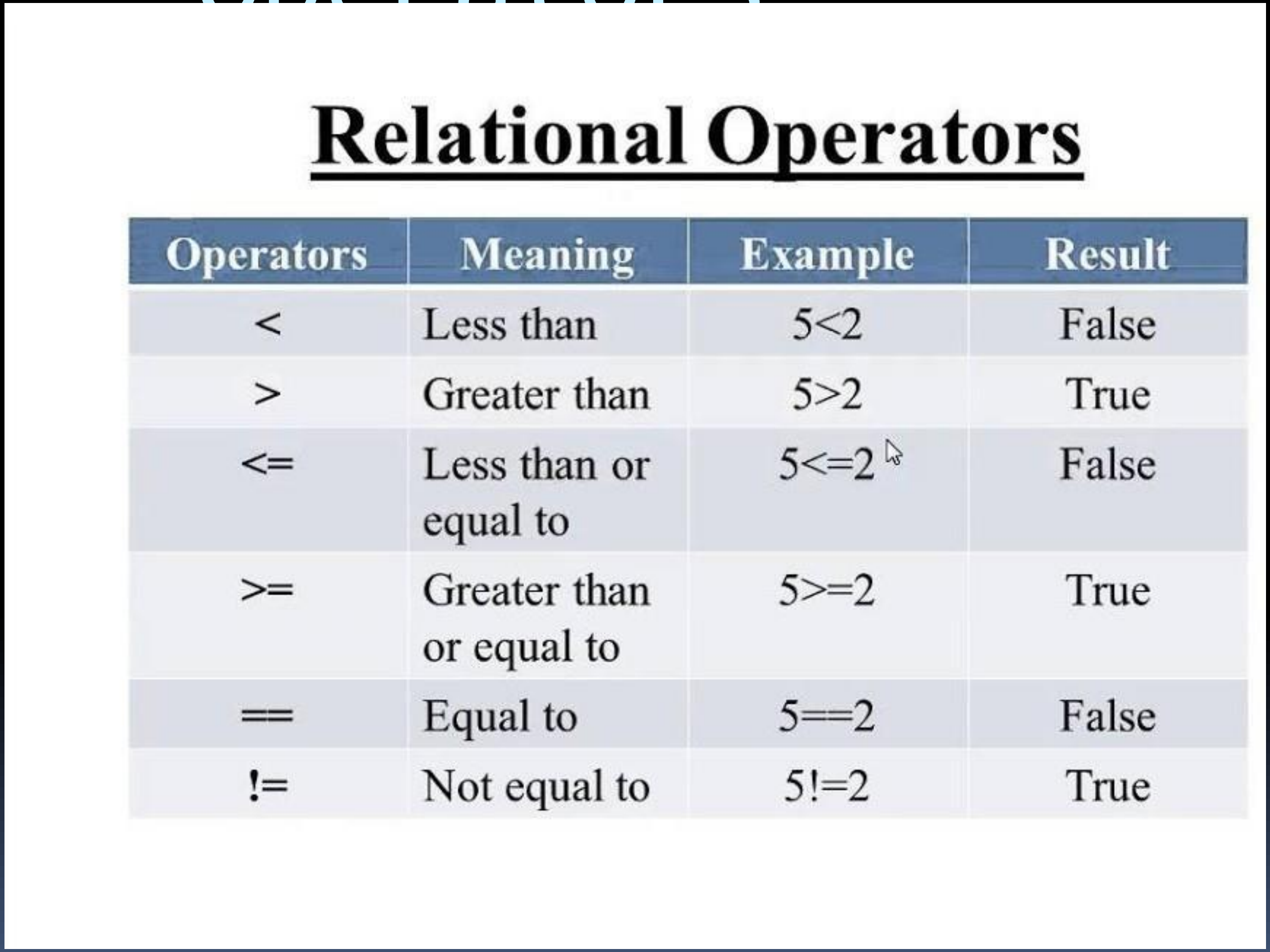
```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
 Remainder is = 1
```

```
// Working of increment and decrement operators
#include <stdio.h>
#include <conio.h>
void main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;
    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    printf("++c = %f \n", ++c);
    printf("--d = %f \n", --d);
    return 0;
}
```

- Output :

```
++a = 11
--b = 99
++c = 11.500000
--d = 99.500000
```

Relational Operators



Relational Operators

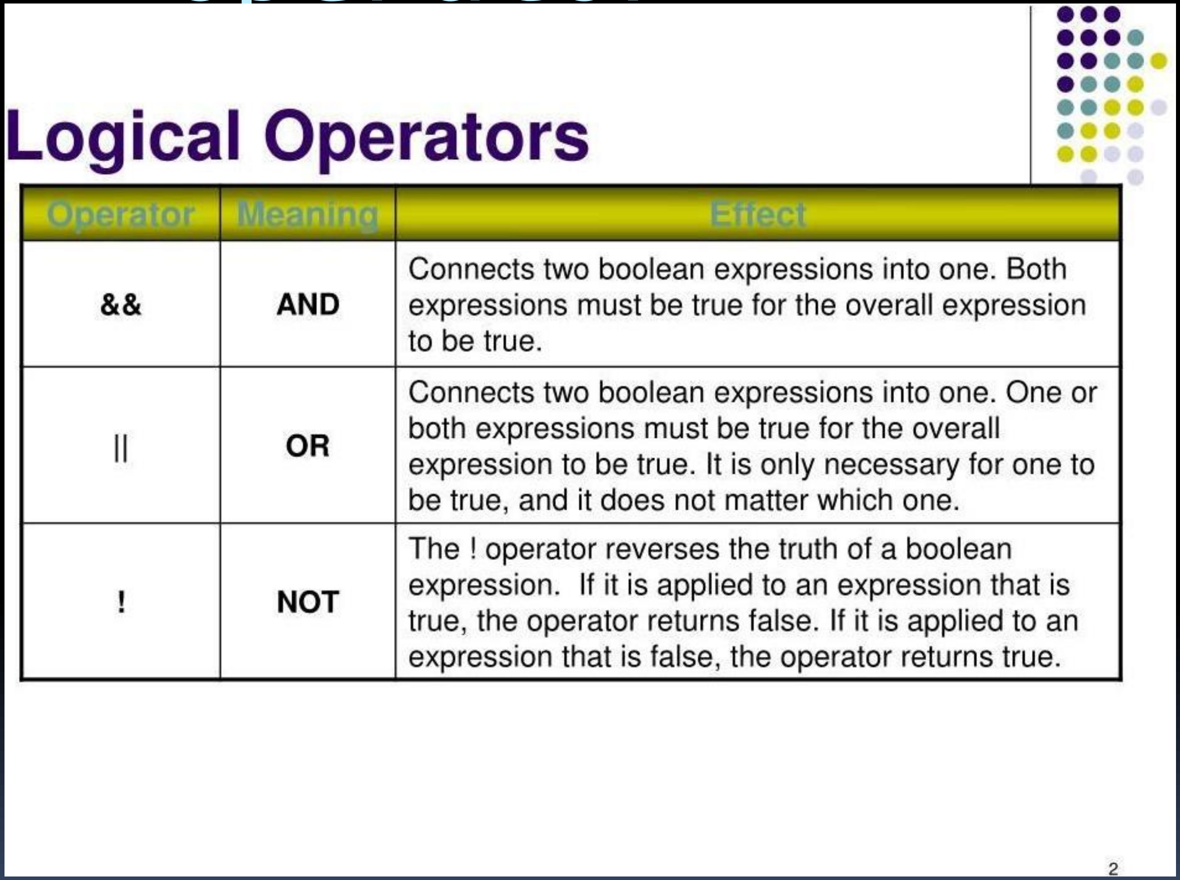
Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

```
#include  <stdio.h>
#include <conio.h>
void main()
{
        int a = 9;
        int b = 4;
        printf(" a >  b: %d \n", a > b);
        printf("a >= b: %d \n", a >= b);
        printf("a <= b: %d \n", a <= b);
        printf("a <  b: %d \n", a < b);
        printf("a == b: %d \n", a == b);
        return 0;
}
```

•Output:

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

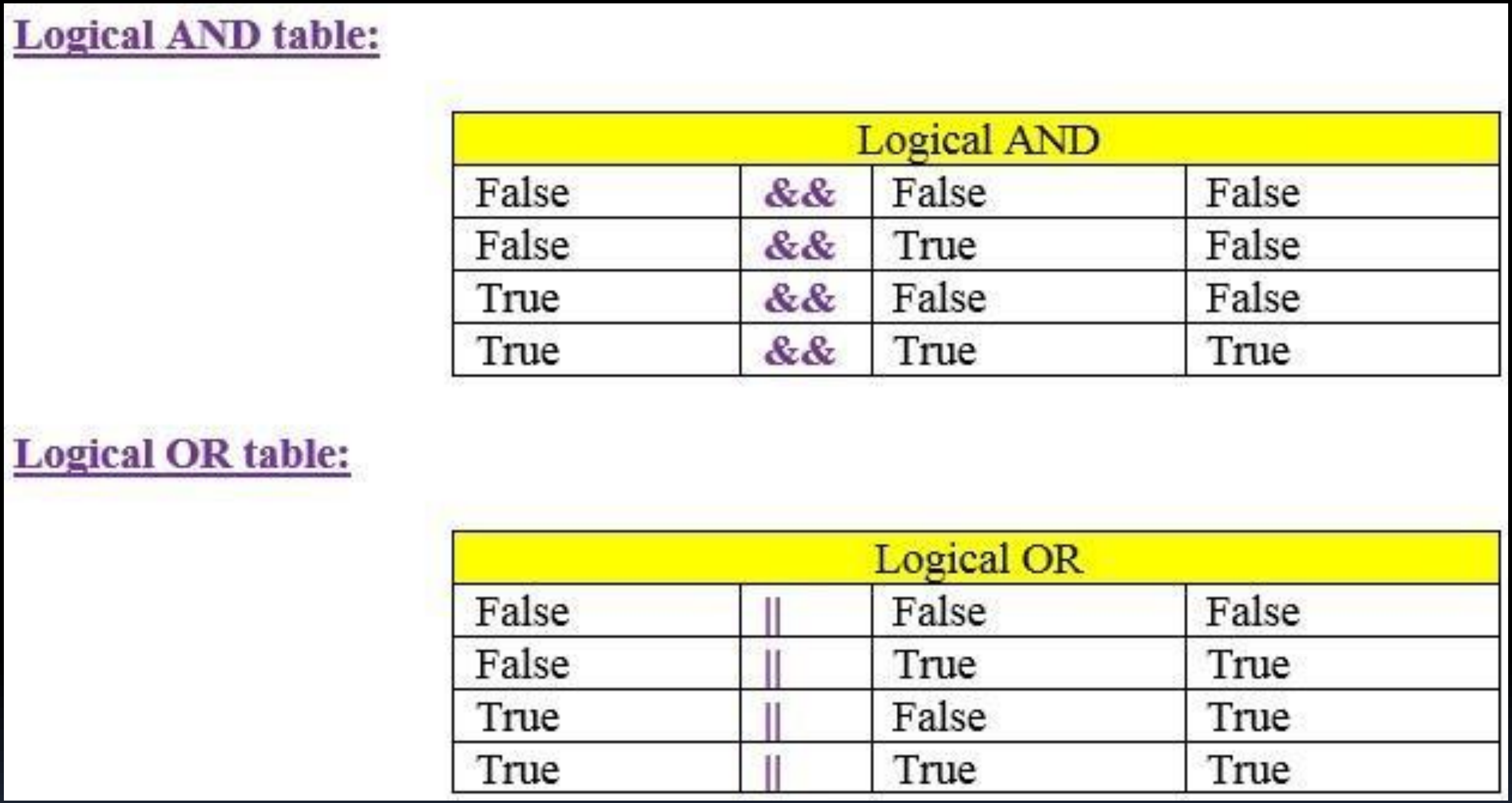
Logical operator



Logical Operators

Operator	Meaning	Effect
&&	AND	Connects two boolean expressions into one. Both expressions must be true for the overall expression to be true.
||	OR	Connects two boolean expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which one.
!	NOT	The ! operator reverses the truth of a boolean expression. If it is applied to an expression that is true, the operator returns false. If it is applied to an expression that is false, the operator returns true.

2



Logical AND table:

Logical AND			
False	&&	False	False
False	&&	True	False
True	&&	False	False
True	&&	True	True

Logical OR table:

Logical OR			
False	||	False	False
False	||	True	True
True	||	False	True
True	||	True	True

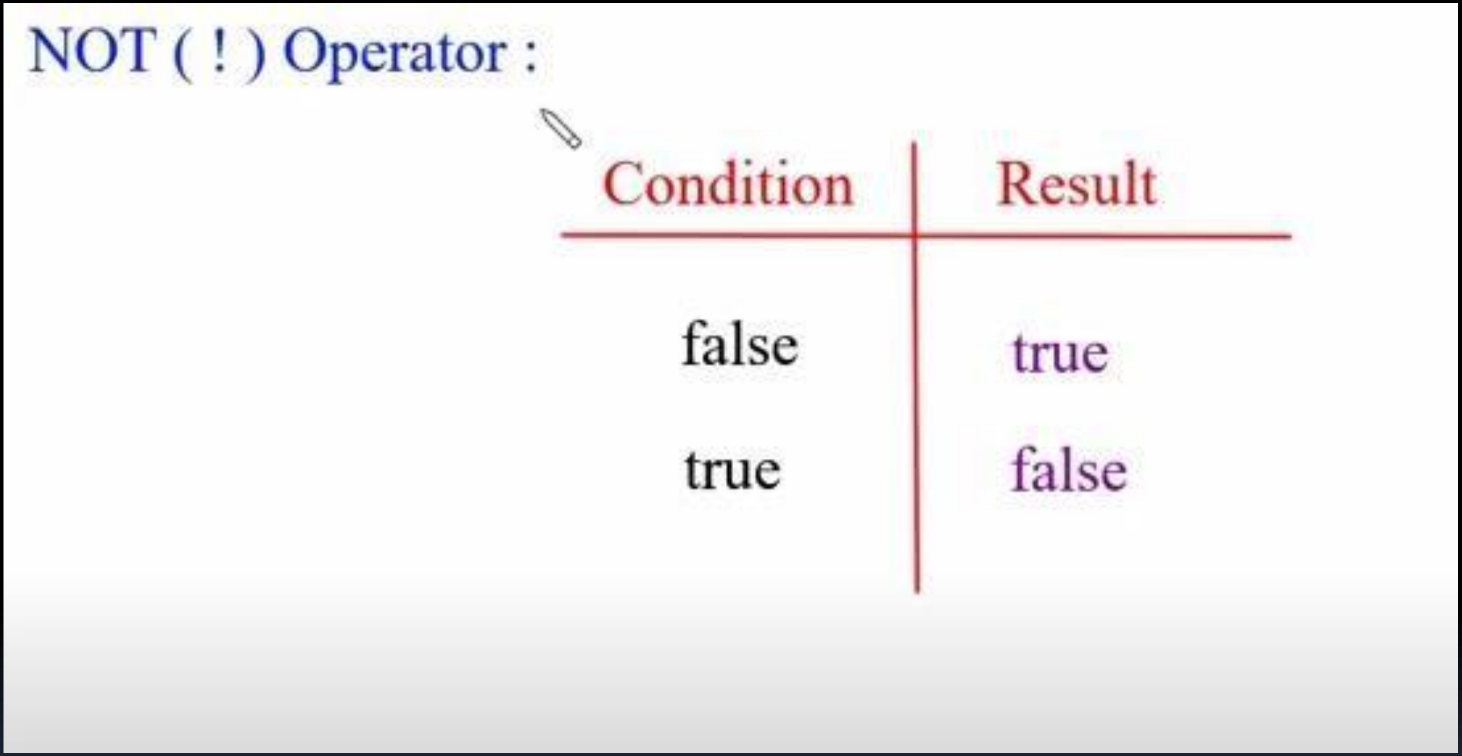
- Example of && operator and || operator

  a=9,b=4

  (a==9)&&(b<a)=0

  (a>=b)||(b>7)

a=9,b=4
!(a>4)=0



NOT (!) Operator :

Condition	Result
false	true
true	false

```
//logical operator
#include <stdio.h>
void main()
{
   int a = 5, b = 5, c = 10, result;
   result = (a == b) && (c > b);
   printf("(a == b) && (c > b) is %d \n", result);
   result = (a == b) && (c < b);
   printf("(a == b) && (c < b) is %d \n", result);
   result = (a == b) || (c < b);
   printf("(a == b) || (c < b) is %d \n", result);
   result = (a != b) || (c < b);
   printf("(a != b) || (c < b) is %d \n", result);
   result = !(a != b);
   printf("!(a != b) is %d \n", result);
   result = !(a == b);
   printf("!(a == b) is %d \n", result);
   return 0;
}
```

Output

```
(a == b) && (c > b) is 1
(a == b) && (c < b) is 0
(a == b) || (c < b) is 1
(a != b) || (c < b) is 0
!(a != b) is 1
!(a == b) is 0
```

Sizeof & ternary operator

- The memory size (in bytes) of a data type or a variable can be found with the sizeof operator:

```
#include <stdio.h>
int main() {
 printf("%d\n", sizeof(int));
 printf("%d\n", sizeof(float));
 printf("%d\n", sizeof(double));
 printf("%d\n", sizeof(char));
 return 0;
}
```

Output

- 2
- 4
- 8
- 1

Assignment operator & Bitwise operator

- Assignment operators are used to assign values to variables.

	Operation Performed
=	Simple assignment
*=	Multiplication assignment
/=	Division assignment
%=	Remainder assignment
+=	Addition assignment
-=	Subtraction assignment
<<=	Left-shift assignment
>>=	Right-shift assignment
&=	Bitwise-AND assignment
^=	Bitwise-exclusive-OR assignment
|=	Bitwise-inclusive-OR assignment

```
#include <stdio.h>
void main()
 {
int a = 7;
int Total = 21;
 printf(" Value of the Total = %d \n", Total += a );
printf(" Value of the Total = %d \n", Total -= a );
printf(" Value of the Total = %d \n", Total *= a );
printf(" Value of the Total = %d \n", Total /= a );
printf(" Value of the Total = %d \n", Total %= a );
getch();
}
```

Output

- Value of the Total = 28
- Value of the Total = 21
- Value of the Total = 147
- Value of the Total = 21
- Value of the Total = 0

Write a program sum of 2 values

```
#include <stdio.h>
void main()
{
        int no1,no2,result;
        printf("entr val 1:\n");
        scanf("%d",&no1);
        printf("entr val 2:\n");
        scanf("%d",&no2);
        result=no1+no2;
        printf("%d",result);
        return 0;
}
```

- Write a program for find out average of two value.
- Write a program for swap two value with out using third variable.
- Write a program for find out area of rectangle.

Bitwise Operator

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise complement
<<	Shift left
>>	Shift right

Let us suppose the bitwise AND operation of two integers 12 and 25.

10 = 00001010 (In Binary)

12 = 00001100 (In Binary)

Bit Operation of 10and 12

```
  00001100
& 00011001
  ________
  00001000 = 8 (In decimal)
```

```
#include <stdio.h>
#include<conio.h>
void main()
{
    int a = 10;
   int b = 12;
   int c = 0;

   c = a & b
  printf("Line 1 - Value of c is %d\n", c );

   c = a | b;
    printf("Line 2 - Value of c is %d\n", c );

   c = a ^ b;
   printf("Line 3 - Value of c is %d\n", c );


}
```