UNIT 2

Elements of C

Character Set

- Set of characters that are used to form words, numbers and expression in C is called c character set.

- Characters in C are grouped into the following four categories:
  1. Letters and alphabets (*A…Z, a…z*)
  2. Digits (0,1,2,3,4,5,6,7,8,9)
  3. Special Characters (, . ; : ? ‘ “ & ^ * - + < >)
  4. White spaces *(Blank Space, Horizontal tab etc)*

Keywords

- These are predefined words for a C programming language.
- All keywords have fixed meaning and these meanings cannot be changed.
- Ex.

auto	double	int	struct
Break	Else	Long	Switch
Case	Enum	Register	Typedef
Char	Return	Union	Const
Float	Short	Unsigned	Continue
Void	For	Default	Goto
Sizeof	Volatile	Do	If
Static	While	Extern	signed

Identifiers

- Every word used in C program to identify the name of variables, functions, arrays, pointers and symbolic constants are known as identifiers.

- Names given by user and consist of a sequence of letters and digits, with a letter as the first character. e.g. myVariable, myName, heyYou, callThisNumber45, add_this_number etc.

- There are certain rules to be followed while naming identifiers. (**KEEP THIS IN MIND**)

Rules to be followed while naming identifiers

1. It must be a combination of letters and digits and must begin with a letter.
2. Underscore is permitted between two digits and must begin with a letter.
3. Only first 31 characters are significant.
4. **Keywords** cannot be used.
5. It is case sensitive, i.e. uppercase and lowercase letters are not interchangeable.

Data Types

- 10 is a whole number where as 100.5 is a fractional/rational number.
- Similarly in C 10 is an ***integer*** number whereas 100.5 is a ***float*** number.
- There are variety of data types available.
- ANSI C supports three classes of data types:
  - Primary/fundamental data types
  - User-defined data types
  - Derived data types

Primary data types are categorized into five types:

1. Integer type (int)
2. Floating point type (float)
3. Double-precision floating point type (double)
4. Character type (char)
5. Void type (void)

Integer types

- Integers are whole numbers
- Requires 16 bit of storage. i.e. 2 bytes
- Three classes of integer:
  - Integer (**int**)
  - Short integer (**short int**)
  - Long integer (**long int**)
- Both signed and unsigned forms.
- Defined as:

  int a;

  Int myVar=6;

Signed Integer	Unsigned Integer
It represents both positive and negative integers	It represents only positive integers
The data type qualifier is **signed int or int.** Variables are defined as: signed int a; Int b;	The data type qualifier is **unsigned int or unsigned** Variables are defined as: unsigned int a; unsigned b;
By default all int are signed	Unsigned int have to be declared explicitly
It reserves 16-bit (2 bytes) in memory	It reserves 16-bit (2 bytes) in memory
Range -2^{15} to $+2^{15}$ i.e. -32,768 to 32,767	Range from 0 to $+2^{16}$ i.e. 0 to 65,535
Its conversion character is **d**	Its conversion character is **u**

Floating Point Types

- Floating point types are fractional numbers
- In C it is defined by **float.**
- Reserves 32bits i.e. 4 bytes
- Variable is defined as:

  float a;

  float myValue=56.5;

Assignment

- Write something about Signed and Unsigned short Integers.
- Write something about Signed and Unsigned long Integers.
- Write something about Double Precision and Long Double Precision Floating point

Character Type

- A single character can be defined as a character type data.
- Stored in 8 bits (1 byte).
- The qualifier signed or unsigned may be used with **char.**
- The unsigned char has values between 0 and 255.
- The signed char has values from -128 to 127.
- The conversion character for this type is **c**

- Each character is represented by an ASCII(American Standard code for information interchange)
- Ex
  - “A” is represented by 65
  - “B” is represented by 66
  - “a” is represented by 97
  - “z” is represented by 122
- With conversion character d, it will display ASCII value.
- With conversion character c, it will display character.

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Note

- The difference of corresponding uppercase and lowercase character is always 32.
- i.e
  - ASCII value of ‘a’ – ASCII value of ‘A’ = 32
  - ASCII value of ‘m’ – ASCII value of ‘M’ = 32

- Using this logic, we can convert uppercase letter into its lowercase and vice versa.

Void Type

- This void type has no value.

- This is usually used to specify a type of function when it does not return any value to the calling function.
- Ex
  - void main()
  - void whatIsThis();

User Defined Data Types

- C supports a feature called type definition which allows users to define an identifier that would represent an existing data type.

- **typedef** statement is used to give new name to an existing data type.

- It allows users to define new data types that are equivalent to an existing data types.

- General form:
  - **typedef** *existing_data_type new_name_for_existing_data_type;*
- Here,
  - *existing_data_type* is any one of the fundamental data type.
  - *new_name_for_existing_data_type* refers to a new identifier.
- Ex:
  - **typedef int** integer;
    - integer symbolizes **int** data type. Now we can declare int variable “a” as “integer a” rather than “int a”

Constants

- A **constant** is a quantity that doesn't change during the 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**. There are enumeration constants as well.

Integer Literals

- An integer literal can be a **decimal**, **octal**, or **hexadecimal** constant.
- A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.
- An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively.
- Following are other examples of various types of integer literals −

85	/* decimal */
0213	/* octal */
0x4b	/* hexadecimal */
30	/* int */
30l	/* long*/

Character Constants

- Character literals are enclosed in single quotes,
  - e.g., 'x' can be stored in a simple variable of **char** type.

- A character literal can be a
  - plain character (e.g., 'x'),
  - an escape sequence (e.g., '\t'),
  - or a universal character (e.g., '\u02C0').

- There are certain characters in C that represent special meaning when preceded by a backslash for example, newline (\n) or tab (\t).

String Constants

- Sequence of characters enclosed in double quotes.
- May contain letters, numbers, special characters or blank spaces.
- Eg.
  - “hello”
  - “hie”
  - “2048”

Variables

- A symbolic name which is used to store data item i.e. a numerical quantity or a character constant.
- Unlike constant, the value of a variable can change during the execution of a program.
- The same variable can store different value at different portion of a program.
- Variable name may consist of letters, digits or underscore characters.

Variable declaration

- Any variable should be defined before using it in a program

- Variable declaration syntax:
  - **data-type variable_name1, variable_name2…..**

- Valid declaration are:
  - int n1;
  - int centi, temp;
  - float radius;
  - char gender;

n1 =1	//valid
radius= 2.6	//valid
gender =‘M’	//valid
temp= ‘F’	//Invalid

Rules for Variable Declaration

- The variable name should start with only letters.
- The variable name shouldn't not be keyword
- White spaces are not allowed between characters of variable but underscores are approved
- The variable name is case sensitive.
  - **TEMP** and **temp** is different variable
- No two variables of the same name are allowed to be declared in the same scope

Preprocessor Directives

- Collection of special statements that are executed at the beginning of a compilation process.

- Placed in the source program before the main function.

#include<stdio.h>	//used for file inclusion
#define PI 3.1416	//defining symbolic constant PI
#define TRUE 1	//used for defining TRUE as 1
#define FLASE 0	//used for defining FALSE as 0

- These statements are called preprocessor directives as they are processed before compilation of any other source code in the program.

Escape sequences

- An escape sequence is a non-printing characters used in C.

- Character combination consisting of backslash (\) followed by a letter or by a combination of digits.

- Each sequences are typically used to specify actions such as carriage return, backspace, line feed or move cursors to next line.

Escape character	Description of Action	Code Used in Programming
\a	Alert sound. A beep is generated by the computer on execution	"\a"
\b	Backspace.	"\b"
\f	Form feed.	"\f"
\n	New line. Shifts the cursor to new line. Words on the right of "\n" go to next line	"\n"
\r	Carriage return. Positions the cursor to the beginning of current line.	"\r"
\t	Horizontal tab, it moves the cursor by a number of spaces or to next tab stop	"\t"
\v	Vertical tab.	"\v"
\\	Backslash. Displays a black slash character (\).	"\ \"
\'	Displays a single-quote character(').	"\'"
\"	Displays a double-quote character(").	"\""
\?	Displays question mark (?).	"\ ?"
\0	Null character. Marks the end of string.	"\0"

```
#include<stdio.h>
#include<conio.h>
Void main(){
        printf("Hello! \n I am testing an escape
        sequence");
        getch();
}
```

OUTPUT:

Hello!

I am testing an escape sequence.

```
#include<stdio.h>
#include<conio.h>
Void main(){
        printf(“Hello \t World \n”);
        printf(“He said, \”hello\””);
}
```

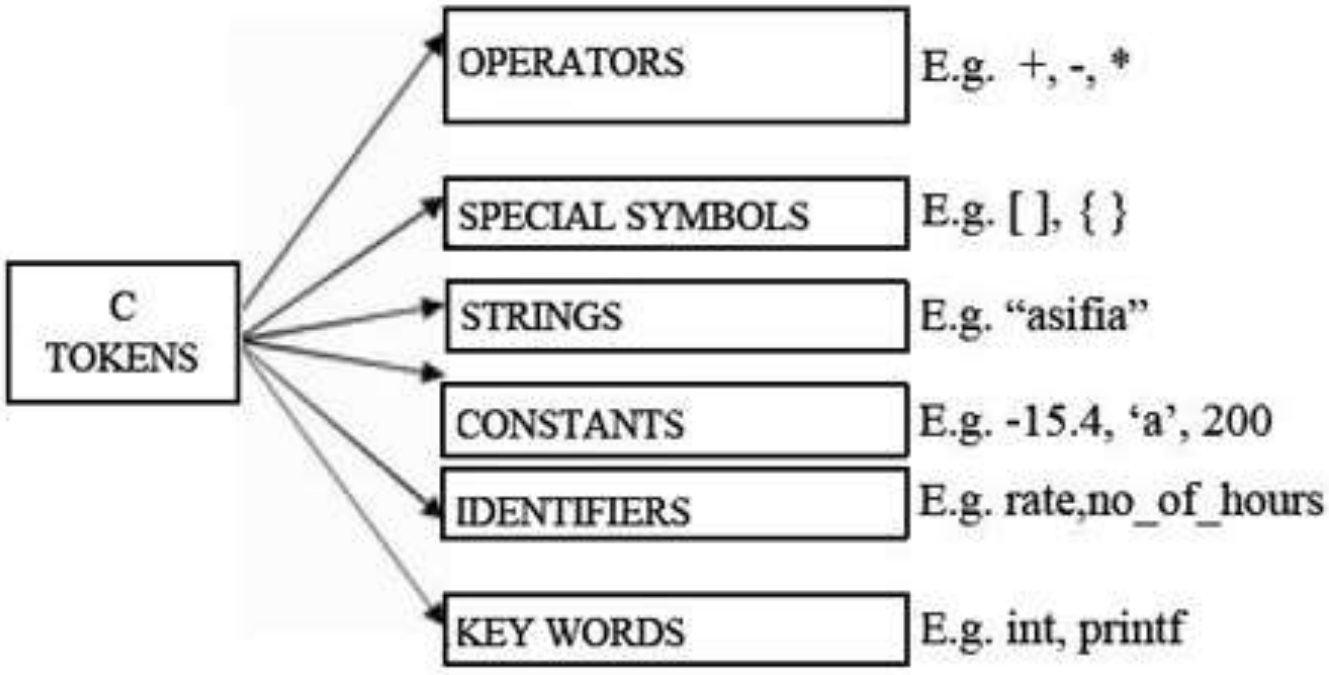
OUTPUTS:

Hello World

He said, “Hello”

Tokens in C

- The basic elements recognized by the C compiler are the “tokens”



Delimiters

- A delimiter is a unique character or series of characters that indicates the beginning or end of a specific statement, string or function body set.

- Delimiter examples include:
  - Round brackets or parentheses: ( )
  - Curly brackets: { }
  - Escape sequence or comments: /*
  - Double quotes for defining string literals: " "

Expressions

- In programming, an expression is any legal combination of symbols that represents a value.
- For example, in the C language x+5 is a legal expression.
- Every expression consists of at least one operand and can have one or more operators.
- Operands are values and Operators are symbols that represent particular actions.
- Ex:
  - in the C language x+5 is a legal expression.

Types of Expression

Type	Explanation	Example
Infix	Expression in which Operator is in between Operands	a + b
Prefix	Expression in which Operator is written before Operands	+ a b
Postfix	Expression in which Operator is written after Operands	a b +

ASSIGNMENT

- Write something about **Real Constants** and also **fractional form constants** and **exponential form constants**(mantissa/exponents).

- Write something about **Symbolic Constants.** Write rules for defining a symbolic constants. Also explain advantages of **symbolic constants**.

END