The C character set

- **Alphabets/Letters:** A, B, C, …, Z, a, b, c, …, z.
- **Digits:** 0, 1, 2, …, 9
- **Special Symbols:** ~ ‘ @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; “ ` < > , . ? /

Identifiers:

- Name given to symbolic constants, variables, functions, labels, arrays etc.

Constants:

- A constant is a quantity that does not change during the execution of the program. This quantity can be stored at a location in the memory of the computer.
- **Types of constants:**
  - Primary Constants
    - Numeric Constants
      - Integer constant: 401, +782, -900
      - Real Constant: +125.48, 209.0, -11.23, +3.2e-5, 4.1e8
    - Character Constant
      - Single character constant: ‘A’, ‘+’, ‘8’
      - String constant: “Ram”, “Hello”
  - Secondary Constant
    - Array
    - Pointer
    - Structure
    - Union
    - Enum
- **Rules for constructing Integer Constants**
  - An integer constant must have at least one digit.
  - It must not have a decimal point.
  - It could be either possitive or negative.
  - If no sign precedes an integer constant it is assumed to be possitive.
  - No commas or blanks are allowed within an integer constant.
  - Range (For 16 bit computer): -32768 to 32767
- **Rules for constructing Real Constants**
  - A real constant must have at least one digit.
  - It must have a decimal point.
  - It could be either possitive or negative.
  - Default sign is possitive.
  - No commas or blanks are allowed within a real constant.
- **Rules for constructing Real Constants expressed in exponential form.**
  - The mantissa part and the exponential part should be separated by a letter e.
  - The mantissa part may have a possitive or negative sign.
  - Default sign of mantissa part is possitive.
  - The exponent must have at least one digit, which must be a possitive or negative integer. Default sign is possitive.
  - Range: -3.4e38 to 3.4e38
  - Every character constant has a decimal value.
- **Rules for constructing single Character Constants**

- A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.
- Maximum length of a character constant can be 1 character.

- **Rules for constructing string Constants**
  - A string constant is a sequence of characters enclosed in double quotes.
  - A character constant is not equivalent to a single character string constant. (i.e. 'A' is not same as "A").

Variables:

- It is a quantity that may vary during the execution of the program.
- Variable names are names given to location in the memory of computer where different constant are stored.
- **Rules for constructing variable names:**
  - A variable name is any combination of 1 to 8 alphabets, digits or underscores. Some compiler allows up to 40 characters.
  - First character must be an alphabet.
  - No commas or blanks are allowed within a variable name.
  - No special symbol other than an underscore can be used in a variable name.
  - Variable name should not be a keyword.
- **Example:** si_int, pop_e_98
- We should always use meaningful variable names. For e.g. we should use rad rather than r to represent radius.

Keywords:

- These are the words whose meaning has already being explained to the C compiler.
- Keywords can't be used as variable names.
- The following are some of the ANSI C Keywords:

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

Data Types:

- Data type defines a set of values that a variable can store and the type of operations can be done on that.
- ANSI C supports the following kinds of data types:
  - Primary data types: int, char, float
  - User-defined data types: structure, union, enum
  - Derived data types: Array, Pointer, Function
  - Empty data set: void
- **Size and range of data types on a 16-bit machine / 32-bit machine:**

Data Type	Range (-2^{n-1} to $2^{n-1}-1$)	Bytes	Format

	n: no of bits		
char	-128 to 127	1	%c
signed char	-128 to 127	1	%c
unsigned char	0 to 255	1	%c
int	-32768 to 32767	2 / 4	%d
signed int	-32768 to 32767	2 / 4	%d
unsigned int	0 to 65535	2 / 4	%u
short signed int	-32768 to 32767	2	%d
short 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 / 12	%Lf

- Data types are used to declare variables before the use of variables.
- **Syntax of variable declaration:** datatype variable1, variable2, . . .
- **Example of variable declaration:** int basal, grosssal;

Instructions

- It is the combination of constant, variables and keywords.
- **Types of Instructions:**
  - **Type declaration instruction:** to declare the type of variables used in a C program.
  - **Input/Output instruction:** to perform the functions of supplying input data to a program and obtaining the output results from it.
  - **Arithmetic instruction:** to perform arithmetic operations between constants and variables.
  - **Control instruction:** to control the sequence of execution of various statements in a C program.

Statements

- It is the combination of identifiers which are executed in sequence unless specified. There are several groups of statements inC. They are:
- **Labelled statement:**
  - Identifier : statement — x:
  - Case const : statement — case 1:
  - Default : statement — default:
- **Expresson statement:**
  - Assignments — x=5
  - Function calls — display();
- **Compound Statement:**

```
{
        declaration list
        statement list
}
```

- **Selection statement:**
  - if(expression)

statement;

- if(expression)
  statement;
  else
  statemet;
- swich(expression)
  statement;

- **Iteration statement:**
  - while(expression)
    statement;
  - do
    {
    statement;
    }while(expression);
  - for(initialization;condition;increment/decrement)
    statement;
- **Jump statement:**
  goto, continue, break, retrun

Expressions:

- An expression may be a single data item or combination of data items.
- **Example:** 5, x, a+b, d=e+f, x<=y, ++x

Operators:

- Operators are symbols which specifies a particular operation on a data item. The data item on which operation is done called operand.
- According to the no of operands used with the operators, they are classified into 3 types:
  - **Unary Operators:** These are used with single operand.
    Example: a++, ++a, a--, --a, Unary +, Unary –
  - **Binary Operators:** These operators operate on two operands.
    Example: a+b, a*b, a/b, a-b, a%b
  - **Ternary Operators:** These operators operate on three operands and two operations.
    Example: a>b ? a+b : a-b
- According to the type of operations we perform on operands, operators are divided into the following types:
  - **Arithmetic Operators:** These types of operators are used or operated on variables or constants.
    - **Example:** +, - , * , / , %, Unary +, Unary –
    - Unary minus (–) multiplies its single operand by –1.
  - **Increment/Decrement Operators:** This is a special type of operator where the value of the variable is increased or decreased by one.
    - These operators are only operated on variables.
    - Increment operators: a++, ++a
    - Decrement operators: --a, a—
    - Types of Increment/Decrement Operators:
      - **PREFIX:** In this form the operator is present before the variable. Example: ++a, --a. The meaning is first the

value of the variable is increased or decreased then other operations are carried out.

- **POSTFIX:** In this form the operator is present after the variable. Example: a++, a--. The meaning is first the operations are carried out then the value of the variable is increased or decreased.

- **Modular division Operator:** Modular division produces the remainder of an integer division. It can't be used with floating point data.
- **Relational Operator:** These operators are used to find the relation between two quantitites by comparing them. The value of the relational expression is either one or zero.
  - **Example:** >, >=, <, <=, !=, = =
  - **Difference between = and ==**
    - **a = b :** The value of b is assigned to a.
    - **a = = b:** The two operands a and b are compaired. If both are same then result is 1 else result is 0.
- **Logical Operators:** Logical operators combine two or more relational expressions to form a single logical expression or a compound relational expression.
  - **Example:** && (Logical AND), || (Logical OR), !(NOT)

Op 1	**Op 2**	**Op1&& Op2**	**Op 1 || Op2**	**! Op2**
T	T	T	T	F
T	F	F	T	T
F	T	F	T	
F	F	F	F	

- **Bitwise Operators:** These operators operate on variable or constants bitwise. That means the value of the operand will be first converted to binary form and then the operation will be carried out. These operators may not be applied to float or double. The types of bitwise operators are:
  - **Bitwise AND (&):** The result of bitwise AND operation is 1 if both the bits have a value 1, otherwise it is 0.
    Example: if x = 13, y = 25 and z = x & y then the value of z will be 9.
    x: 0000 0000 0000 1101
    y: 0000 0000 0001 1001
    z: 0000 0000 0000 1001
  - **Bitwise OR (|):**The result of bitwise OR operation is 1 if at least one of the bits have a value 1, otherwise it is 0.
    Example: if x = 13, y = 25 and z = x | y then the value of z will be 29.
    x: 0000 0000 0000 1101
    y: 0000 0000 0001 1001
    z: 0000 0000 0001 1101
  - **Bitwise Exclusive OR (^):** The result of exclusive OR operation is 1 if only one of the bits is 1, otherwise it is 0.
    Example: if x = 13, y = 25 and z = x ^ y then the value of z will be 20.

x: 0000 0000 0000 1101
y: 0000 0000 0001 1001
z: 0000 0000 0001 0100

- **Left Shift (<<):** This operator is used to move bit patterns to the left. By executing **op<<n** causes all the bits in the operand op to be shifted to the left by n positions. The left most n bits in the original bit pattern will be lost and the rightmost n bit positions that are vacated will be filled with 0s.
  Example: **x** **:** 0100 1001 1100 1011
  x<<3 **:** 0100 0001 0101 1000
  This operator is used to multiply by power of two.
- **Right Shift (>>):** This operator is used to move bit patterns to the right. By executing **op>>n** causes all the bits in the operand op to be shifted to the right by n positions. The right most n bits in the original bit pattern will be lost and the leftmost n bit positions that are vacated will be filled with 0s.
  Example: **x** **:** 0100 1001 1100 1011
  x>>3 **:** 0000 1001 0011 1001
  This operator is used to divide by power of two.
- **Bitwise NOT (~):**The complement operator ~ is an unary operator and invets all the bits represented by its operand
  Example: if x = 13 and y = ~ x then the value of y will be 65522.
  x: 0000 0000 0000 1101
  y: 1111 1111 1111 0010

- **Conditional Operator:** The conditional operators ? and : are sometimes called ternary operators since they take three arguments.
  Syntax: Expression 1? Expression 2: Expression 3
  Example: y = x>4 ? 2 :7
  The meaning is if x is greater than 4 then 2 is assigned to y else 7 is assigned to y
- **Assignment Operator:** Assignment operators are used to assign the result of an expression to a variable. = is the normal assignment operator. C has a set of 'shorthand' assignment operators of the form: **v op = exp.** It is equivalent to **v = v op (exp).** The shorthand assignment operators are: +=, -=, *=, /=, %=, &=, ^=, |=, >>=, <<=.
  - The advantage of shorthand assignment operators are: it is easier to read/write and is more efficient because the value of v will be calculated/referred once.
- **sizeof Operator:** It is a compile time operator and, when used with an operand, it returns the number of bytes the operand occupies. The operand may be a variable, a constant or a data type qualifier.
  - This operator is normally used to determine the lengths of arrays, structures when their sizes are not known to the programmer.
  - It is also used to allocate memory dynamically.
  - **Example:** sizeof(a) will return the size of the variable a.
- **Comma Operator:** The comma operator can be used to link the related expressions together. A comma-linked list of operations are evaluated left to right and the value of right-most expression is the value of the combined expression

- **Example:** x = (a=5, b=10, a+b) assigns 5 to a, 10 to b and then add a & b & assigns to x.

Precedence of Operators:

- Each operator in C has a precedence which is used to determine how an expression involving more than one operator is evaluated.
- There are distinct levels of precedence and every operator may belong to one of the levels.
- The operators at the highest level of precedence are evaluated first. The operators of the same precedence are evaluated either from left to right or from right to left depending on the level. This property of the operators is known as associativity.
- The precedence of operators are:

Description	Operator	Associativity
Function Expression	( )	Left to right
Array Expression	[ ]	Left to right
Structure Operator	->	Left to right
Structure Operator	.	Left to right
Unary minus	-	Right to left
Increment/Decrement	++ --	Right to left
One's complement	~	Right to left
Negation	!	Right to left
Address of	&	Right to left
Value of address	*	Right to left
Type cast	(type)	Right to left
Size in bytes	Sizeof	Right to left
Multiplication	*	Left to right
Division	/	Left to right
Modulus	%	Left to right
Addition	+	Left to right
Subtraction	-	Left to right
Left Shift	<<	Left to right
Right Shift	>>	Left to right
Less than	<	Left to right
Less than or equal to	<=	Left to right
Greate than	>	Left to right
Greater than or equal to	>=	Left to right
Equal to	= =	Left to right
Not equal to	!=	Left to right
Bitwise AND	&	Left to right
Bitwise exclusive OR	^	Left to right
Bitwise inclusive OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	? :	Right to left
Assignment	=	Right to left

	*= /= %= += /= &= ^= |= <<= >>=	Right to left Right to left Right to left Right to left
Comma	,	Right to left