Character Set in C

The character set in C Language can be grouped into the following categories.
1. Letters
2. Digits
3. Special Characters
4. White Spaces

Tokens

A C program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol.

Keywords and Identifiers

Every word in C language is a keyword or an identifier.Keywords in C language cannot be used as a variable name. They are specifically used by the compiler for its own purpose .They serve as building blocks of a c program.

The following are the Keyword set of C language.

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

Data types in C

A C language programmer has to tell the system before-hand, the type of numbers or characters he is using in his program. These are data types.
C language data types can be broadly classified as

- **Primary data type**
- **Derived data type**
- **User-defined data type**

All C Compilers accept the following fundamental data types
1. Integer(int)
2. Character (char)
3. Floating Point (float)
4. Double precision floating point (double)

The basic data types can be augmented by data type qualifiers short,long,signed ,unsigned
Eg:-short int or unsigned int

Variables in C

A variable is a named memory location.It is an valid identifier.It is a series of characters consisting of letters digits and underscores .A variable doesn't begin with a digit or a special character.C is case sensitive hence..variable a and A are different.A keyword can't be used for naming a variable.
Eg of valid identifiers :- a,a1,net_pay etc

Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. 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
- Misc Operators

Arithmetic Operators

Addition (+)
Subtraction (–)
Multiplication (*)
Division(/)
Remainder(%)
The arithemtic operators are all binary operators.Integer division yields an integer result
Eg:- expression 7/4 evaluates to 1 and 17/5 evaluates to 3
The remainder operator yields remainder of the integer division.
It can only be used with integer operands.
Thus 7%4 evaluates to 3 and 17%5 evaluates to 2

Relational operators

< - less than
> -greater than
<= - less than or equal to
>= - greater than or equal to
== - equal to
!= - not equal to

Examples
i,j,k are integer variables having values 1,2,3 respectively

Expression	Intrepretation	Value
i<j	True	1
(i+j)>=k	True	1
k!=3	False	0
j==2	True	1

The value of the expression is non-zero if its true,zero if its false

Logical Operators

C allows usage of 3 logical operators

1. &&(and)
2. || (or)
3. !(Not)

Assume variable a=1 and variable **b=** 0, then:

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.

The first two operators,&& and || allow two or more conditions to combined in an if statement.
Eg:- if((marks>=50) &&(marks<60))
{ printf(“second class”);}

Eg:-

```
printf("Enter a number between 1 and 100");
scanf("%d", &number );
 if( (number < 1) || (number > 100) ) printf("Number is outside legal range\n");
```

Increment & Decrement Operators
One of the most common addition assignments is:
x = x + 1;
We learned that this statement increases the value of x by 1.
That is, the statement takes the old value of x, adds 1 to it, then assigns the resulting sum to x.
For this statement, we could use the addition assignment operator:
x += 1;

x = x + 1;
For this statement, we could use the addition assignment operator:
x += 1;
But this statement is MUCH more common than
x += y;
for generic y, so the C language has another special operator, called the *increment operator*:
x++;
x = x + 1;
x += 1;
Increment operator:
x++;
Also:
x = x – 1;
x -= 1;
x--;
This is known as the *decrement operator*.

This:	is identical to this:	is identical to this:	Name
x++;	x += 1;	x = x + 1;	Increment

x--;	x -= 1;	x = x – 1;	Decrement

Operator	Sample Expression	Explanation
++	++a	Increment a by 1 then use the new value of a in the exp in which a resides
++	a++	Use the current value of a in the exp in which a resides ,then increment a by 1
--	--b	Decrement b by 1, then use the new value of b in the exp in which b resides
--	b--	Use the current value of b in the exp in which b resides ,then decrement b by 1

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
%=	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

Misc Operators :- sizeof & ternary

There are few other important operators including **sizeof** and **? :** supported by C Language.

Operator	Description	Example
sizeof()	Returns the size of an variable.	sizeof(a), where a is integer, will return 2.
&	Returns the address of an variable.	&a; will give actual address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.

? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Operator Precedence & Associativity

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	**Operator**	**Associativity**
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right