INTRODUCTION

A programming language is designed to help certain kinds of *data process* consisting of numbers, characters and strings to provide useful output known as *information*. The task of processing of data is accomplished by executing a sequence of precise instructions called *program.*

CHARACTER SET

C characters are grouped into the following categories.

1. Letters
2. Digits
3. Special Characters
4. White Spaces

Note: The compiler ignores white spaces unless they are a part of a string constant.

Digits All decimal digits 0…..9

Letters

Uppercase A….Z

Lowercase a…..z

Special characters

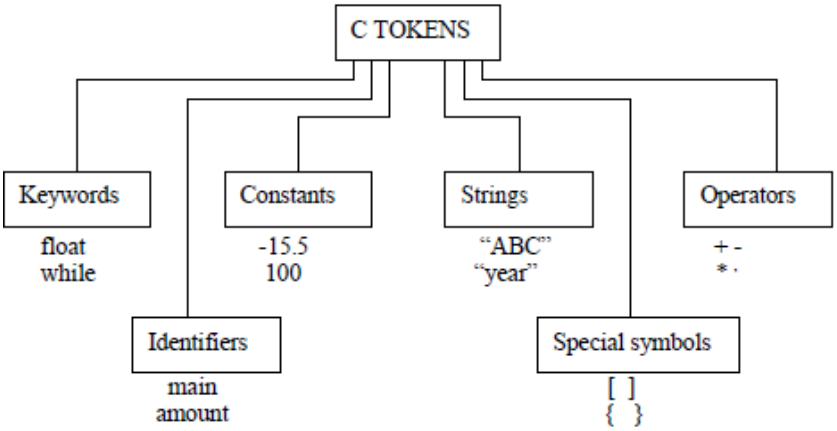
,	Comma	&	Ampersand
.	Period	^	Caret
;	Semicolon	*	Asterisk
:	Colon	-	Minus
?	Question mark	+	Plus sign
‘	Apostrophe	<	Less than
“	Quotation mark	>	Greater than
!	Exclamation	(	Left parenthesis
|	Vertical Bar	)	Right parentheses
/	Slash	[	Left bracket
\	Back slash	]	Right bracket
~	Tilde	{	Left brace
_	Underscore	}	Right brace
$	Dollar sign	#	Number sign
%	Percent sign		

White Spaces

- Blank Space
- Horizontal Tab
- Carriage Return
- New Line
- Form Feed

C TOKENS

In C programs, the smallest individual units are known as *tokens*.

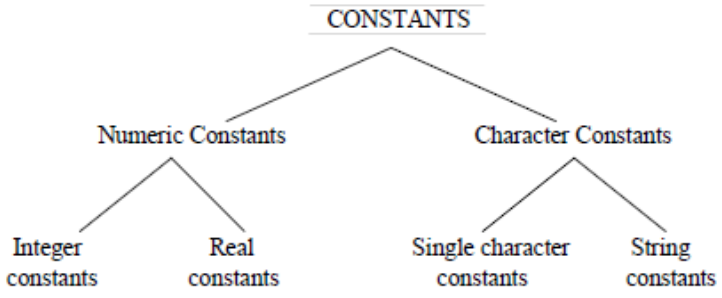


KEYWORDS AND IDENTIFIERS

Every C word is classified as either a *keyword* or an *identifier.* All keywords have fixed meanings and these meanings cannot be changed. Eg: auto, break, char, void etc., Identifiers refer to the names of variables, functions and arrays. They are user-defined names and consist of a sequence of letters and digits, with a letter as a first character. Both uppercase and lowercase letters are permitted. The underscore character is also permitted in identifiers.

Constants

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



Integer Constants

An integer constant refers to a sequence of digits, There are three types integers, namely, *decimal, octal*, and *hexa decimal*.

Decimal Constant

Eg:123,-321 etc.,

Note: Embedded spaces, commas and non-digit characters are **not** permitted between digits.

Eg: 1) 15 750 2)$1000

Octal Constant

An *octal* integer constant consists of any combination of digits from the set 0 through 7, with a leading 0.

Eg: 1) 037 2) 0435

Hexadecimal Constant

A sequence of digits preceded by 0x or 0X is considered as *hexadecimal* integer. They may also include alphabets A through F or a through f. Eg: 1) 0X2 2) 0x9F 3) 0Xbcd

Real Constants

Certain quantities that vary continuously, such as distances, heights etc., are represented by numbers containing functional parts like 17.548.Such numbers are called *real* (or *floating point)*constants. Eg:0.0083,-0.75 etc.,A real number may also be expressed in *exponential or scientific notation.* Eg:215.65 may be written as 2.1565e2

Single Character Constants

A single character constants contains a single character enclosed within a pair of *single* quote marks. Eg: '5' 'X' ';'

String Constants

A string constant is a sequence of characters enclosed in *double* quotes. The characters may be letters, numbers, special characters and blank space. Eg:"Hello!" "1987" "?….!"

Backslash Character Constants

C supports special backslash character constants that are used in output functions. These character combinations are known as *escape sequences.*

Constant	Meaning
'\a'	audible alert
'\b'	backspace
'\f'	form feed
'\n'	new line
'\0'	null
'\v'	vertical tab
'\t'	horizontal tab
'\r'	carriage return

VARIABLES

Definition:

A *variable* is a data name that may be used to store a data value. A variable may take different values at different times of execution and may be chosen by the programmer in a meaningful way. It may consist of letters, digits and underscore character.

Eg: 1) Average 2) Height

Rules for defining variables

v They must begin with a letter. Some systems permit underscore as the first character.

v ANSI standard recognizes a length of 31 characters. However, the length should not be normally more than eight characters.

v Uppercase and lowercase are significant.

v The variable name should not be a keyword.

v White space is not allowed.

DATA TYPES

ANSI C supports four classes of data types.

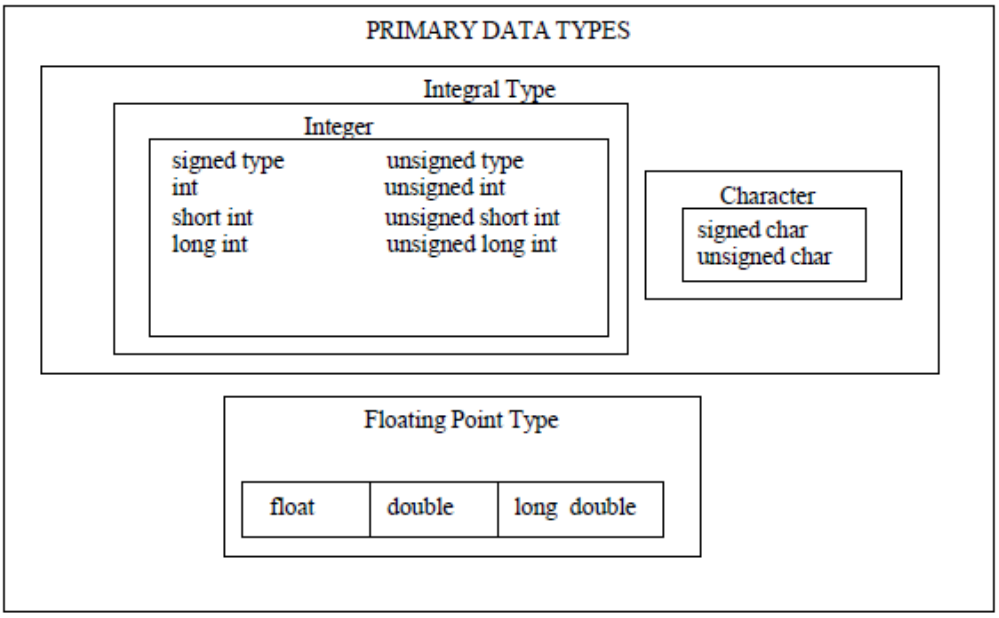
1. Primary or Fundamental data types.

2. User-defined data types.

3. Derived data types.

4. Empty data set.

PRIMARY DATA TYPES



Integer Types

Type	Size (bits)	Range
int or signed int	16	-32,768 to 32767
unsigned int	16	0 to 65535
short int	8	-128 to 127
unsigned short int	8	0 to 255
long int	32	-2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295

Floating Point Types

Type	Size(bits)	Range
float	32	3.4E-38 to 3.4E+38
double	64	1.7E-308 to 1.7E+308
long double	80	3.4E-4932to 1.1E+4932

Character Types

Type	Size (bits)	Range
char	8	-128 to 127
unsigned char	8	0 to 255

DECLARATION OF VARIABLES

The syntax is

Data-type v1,v2…..vn;

Eg:1.**int** count;

2.**double** ratio, total;

User-defined type declaration

C allows user to define an identifier that would represent an existing **int** data type.

The general form is

typedef type identifier;

Eg: 1) **typedef int** units;

2) **typedef float** marks;

Another user defined data types is enumerated data type which can be used to declare variables that can have one of the values enclosed within the braces.

enum identifier {value1,value2,……valuen};

Declaration of storage class

Variables in C can have not only data type but also storage class that provides information about their locality and visibility.

```
/*Example of storage class*/
int m;
main()
{
        int i;
        float bal;
        ……
        ……
        function1();
}
        function1()
        {
                int i;
```

```
float sum;
……
……
}
```

Here the variable **m** is called the global variable. It can be used in all the functions in the program. The variables **bal, sum and i** are called local variables. Local variables are visible and meaningful only inside the function in which they are declared. There are four storage class specifiers, namely, auto, static, register and extern.

ASSIGNING VALUES TO VARIABLES

The syntax is

```
Variable_name=constant
```

Eg:1) int a=20;

2) bal=75.84;

3) yes='x';

C permits multiple assignments in one line.

Example:

initial_value=0;final_value=100;

Declaring a variable as constant

Eg: 1) **const int** class_size=40;

This tells the compiler that the value of the int variable class_size must not be modified by the program.

Declaring a variable as volatile

By declaring a variable as volatile, its value may be changed at any time by some external source.

Eg:1) **volatile int** date;

READING DATA FROM KEYWORD

Another way of giving values to variables is to input data through keyboard using the **scanf** function.

The general format of **scanf** is as follows.

```
scanf("control string",&variable1,&variable2,....);
```

The ampersand symbol **&** before each variable name is an operator that specifies the variable name's *address*.

Eg: 1) **scanf("%d",&number);**

DEFINING SYMBOLIC CONSTANTS

We often use certain unique constants in a program. These constants may appear repeatedly in a number of places in the program. One example of such a constant is 3.142, representing the value of the mathematical constant **"pi"**.We face two problems in the subsequent use of such programs.

1. Problem in modification of the programs.
2. Problem in understanding the program.

A constant is defined as follows:

```
#define symbolic-name value of constant
```

Eg: 1) **#define pi** 3.1415

2) #**define** pass_mark 50

The following rules apply to a **#define** statement which define a symbolic constant

Symbolic names have the same form as variable names.

No blank space between the sign **'#'** and the word **define** is permitted

'#' must be the first character in the line.

A blank space is required between **#define** and *symbolic name* and between the *symbolic name* and the *constant.*

#define statements must not end with the semicolon.

After definition, the symbolic name should not be assigned any other value within the program by using an assignment statement.

v Symbolic names are NOT declared for data types. Their data types depend on the type of constant.

v **#define** statements may appear *anywhere* in the program but before it is referenced in the program.

OPERATORS

- arithmetic operators
- relational operators
- logical, assignment operators
- increment, decrement, conditional operators
- bitwise and special operators.

OPERATORS OF C

C supports a rich set of operators. Operators are used in programs to manipulate data and variables. They usually form a part of the mathematical of logical expressions. C operators are classified into a number of categories.

They include:

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment and Decrement operators
6. Conditional operators

7. Bitwise operators

8. Special operators

ARITHMETIC OPERATORS

The operators are

+ (Addition)

- (Subtraction)

* (Multiplication)

/ (Division)

% (Modulo division)

Eg: 1) a-b 2) a+b 3) a*b 4) p%q

The modulo division produces the remainder of an integer division.

The modulo division operator cannot be used on floating point data.

Note: C does not have any operator for *exponentiation*.

Integer Arithmetic

When both the operands in a single arithmetic expression are integers, the expression is called an *integer expression ,* and the operation is called *integer arithmetic.* During modulo division the sign of the result is always the sign of the first operand. That is

-14 % 3 = -2

-14 % -3 = -2

14 % -3 = 2

Real Arithmetic

An arithmetic operation involving only real operands is called *real arithmetic.* If **x and y** are floats then we will have:

1) x = 6.0 / 7.0 = 0.857143

2) y = 1.0 / 3.0 = 0.333333

The operator % cannot be used with real operands.

Mixed-mode Arithmetic

When one of the operands is real and the other is integer, the expression is called *a mixed-mode arithmetic* expression and its result is always a real number.

Eg: 1) 15 / 10.0 = 1.5

RELATIONAL OPERATORS

Comparisons can be done with the help of *relational operators.* The expression containing a relational operator is termed as a *relational expression.* The value of a relational expression is either *one* or *zero.*

1) < (is less than)

2) <= (is less than or equal to)

3)> (is greater than)

4) >= (is greater than or equal to)

5) = = (is equal to)

6) != (is not equal to)

LOGICAL OPERATORS

C has the following three *logical operators.*

&& (logical **AND**)

|| (logical **OR**)

! (logical **NOT**)

Eg: 1) if(age>55 && sal<1000)

2) if(number<0 || number>100)

ASSIGNMENT OPERATORS

The usual assignment operator is '='.In addition, C has a set of 'shorthand' assignment operators of the form, v **op** = exp;

Eg:1.x += y+1;

This is same as the statement

x=x+(y+1);

Shorthand Assignment Operators

Statement with shorthand operator	Statement with simple assignment operator
a + =1	a = a + 1
a - = 1	a = a – 1
a *= n + 1	a = a * (n+1)
a /= n + 1	a = a / (n+1)
a %= b	a = a % b

INCREMENT AND DECREMENT OPERATORS

C has two very useful operators that are not generally found in other languages. These are the *increment* and *decrement* operator:

++ and --

The operator ++ adds 1 to the operands while – subtracts 1.It takes the following form:

++m; or m++

--m; or m—

CONDITIONAL OPERATOR

A ternary operator pair "?:" is available in C to construct conditional expression of the form:

exp1 ? exp2 : exp3;

Here *exp1* is evaluated first. If it is true then the expression *exp2* is evaluated and becomes the value of the expression. If *exp1* is false then *exp3* is evaluated and its value becomes the value of the expression.

Eg:1) if(a>b)

x = a;

else

x = b;

BITWISE OPERATORS

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Shift left
>>	Shift right
~	One's complement

SPECIAL OPERATORS

C supports some special operators such as

- Comma operator
- Size of operator
- Pointer operators(& and *) and
- Member selection operators(. and ->)

The Comma Operator

The comma operator can be used to link the related expressions together. A commalinked list of expressions are evaluated *left* to *right* and the value of *right-most* expression is the value of the combined expression.

Eg: value = (x = 10, y = 5, x + y);

This statement first assigns the value 10 to **x**, then assigns 5 to **y,** and finally assigns 15(i.e, 10+5) to **value**.

The Size of Operator

The size of is a compiler time operator and, when used with an operand, it returns the number of bytes the operand occupies.

Eg: 1) m = **sizeof**(sum);

2) n = **sizeof(long int)**

3) k = **sizeof**(235L)

EXPRESSIONS

identify expressions
- understand the precedence of arithmetic operators
- know how type conversion works
- get knowledge about mathematical functions of C
- mange input and output operations of C

EXPRESSIONS

The combination of operators and operands is said to be an expression.

ARITHMETIC EXPRESSIONS

An arithmetic expression is a combination of variables, constants, and operators arranged as per the syntax of the language.

Eg 1) a = x + y;

EVALUATION OF EXPRESSIONS

Expressions are evaluated using an assignment statement of the form

variable = expression;

Eg:1) x = a * b – c;

2) y = b / c * a;

PRECEDENCE OF ARITHMETIC OPERATORS

An arithmetic expression without parenthesis will be evaluated from left to right using the rule of precedence of operators. There are two distinct priority levels of arithmetic operators in C.

High priority * / %

Low priority + -

Program

```
/*Evaluation of expressions*/
main()
{
float a, b, c, y, x, z;
a = 9;
b = 12;
c = 3;
x = a – b / 3 + c * 2 – 1;
y = a – b / (3 + c) * (2 – 1);
z = a – (b / (3 + c) * 2) – 1;
printf(“x = %f \n”,x);
printf(“y = %f \n”,y);
printf(“z = %f \n”,z);
}
```

OUTPUT

```
x = 10.000000
y = 7.000000
z = 4.000000
```

Mathematical Functions

Mathematical functions such as sqrt, cos, log etc., are the most frequently used ones. To use the mathematical functions in a C program, we should include the line

#include<math.h>

in the beginning of the program.

Function	Meaning
Trignometric	
acos(x)	Arc cosine of x
asin(x)	Arc sine of x
atan(x)	Arc tangent of x
atan2(x,y)	Arc tangent of x/y
cos(x)	cosine of x
sin(x)	sine of x
tan(x)	tangent of x
Hyperbolic	
cosh(x)	Hyperbolic cosine of x
sinh(x)	Hyperbolic sine of x
tanh(x)	Hyperbolic tangent of x
Other functions	
ceil(x)	x rounded up to the nearest integer
exp(x)	e to the power x
fabs(x)	absolute value of x
floor(x)	x rounded down to the nearest integer
fmod(x,y)	remainder of x/y
log(x)	natural log of x, x>0
log10(x)	base 10 log of x.x>0
pow(x,y)	x to the power y
sqrt(x)	square root of x,x>=0