Parul® University

Programming for Problem Solving **(PPS)**

Chapter-2

Constants, Variables, Data types, Operators and Expressions

Introducing C

- C program is developed by Dennis Ritchie in 1972 at Bell laboratory in USA.
- Designed for systems programming
  - Operating systems
  - Utility programs
  - Compilers
  - Filters
- Evolved from B, which evolved from BCPL

Parul® University

DIGITAL LEARNING CONTENT

Why C Language is Used

- Currently, the most commonly-used language for embedded systems is C language.
- Very portable: compilers exist for virtually every processor
- Easy-to-understand compilation
- Produces efficient code

Features of C

01

Portability:

- C Programs are portable i.e. they can be run on any Compiler with Little or no Modification.

02

Powerful:

- Provides Wide variety of '**Data Types**'
- Provides Wide variety of 'Functions'
- Provides useful Control & Loop Control Statements

03

Bit Manipulation :
C Programs can be manipulated using bits. We can perform different operations at bit level. We can manage memory representation at bit level.

Why C Language is Used

- **Effective use of pointers:**

  Pointers has direct access to memory.

  C Supports efficient use of pointer
- **Structured programming language**

  C is a structured programming language in the sense that **we can break the program into parts using functions.** So, it is easy to understand and modify.

Why C Language is Used

- **Memory Management**

  It supports the feature of **dynamic memory allocation.** In C language, we can free the allocated memory at any time by calling the **free()** function.

C Tokens

- C tokens are the basic buildings blocks in C language which are constructed together to write a C program.
- Each and every smallest individual units in a C program are known as C tokens.

Type of Tokens

Keywords

Example: float, double, integer, etc..

Identifiers

Example: main, amount, etc..

Constants

Example : 12.4, 3.14, etc..

Strings

Example : “Thursday”

Operators

Example : +,*,/,-

THE KEYWORDS

- "**Keywords**" are words that have special meaning to the **C** compiler.
- Their meaning cannot be changed at any instance.
- Serve as basic building blocks for program statements.
- All keywords are written in **only lowercase.**

THE KEYWORDS IN ANSI C

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

THE IDENTIFIERS

- Identifier are created to give unique name to C entities to identify it during the execution of program.

```
int money;
int mango_tree;
```

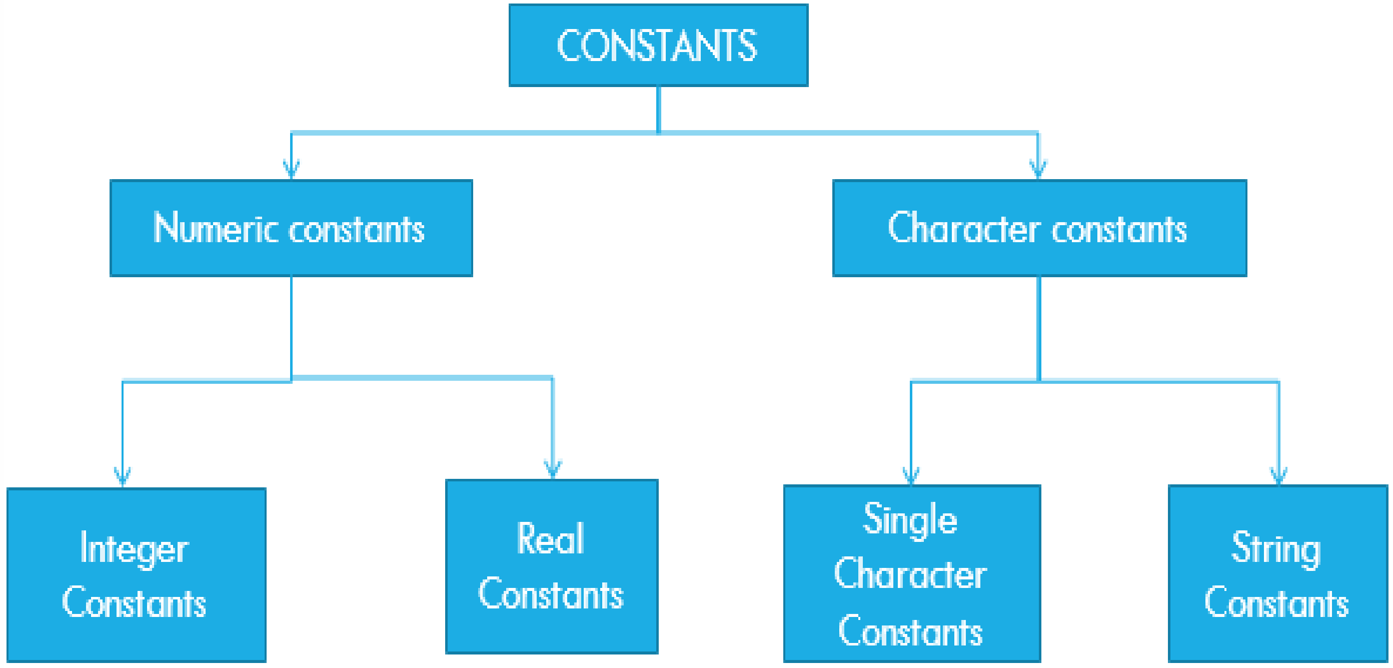
- Cannot use C keywords as identifiers
- Must begin with alpha character or _, followed by alpha, numeric, or _
- Upper- and lower-case characters are important (case-sensitive)
- Must consist of only letters, digits or underscore (_).
- Only first 31 characters are significant.
- Must NOT contain spaces (   ).

THE IDENTIFIERS - EXAMPLES

IDENTIFIER	VALID?	REASON IF INVALID
`totalSales`	Yes	
`total_Sales`	Yes	
`total.Sales`	No	Cannot contain .
`4thQtrSales`	No	Cannot begin with digit
`totalSale$`	No	Cannot contain $

CONSTANTS

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



CONSTANTS EXAMPLES

Integer Constants	Refers to sequence of digits such as decimal integer, octal integer and hexadecimal integer. Some of the examples are 112, 0551, 56579u, 0X2 etc.
Real Constants	The floating point constants such as 0.0083, -0.78, +67.89 etc.
Single Character Constants	A single char const contains a single character enclosed within pair of *single quotes* [‘ ’]. For example, ‘8’, ‘a’ , ‘i’ etc.
String Constants	A string constant is a sequence of characters enclosed in double quotes [“ ”]; For example, “0211”, “Stack Overflow” etc.

PROGRAM STRUCTURE

A sample C Program

```
#include<stdio.h>
#include<conio.h>
Void main()
{
   --other statements
}
```

HEADER FILES

- The files that are specified in the include section is called as header file
- These are precompiled files that has some functions defined in them
- We can call those functions in our program by supplying parameters
- Header file is given an extension .h
- C Source file is given an extension .c

MAIN FUNCTION

- This is the entry point of a program
- When a file is executed, the start point is the main function
- From main function the flow goes as per the programmers choice.
- There may or may not be other functions written by user in a program
- Main function is compulsory for any C program

RUNNING A ‘C’ PROGRAM

- Type a program
- Save it
- Compile the program – This will generate an exe file (executable)
- Run the program (Actually the exe created out of compilation will run and not the .c file)
- In different compiler we have different option for compiling and running. We give only the concepts.

DATE TYPES

- A data type is
  - A set of values AND
  - A set of operations on those values
- A data type is used to
  - Identify the type of a variable when the variable is declared
  - Identify the type of the return value of a function
  - Identify the type of a parameter expected by a function

DATE TYPES

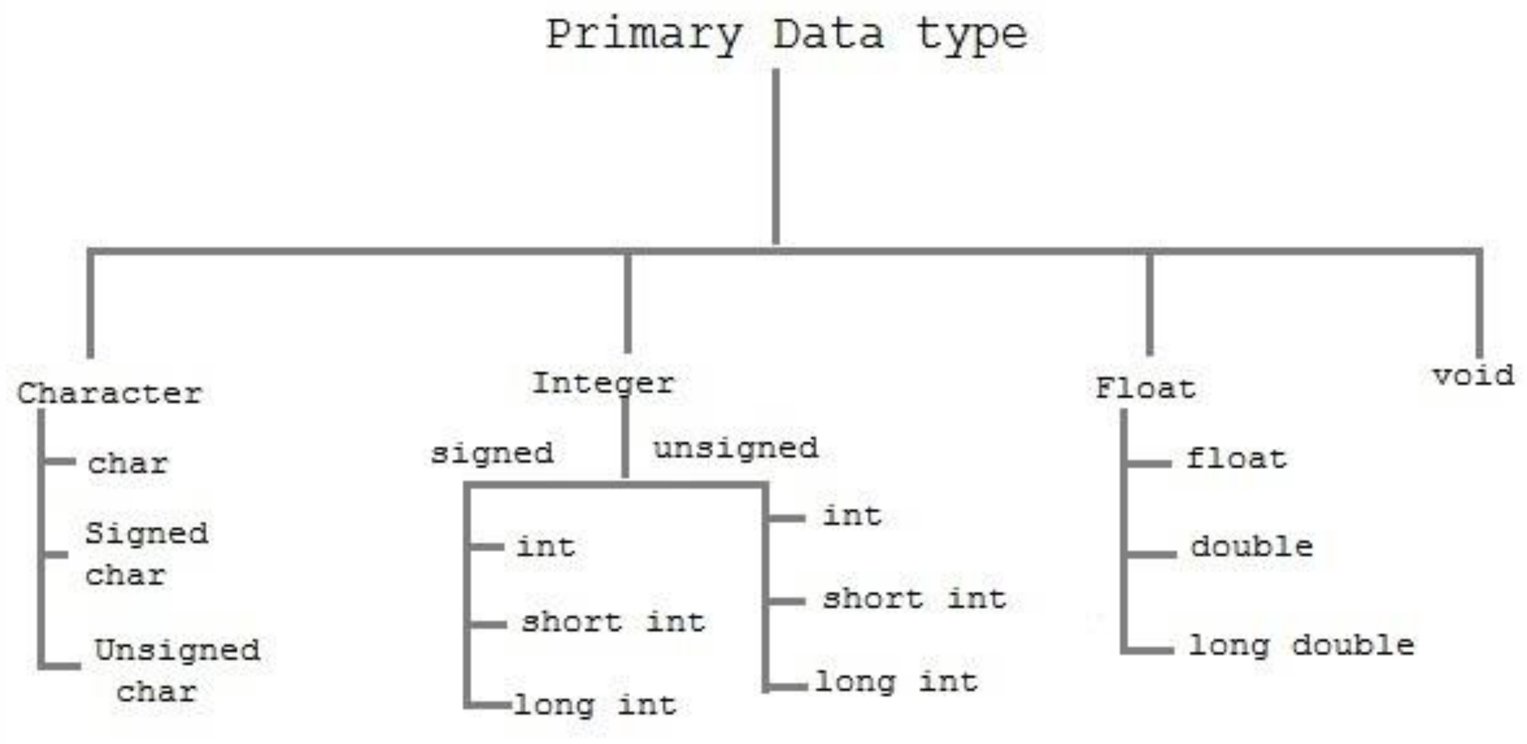
ANSI C supports three classes of data types.

1. Primary or Fundamental data types.
2. User-defined data types.
3. Derived data types.

Parul® University

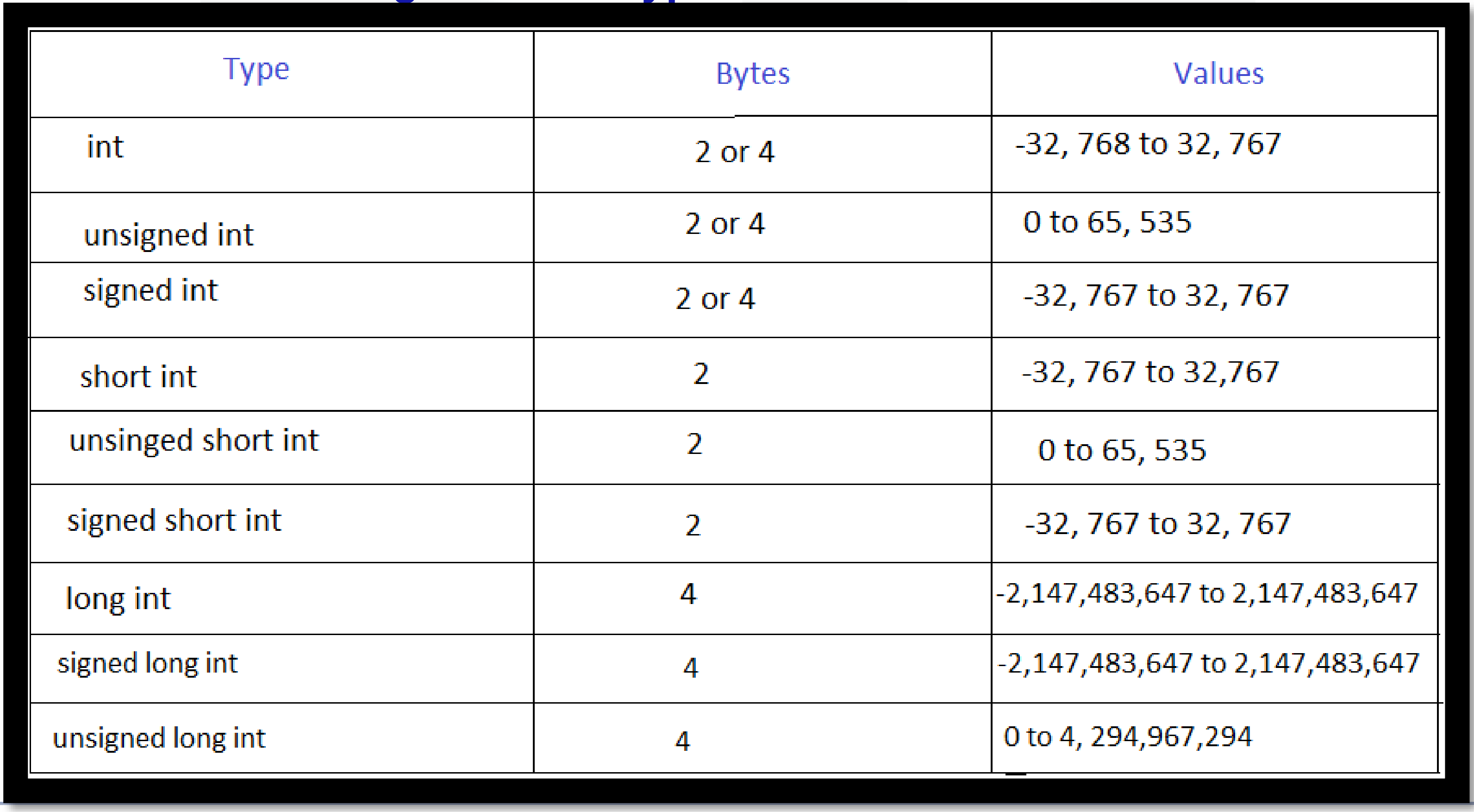
DIGITAL LEARNING CONTENT

PRIMARY DATA TYPES IN C



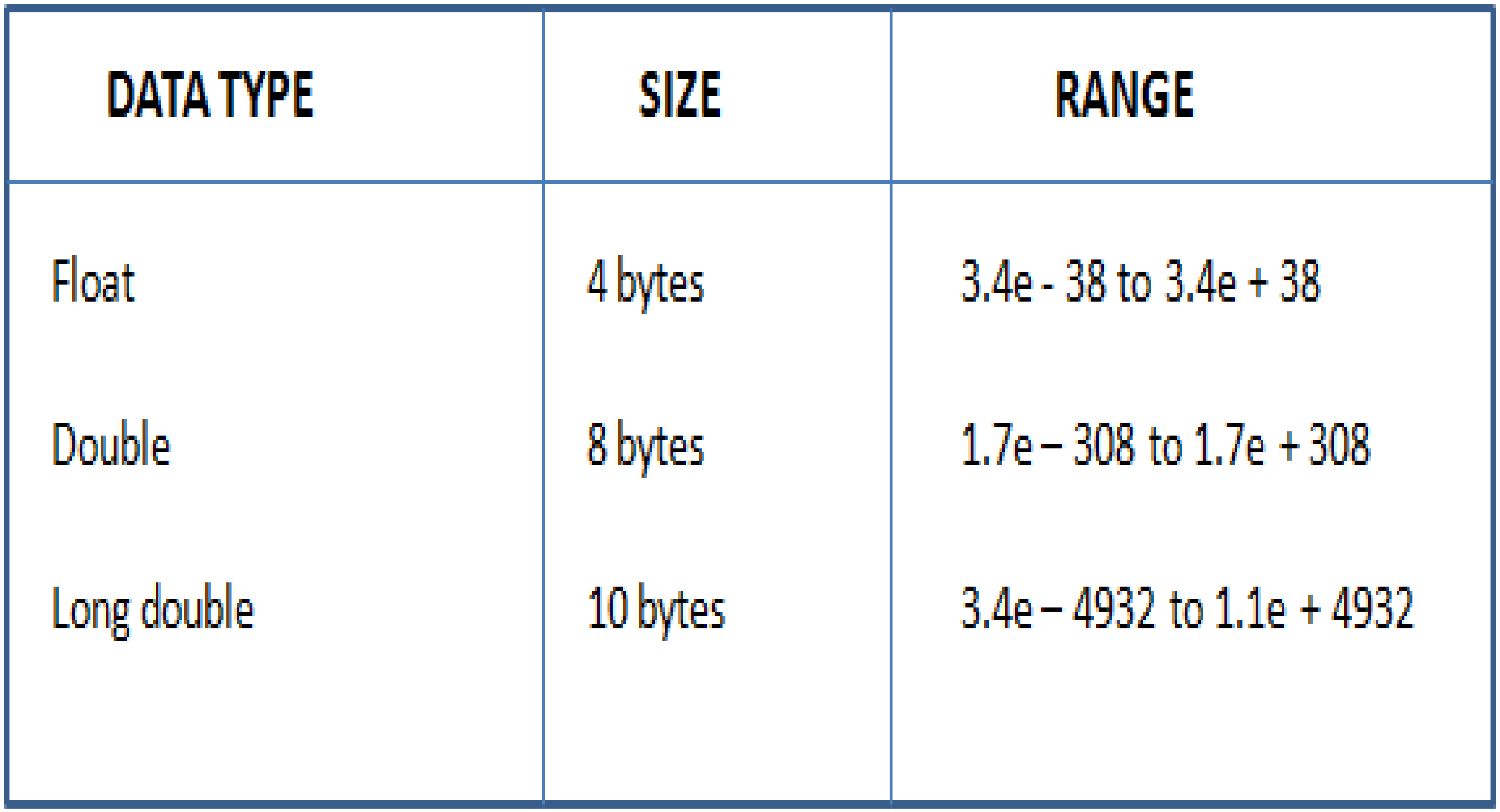
INTEGER TYPES

Size and Range Of Data types on 16 bit machine



Type	Bytes	Values
int	2 or 4	-32, 768 to 32, 767
unsigned int	2 or 4	0 to 65, 535
signed int	2 or 4	-32, 767 to 32, 767
short int	2	-32, 767 to 32,767
unsinged short int	2	0 to 65, 535
signed short int	2	-32, 767 to 32, 767
long int	4	-2,147,483,647 to 2,147,483,647
signed long int	4	-2,147,483,647 to 2,147,483,647
unsigned long int	4	0 to 4, 294,967,294

FLOATING POINT TYPES



DATA TYPE	SIZE	RANGE
Float	4 bytes	3.4e - 38 to 3.4e + 38
Double	8 bytes	1.7e – 308 to 1.7e + 308
Long double	10 bytes	3.4e – 4932 to 1.1e + 4932

Parul® University

DIGITAL LEARNING CONTENT

USER-DEFINED TYPE DECLARATION

- C allows user to define an identifier that would represent an existing **data type.**
- The general form is **typedef type identifier;**

Eg:

typedef int units;

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};**

DERIVED DATA TYPE

- C allows a different types of derived data structure
- Different types of datatypes are
- array
- Functions
- Pointer
- Structure

DECLARATION OF VARIABLES

DECLARATION OF VARIABLES

- **Declarations does two things:**
- ➢ It tells the compiler what the variable name is
- ➢ It specifies what type of data the variable will hold

- ➢ **Primary Type Declaration**
- The syntax is
- **Data-type v1,v2…..vn;**

Eg:

int count;

double ratio, total;

USER-DEFINED TYPE DECLARATION

- C allows user to define an identifier that would represent an existing **data type.**
- The general form is **typedef type identifier;**

Eg:

typedef int units;

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};**

USER-DEFINED TYPE DECLARATION

Declaring a variable as constant

Eg:

```
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.

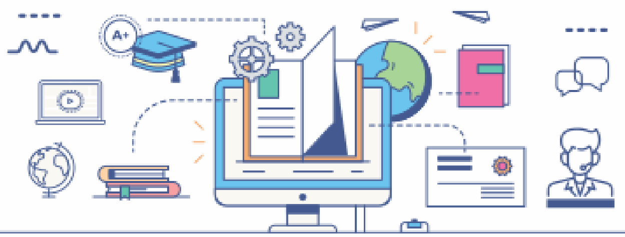
Declaring a variable as volatile

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

Eg:

```
volatile int date;
```

OPERATORS IN C



OPERATORS IN C

Operators are the verbs of a language that help the user perform computations on values.

C language supports a rich set of operators.

Different types of operators in C are :

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

These are used to perform arithmetic operations.

All of these can be used as binary operators. These are :

+	add
-	subtract
*	multiply
/	divide(divisor must be non zero)
%	modulo(gives remainder after div)

The parenthesis() are used to clarify complex operations.

The operators + and - cab be used as unary plus and unary minus arithmetic operators also. The unary – negates the sign of it's operand .

ARITHMETIC OPERATORS

Note : C language has no operator for exponentiation.

The function pow(x,y) which exists in math.h returns

X^y

Following are some examples of arithmetic operators :

x+y, x-y, x*y, x/y, x%y, -x*y

Here x and y are operands. The % operator cannot be used on floating point data type.

ARITHMETIC OPERATORS

An expression consisting of numerical values(either any number, variable or even some function call) joined together by arithmetic operators is known as an arithmetic expression. For example , consider the following expression :

(x-y)*(x+y)/5

Here x,y and 5 are operands and the symbols -,*,+,/ are operators.

The precedence of operators for the expression evaluation has been given by using parenthesis which will over rule the operators precedence. If x=25 and y=15,then the value of this expression will be 80.

ARITHMETIC OPERATORS

Consider the following expression :

3*((i%4)*(5+(j-2)/(k+3)))

Where i, j, k are integer variables. If i, j, k have values 9 ,14 and 16 respectively then above expression would be evaluated as

3 * ((9%4) * (5 + (14 – 2) / (6 +3)))

= 3 * (1 * (5 + (12 / 9)))

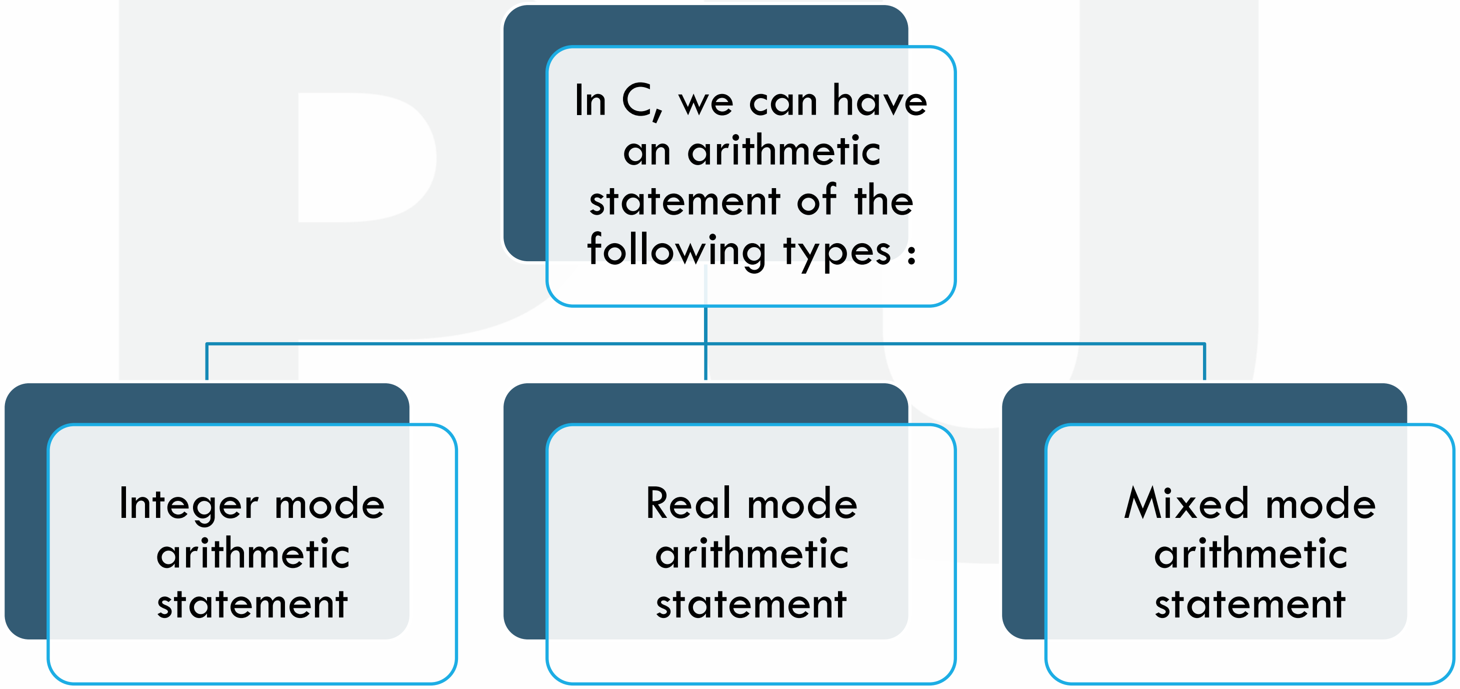
= 3 * (1 * (5 + 1))

= 3 * (1 * 6)

= 3 * 6

= 18

MODES OF OPERATION



In C, we can have an arithmetic statement of the following types :

- Integer mode arithmetic statement
- Real mode arithmetic statement
- Mixed mode arithmetic statement

INTEGER MODE ARITHMETIC STATEMENT

It consist of all operands as either integer constants or integer variables. For example ,

```
 int  s, y, l, j, result ;
x=50;
y=x+10;
i=2;
j=i*3;
result=x/7 – y%j + i;
```

In case of integer division, result is turnicated towards zero when both the operands are of the same sign. When one of the operands is negatic the turnication depends upon the implementation. For example :

INTEGER MODE ARITHMETIC STATEMENT

3/5 = 0 and -3/-5 =0

But -3/5 can result into 0 or -1 (it is machine dependent)

For modulo division,the sign of result is always that of the first operand or dividend. For example,

13 % - 5 = 3

- 13 % - 5 = - 3

- 13 % 5 = - 3

INTEGER MODE ARITHMETIC STATEMENT

```
/*sum of 4 digit number*/
#include<stdio.h>
main()
{int num,digit,sum=0;
clrscr();
printf("Enter 4 digit no");
scanf("%d", &num);
digit=num%10;
sum=sum+digit;
num=num/10;
digit=num%10;
sum=sum+digit;
num=num/10;
digit=num%10;
sum=sum+digit;
num=num/10;
digit=num%10;
sum=sum+digit;
printf("sum is %d",sum);
getch();}
```

Output

```
----------
Enter 4 digit no
3275
Sum is 17
```

REAL MODE ARITHMETIC STATEMENT

It consist of all operands as either real variables or real constants

For example,

```
float   a,b,d;
a=5.0;
b=a*20.5;
d=a / 2.0 * b – 10.0;
```

In C, real operand may store value in either of the two forms i.e., in fractional form or exponential form. The rounding takes place in such cases and the final result of an expression in real mode is alwys an approximation of the calculation performed. We cannot use % operator with real operands.

REAL MODE ARITHMETIC STATEMENT

The following program illustrates the use of real mode arithmetic :

```
/*conversion of centigrade temperature to fahrenheit */
#include<stdio.h>
main()
{float cent,fahr;
clrscr();
printf("Enter centegrade ");
scanf("%f",&cent);
fahr=1.8*cent+32.0;
printf("Centegrade temp is : %f",cent);
printf("fahr temp is : %f",fahr); }
```

MIXED MODE ARITHMETIC STATEMENT

It consists of operands of both types i.e., integers and reals. When any one of the two operands is real, the expression is known as mixed mode arithmetic expression and the operation performed in such a case results in a real value as the answer. For example, consider the following division :

24/10.0=24

If we write 24/10 then the result will be 2 as seen earlier.

The following program illustrates the use of mixed mode arithmetic :

MIXED MODE ARITHMETIC STATEMENT

```
/*sum and average of three numbers */
#include<stdio.h>
main()
{int num1,num2,num3,sum=0;
float avg=0.0;
printf("enter three numbers");
scanf("%d%d%d",&num1,&num2,&num3);
sum=num1+num2+num3;
avg=sum/3.0;
printf("sum is %d:",sum);
printf("average is %f",avg);
}
```

MIXED MODE ARITHMETIC STATEMENT

```
/*illustration of arithmetic operations on char data type */
#include<stdio.h>
main()
{   int I,j;
    char ch1,ch2;
    clrscr();
    I=10;
    ch1=‘A’; /* ASCII value of A is 65 */
    j=ch1-I;
    ch2=j+42;
    printf(“%d%d %c %c”,I,j,ch1,ch2);
}  output :  10   55   A   a
```

ARITHMETIC OPERATORS PRECEDENE

In C, the arithmetic operators have the priority as shown below:

First priority	* ? %
Second priority	+ -
Third priority	=

The sequence of operations in evaluating an arthmetic expression is also known as hierarchy of operations. This is necessary to avoid anydoubt while evaluating an expression. The following precedence rules are followed in expression evaluation :

ARITHMETIC OPERATORS PRECEDENE

(i) All the subexpressions within the parentheses are evaluated first. Nested parenthesized subexpressions are evaluated inside-out, with the innermost expression being first to be evaluated.

(ii) Operators in the same sub expression are evaluated as given : *, / ,% perform first +, - performed next. Any function referenced (i.e.,invoked)in the expression gets the highest precedence over all the arithmetic operators.

(iii) Operators in the same expression with the same priority are evaluated from left to right.

ARITHMETIC OPERATORS PRECEDENE

For example : consider the following expression for checking the operators precedence.

15 * 7 / (2 – 3 * 5 / 7 + 4) – 7 * 9 % 4

= 15 * 7 / (2 – 15 / 7 + 4) – 7 * 9 % 4

= 15 * 7 / (2 – 2 + 4) – 7 * 9 % 4

= 15 * 7 / 4 – 7 * 9 % 4

= 105 / 4 - 63 % 4

= 26 – 3

= 23

INCREMENT AND DECREMENT OPERATOR

C language has two useful operators called increment(++) and decrement (--) that operate on integer data only.

The increment (++) operator increments the operand by 1, while the decrement operator (--) decrements the operand by 1, for example ,:

```
int  i  , j;
i = 10;
j = i++ ;
printf(“ %d  %d “,  i, j);
```

OUTPUT

11   10 . First i is assigned to j and then i is incremented by 1 i.e.,post-increment takes place

INCREMENT AND DECREMENT OPERATOR

If we have :

```
int  i,  j ;
i  =  20;
j =  ++i;
printf("%d %d", i,  j);
```

OUTPUT : 21 21. first i is incremented by 1 and then assignment take place i.e., pre-increment of i.

now, consider the example for (--) operator :

```
int  a,b;
a=10;
b= a--;
printf("%d %d", a  , b)
```

OUTPUT : 9 10. first *a* is assigned to *b* then *a* is decremented by 1. i.e.,post decrement takes place

INCREMENT AND DECREMENT OPERATOR

If we have :

```
int  i,  j ;
I  =  20;
j =  --i;
printf("%d %d", i,  j);
```

OUTPUT :   19   19.   first i is decremented by 1 and then assignment take place i.e., pre-decrement of i.

Note : on some compilers a space is required on both sides of ++I or i++ , i--   or --i

RELATIONAL OPERATOR

These are used to compare two variables or constants . C has the following relational operators :

OPERATOR	MEANING
==	Equals
!=	Not Equals
<	Less than
>	Greater than
<=	Less than or equals
>=	Greater than or equals

LOGICAL OPERATORS

In C, we can have simple conditions (single) or compound conditions(two or more). The logical operators are used to combine conditions. The notations for these operators is given below :

Operator	Notation in C
NOT	!
AND	&&
OR	||

The notation for the operator OR is given by two broken lines. These follow the same precedence as in other language. NOT(!) is evaluated before AND(&&) which is evaluated before OR(||). Parenthesis( ) cab be used to change this order.

PRECEDENCE OF RELATIONAL OPERATORS AND LOGICAL OPERATORS

Each operator in C has a precedence of its own. It helps in evaluation of an expression. Higher the precedence of the operator, earlier it operates. The operators having same precedence are evaluated either from left to right or from right to left, depending on the level, known as the associativity of the operator.

! , < , <= , > , >=, ==, !=, ==, !=, &&, ||

THE CONDITIONAL OPERATOR

This operator ? And : together forms a ternary operator called as the conditional operator.

Syntax : (test-expression) ? T-expr : F-expr ;

Let us see example program :

THE CONDITIONAL OPERATOR

```
#include<stdio.h>
main()
{
int age;
clrscr();
printf)("Enter ur age");
scanf("%d",&age);
(age>18)? Printf("Eligible to vote"):printf("nt eligible");
getch();
}
```

BITWISE OPERATORS

These are used to perform bitwise operations such as testing the bits, shifting the bits to left to right, one's complement of bits etc. these operations can be applied only on int and char data types but not on float and double data types. Various bitwise operators in C language are :

~	Bitwise (1's) complement)
<<	shift left
>>	shift right
&	bitwise AND
^	bitwise XOR(Exclusive OR)
|	bitwise OR

SPECIAL OPERATORS

C provides the following special operators

- Comma Operator
- sizeof operator
- Address operator
- Dereferencing operator
- Dot operator
- Member selection operator
- Pointer

THE COMMA OPERATOR

The comma operator (,) has the lowest precedence.

The comma operator is mainly used in for statement. For example :

```
int  i , j;
for(i=1 ,  j=400  ; i<=10 ; ++I , j/=2)
printf("%d\n", i+j ) ;
```

The initial value of i is 1 and that of j is 400 and every time the value of i is incremented by 1 and that of j is divided by 2 after execution of the body of the for loop .

The distinct expression on either side of the comma operator are evaluated from left to right.

The associativity of comma operator is from left to right .

THE SIZEOF OPERATOR

It is a unary operator which provides the size , in bytes, of the given operand. The syntax of sizeof operator is :

```
sizeof(operand)
```

Here the operand is a built in or user defined data type or variable.

The sizeof operator always precedes its operand.

For example,   sizeof (float) returns the value 4 .

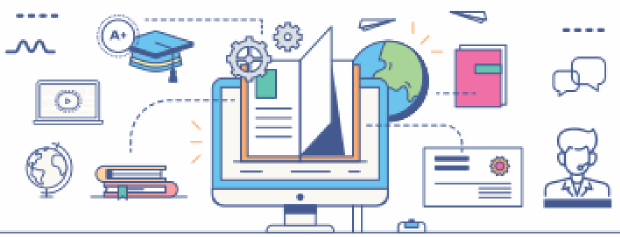
The sizeof operator mainly used in dynamic memory allocation for calculating the number of bytes used by some user defined data type.

Precedence of operators among themselves and across all sets of operators

The TURBO C operators are divided into the following 16 categories : these are ordered from the highest precedence to the lowest precedence. The operation within each category have equal precedence.

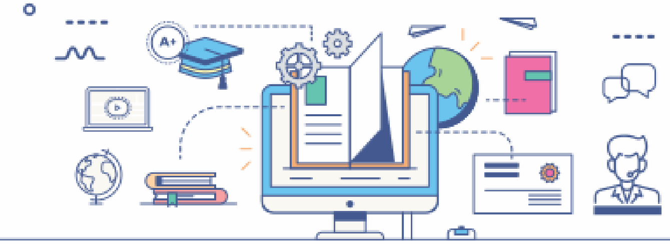
Precedence of operators among themselves and across all sets of operators

Category	Operator	What it does ?
1. Highest precedence	()	Function call
	[]	Array subscript
	->	C indirect component selector
2.Unary	!	NOT
	~	Bitwise(1's) component
	+	Unary plus
	-	Unary minus
3.Member acces	.*	Dereference
	->*	Dereference
4.Multiplication	*	Multiply
	/	Divide
	%	Remainder (Modulus)



Precedence of operators among themselves and across all sets of operators

Category	Operator	What it does ?
5.Additive	+	Binary plus
	-	Binary minus
6.Shift	<<	Shift left
	>>	Shift right
7.Relational	<	Less than
	<=	Less than or equal to
	>	Greater than
	>=	Greater than equal to
8.Equality	==	Equal to
	!=	Not equal to
9.Bitwise AND	&	Bitwise AND
10.Bitwise XOR	^	Bitwise XOR
11.Bitwise OR	|	Bitwise OR



Precedence of operators among themselves and across all sets of operators

Category	Operator	What it does ?
12.Logical AND	&&	Logical AND
13.Logical OR	||	Logical OR
14.Conditional	?:	(exp?x:y)
15.Assignment	=	Simple assignment
	*=	Assign product
	/=	Assign quotient
	%=	Assign remainder (modulus)
	+=	Assign sum
	-=	Assign difference
	&=	Assign bitwise AND
	^=	Assign bitwise XOR
	|=	Assign bitwise OR
	<<=	Assign left shift
	>>=	Assign right shift
16.Comma	,	Evaluate

THE ASSOCIATIVITY OF OPERATORS

In C , the operators having the equal precedence are evaluated either from left to right or from right to left, depending on the level. It is known as associativity property of the operator.

THE ASSOCIATIVITY OF OPERATORS

Category	Operator	Associativity
1.Highest precedence	( )	Left to Right
	[ ]	
	->	
	::	
	.	
2.Unary	!	Right to left
	~	
	+	
	-	
	++	
	--	
	&	
	*	
	sizeof	

THE ASSOCIATIVITY OF OPERATORS

Category	Operator	Associativity
3.Member access	.*	Left to Right
	->*	
4.Multiplication	*	Left to right
	/	
	%	
5.Additive	+	Left to Right
	-	
6.Shift	<<	Left to Right
	>>	
7.Relational	<	Left to Right
	<=	
	>	
	>=	

THE ASSOCIATIVITY OF OPERATORS

Category	Operator	Associativity
8.Equality	==	Left to Right
	!=	
9.Bitwise AND	&	Left to right
10.Bitwise XOR	^	Left to Right
11.Bitwise OR	|	Left to Right
12.Logical AND	&&	Left to Right
13.Logical OR	| |	Left to Right
14.Conditional	?:	Right to Left
15.Assignment	=	Right to Left
	*=	
	/=	
	%=	
	+=	
	-=	

DIGITAL LEARNING CONTENT

THE ASSOCIATIVITY OF OPERATORS

Category	Operator	Associativity
	&=	Right to Left
	^=	
	|=	
	<<=	
	>>=	
16.Comma	.	Left to Right