C FUNDAMENTALS

The C programming language is a structure oriented programming language developed at AT & T's Bell Laboratories of USA in 1972

It was designed and written by a man named Dennis Ritchie

ADVANTAGE:

- Major parts of popular operating systems like Windows, UNIX, Linux is still written in C. This is because even today when it comes to performance (speed of execution) nothing beats C.

- To write device driver programs.

- Mobile devices like cellular phones and palmtops are becoming increasingly popular. Also, common consumer devices like microwave oven, washing machines and digital cameras are getting smarter by the day

- Several professional 3D computer games

USES OF C PROGRAMMING LANGUAGE:

- Database systems
- Graphics packages
- Word processors
- Spreadsheets
- Operating system development
- Compilers and Assemblers
- Network drivers
- Interpreters

THE C CHARACTER SET

A character denotes any alphabet, digit or special symbol used to represent information.

Alphabets	A, B, ….., Y, Z a, b, ……, y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /

CONSTANTS

A constant is an entity that doesn’t change whereas a variable is an entity that may change

X=3
X=5

Types of C Constants

C constants can be divided into two major categories:

(a) Primary Constants
(1)Integer Constant
(2)Real Constant
(3)Character Constant

(b) Secondary Constants
(1)Array
(2)Pointer
(3)Structure
(4)Union
(5)Enum,etc.

Rules for Constructing Integer Constants

(a) An integer constant must not have a decimal point.

(b) It can be either positive or negative.

(c) If no sign precedes an integer constant it is assumed to be positive.

(d) No commas or blanks are allowed within an integer constant.

(e) The allowable range for integer constants is -32768 to 32767

(f) It must have at least one digit

Ex.: 426
+782
-8000
-7605

Rules for Constructing Real Constants

Real constants are often called Floating Point constants. The real constants could be written in two forms—Fractional form and Exponential form.

Following rules must be observed while constructing real constants expressed in fractional form:

(a) A real constant must have at least one digit.
(b) It must have a decimal point.
(c) It could be either positive or negative.
(d) Default sign is positive.
(e) No commas or blanks are allowed within a real constant.

Ex.: +325.34
426.0
-32.76
-48.5792

Following rules must be observed while constructing real constants expressed in exponential form:

(a) The mantissa part and the exponential part should be separated by a letter e.
(b) The mantissa part may have a positive or negative sign.
(c) Default sign of mantissa part is positive.
(d) The exponent must have at least one digit, which must be a positive or negative integer. Default sign is positive.
(e) Range of real constants expressed in exponential form is -3.4e38 to 3.4e38.

Ex.: +3.2e-5
4.1e8
-0.2e+3
-3.2e-5

Rules for Constructing Character Constants

(1) A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas. Both the inverted commas should point to the left. For example, ’A’ is a valid character constant whereas ‘A’ is not.

(2) The maximum length of a character constant can be 1 character.

Ex.: 'A'
'I'
'5'
'='

Types of C Variables

An integer variable can hold only an integer constant, a real variable can hold only a real constant and a character variable can hold only a character constant.

Rules for Constructing Variable Names

(a) A variable name is any combination alphabets, digits or underscores

(b) The first character in the variable name must be an alphabet or underscore.

(c) No commas or blanks are allowed within a variable name

(d) No special symbol other than an underscore can be used in a variable name.

Ex.: si_int
m_hra
pop_e_89

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

The First C Program

```
/* Calculation of simple interest */
#include<stdio.h>
int main( )
{
int p, n ;
float r, si ;
p = 1000 ;
n = 3 ;
r = 8.5 ;
/* formula for simple interest */
si = p * n * r / 100 ;
printf ( "%f" , si ) ;
return 0;
}
```

- Comment about the program should be enclosed within /* */.
- Though comments are not necessary, it is a good practice
- Any number of comments can be written at any place in the program.
  ```
  /* formula */ si = p * n * r / 100 ;
  si = p * n * r / 100 ; /* formula */
  si = p * n * r / /* formula */ 100 ;
  ```
- Comments cannot be nested. For example,
  ```
  /* Cal of SI /* Author date 01/01/2018 */ */
  ```
  is invalid
- A comment can be split over more than one line, as in,
  ```
  /* This is
  a jazzy
  ```

comment */

- Next is the preprocessor command that includes standard input output header file(stdio.h) from the C library before compiling a C program

- main( ) is a collective name given to a set of statements
```
main( )
{
statement 1 ;
statement 2 ;
statement 3 ;
}
```
- Any variable used in the program must be declared before using it.
```
int p, n ;
float r, si ;
```
- Any C statement always ends with a ;
- The general form of printf( ) function is
  printf ("<format string>", <list of variables>) ;
  <format string> can contain,
  %f for printing real values
  %d for printing integer values
  %c for printing character values
  Following are some examples of usage of printf( ) function:
```
printf ( "%f", si ) ;
printf ( "%d %d %f %f", p, n, r, si ) ;
printf ( "Simple interest = Rs. %f", si ) ;
printf ( "Prin = %d \nRate = %f", p, r ) ;
```
- printf ("%d %d %d %d", 3, 3 + 2, c, a + b * c – d) ;
- return 0 command terminates C program (main function) and returns 0.

Receiving Input

```
main( )
{
int p, n ;
float r, si ;
printf ( "Enter values of p, n, r" ) ;
scanf ( "%d %d %f", &p, &n, &r ) ;
si = p * n * r / 100 ;
printf ( "%f" , si ) ;
}
```

Ex.: The three values separated by blank
1000 5 15.5
Ex.: The three values separated by tab.
1000 5 15.5
Ex.: The three values separated by newline.
1000
5
15.5

INTEGER AND FLOAT CONVERSIONS

(a) An arithmetic operation between an integer and integer always yields an integer result.

(b) An operation between a real and real always yields a real result.

(c) An operation between an integer and real always yields a real result. In this operation the integer is first promoted to a real and then the operation is performed. Hence the result is real.

Operation	Result	Operation	Result
5 / 2	2	2 / 5	0
5.0 / 2	2.5	2.0 / 5	0.4
5 / 2.0	2.5	2 / 5.0	0.4
5.0 / 2.0	2.5	2.0 / 5.0	0.4

Type Conversion in Assignments (Assume k is integer, a is real)

Arithmetic Instruction	Result	Arithmetic Instruction	Result
k = 2 / 9	0	a = 2 / 9	0.0
k = 2.0 / 9	0	a = 2.0 / 9	0.2222
k = 2 / 9.0	0	a = 2 / 9.0	0.2222
k = 2.0 / 9.0	0	a = 2.0 /9.0	0.2222
k = 9 / 2	4	a = 9 / 2	4.0
k = 9.0 / 2	4	a = 9.0 / 2	4.5
k = 9 / 2.0	4	a = 9 / 2.0	4.5
k = 9.0 / 2.0	4	a = 9.0 / 2.0	4.5

```
#include <stdio.h>
int main()
{
  int a;
  char b;
  float c;
  double d;
  printf("Storage size for int data type:%d \n",sizeof(a));
  printf("Storage size for char data type:%d \n",sizeof(b));
  printf("Storage size for float data type:%d \n",sizeof(c));
  printf("Storage size for double data type:%d\n",sizeof(d));
  return 0;
}
Storage size for int data type:4
Storage size for char data type:1
Storage size for float data type:4
Storage size for double data type:8

Process returned 0 (0x0)   execution time : 0.008 s
Press any key to continue.
```

MODIFIERS IN C LANGUAGE

The amount of memory space to be allocated for a variable is derived by modifiers.

C Data types / storage Size	**Range**
char / 1	–127 to 127
int / 2	–32,767 to 32,767
float / 4	1E–37 to 1E+37 with six digits of precision
double / 8	1E–37 to 1E+37 with ten digits of precision
long double / 10	1E–37 to 1E+37 with ten digits of precision

long int / 4	−2,147,483,647 to 2,147,483,647
short int / 2	−32,767 to 32,767
unsigned short int / 2	0 to 65,535
signed short int / 2	−32,767 to 32,767
long long int / 8	−(2power(63) −1) to 2(power)63 −1
signed long int / 4	−2,147,483,647 to 2,147,483,647
unsigned long int / 4	0 to 4,294,967,295
unsigned long long int / 8	2(power)64 −1

HIERARCHY OF OPERATIONS

Determine the hierarchy of operations and evaluate the following expression:

i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8

Stepwise evaluation of this expression is shown below:

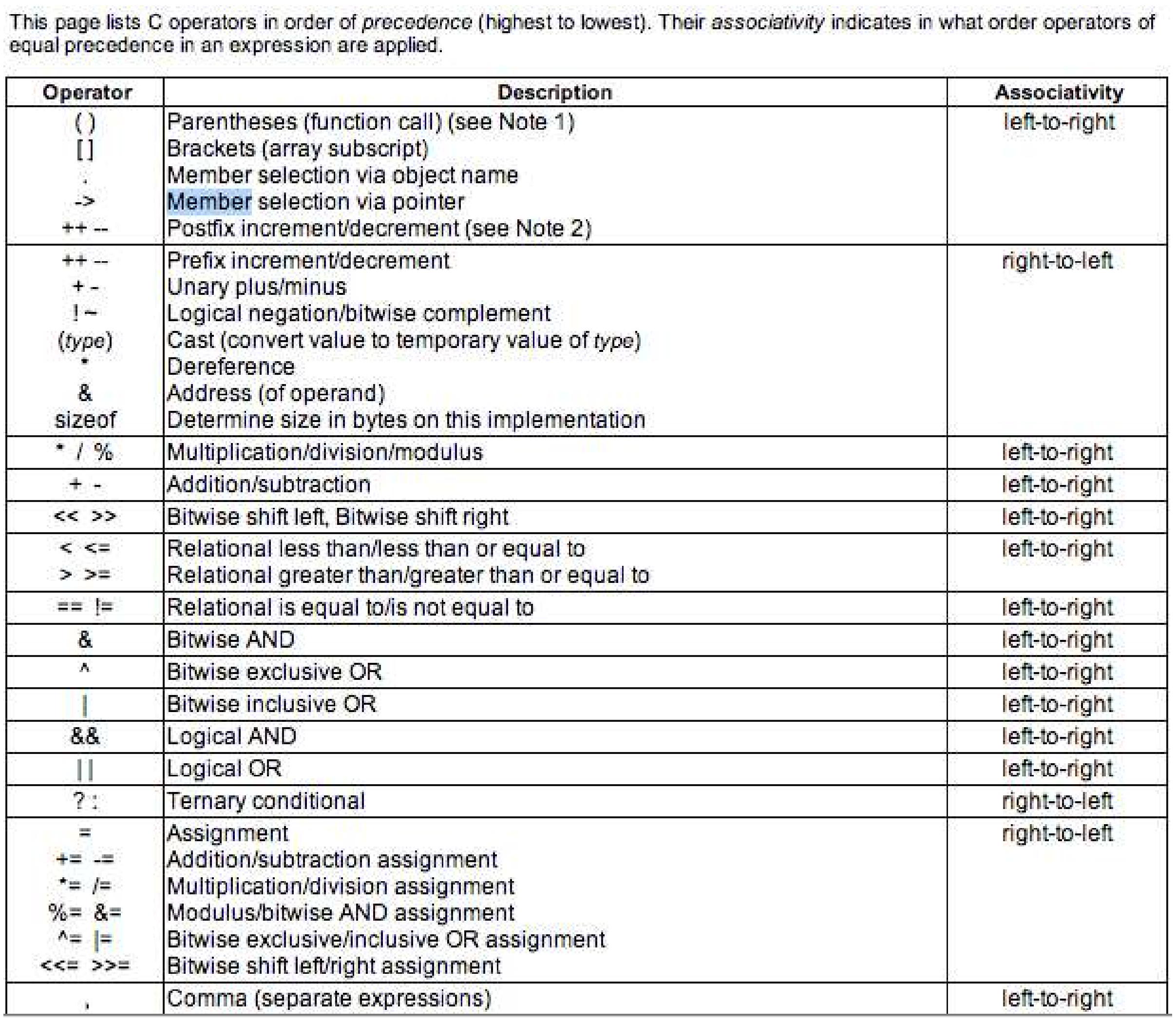
i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8	
i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8	operation: *
i = 1 + 4 / 4 + 8 - 2 + 5 / 8	operation: /
i = 1 + 1+ 8 - 2 + 5 / 8	operations: /
i = 1 + 1 + 8 - 2 + 0	operation: /
i = 2 + 8 - 2 + 0	operation: +
i = 10 - 2 + 0	operation: +
i = 8 + 0	operation: -
i = 8	operations: +

Determine the hierarchy of operations and evaluate the following expression:

kk = 3 / 2 * 4 + 3 / 8 + 3

Stepwise evaluation of this expression is shown below:

kk = 3 / 2 * 4 + 3 / 8 + 3	
kk = 1 * 4 + 3 / 8 + 3	operation: /
kk = 4 + 3 / 8 + 3	operation: *
kk = 4 + 0 + 3	operation: /
kk = 4 + 3	operation: +
kk = 7	operation: +



This page lists C operators in order of *precedence* (highest to lowest). Their *associativity* indicates in what order operators of equal precedence in an expression are applied.

Operator	Description	Associativity
( ) [ ] . -> ++ --	Parentheses (function call) (see Note 1) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement (see Note 2)	left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast (convert value to temporary value of *type*) Dereference Address (of operand) Determine size in bytes on this implementation	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is 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
? :	Ternary conditional	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left
,	Comma (separate expressions)	left-to-right

Exercise

```
#include <stdio.h>
int main( )
{
int i = 2, j = 3, k, l ;
float a, b ;
k = i / j * j ;
l = j / i * i ;
a = i / j * j ;
b = j / i * i ;
printf( "%d %d %f %f", k, l, a, b ) ;
return 0;
}
```

```
0 2 0.000000 2.000000
Process returned 0 (0x0)   execution time : 0.010 s
Press any key to continue.
```

```
#include <stdio.h>
int main()
{
int a, b ;
a = -3 - - 3 ;
b = -3 - - ( - 3 ) ;
printf ( "a = %d b = %d", a, b ) ;
return 0;
}
```

```
a = 0 b = -6
Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.
```

```
#include <stdio.h>
int main()
{
float a = 5, b = 2 ;
int c ;
c = a % b ;
printf ( "%d", c ) ;
return 0;
}
```

```
error: invalid operands to binary % (have 'float' and 'float')
```

```
#include <stdio.h>
int main(){
int a, b ;
printf ( "Enter values of a and b" ) ;
scanf ( " %d %d ", &a, &b ) ;
printf ( "a = %d b = %d", a, b ) ;
return 0;
}
Enter values of a and b
10 20
30
a = 10 b = 20
Process returned 0 (0x0)   execution time : 7.816 s
Press any key to continue.
```

```
#include <stdio.h>
int main()
{
printf ( "nn \n\n nn\n" ) ;
printf ( "nn /n/n nn/n" ) ;
return 0;
}
nn

nn
nn /n/n nn/n
Process returned 0 (0x0)   execution time : 0.010 s
Press any key to continue.
```

Evaluate the following expressions and show their hierarchy

(a) g = big / 2 + big * 4 / big - big + abc / 3 ;
(abc = 1.5, big = 3, assume g to be a float)

Ans: g=2.5

(b) on = ink * act / 2 + 3 / 2 * act + 2 + tig ;
(ink = 3, act = 2, tig = 3.2, assume on to be an int)