C Programming

Introduction

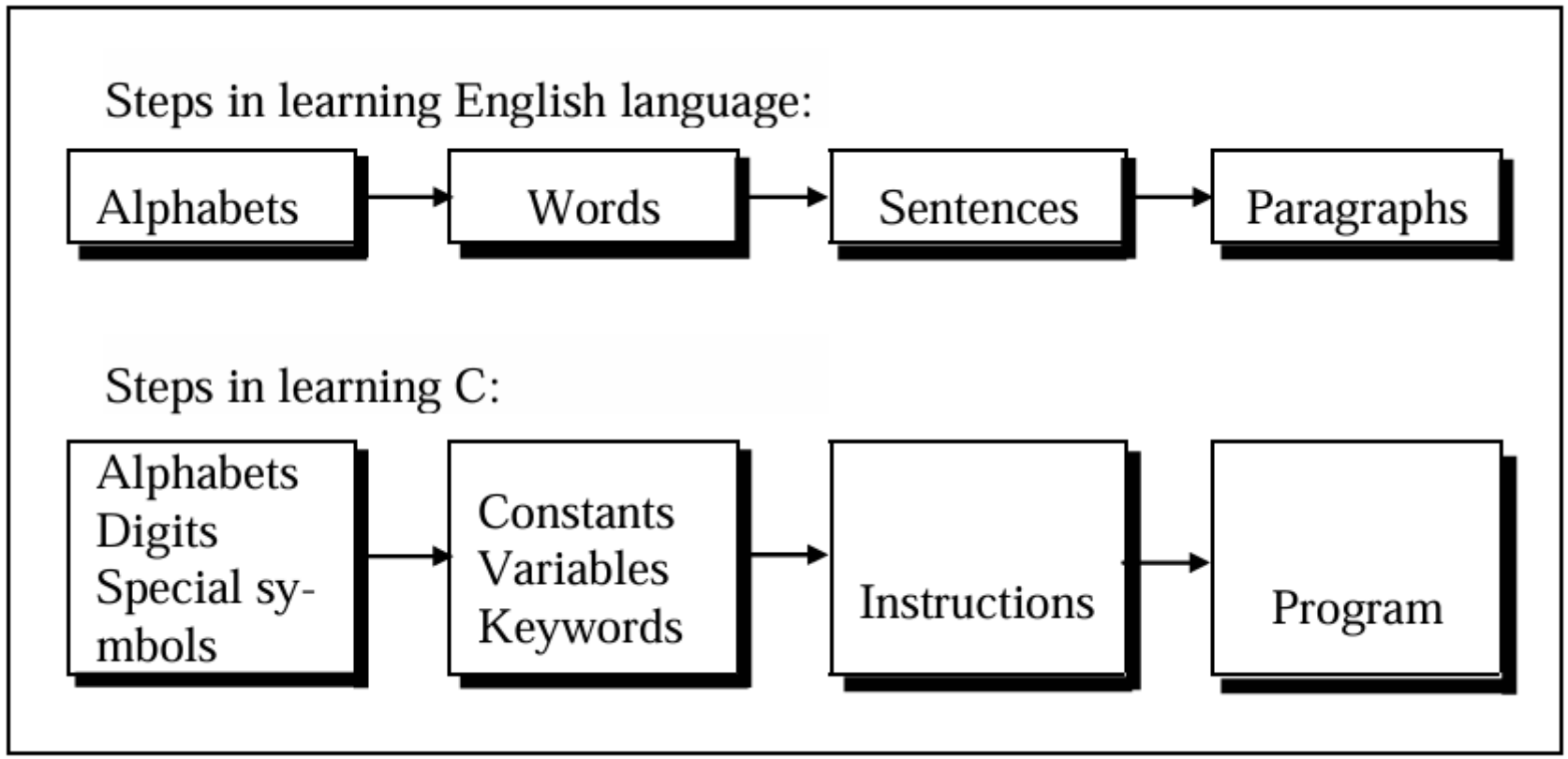
- C is a programming language developed at AT & T Bell labs of USA in 1972
- Designed and written by Dennis Ritchie
- In late 70’s it began to replace more familiar languages
- A very popular programming language, C is still used widely
- Major part of operating systems like Windows, Linux, Unix is still written in C
- Device driver programs, used for extending the operating system to work for new devices are still written in C

Introduction

- Applications where there are constraints on space and time, C is the language of choice. E.g. mobile devices like cell phone, microwave oven, washing machine, digital cameras that include a microprocessor, an operating system and a program embedded in the device
- Many popular gaming frameworks have been built using C
- Applications where close interaction with hardware device is required, C is the preferred choice as it provides several language elements that make this interaction feasible without compromising the performance.

Learning C

- Similar to learning language
- Alphabets then words then sentences then paragraphs
- Alphabets, numbers and special symbols
- Constants, variables, keywords then finally writing instructions
- Group of instructions are later combined to form programs


Steps in learning English language:
Alphabets
Words
Sentences
Paragraphs
Steps in learning C:
Alphabets
Digits
Special sy-
mbols
Constants
Variables
Keywords
Instructions
Program

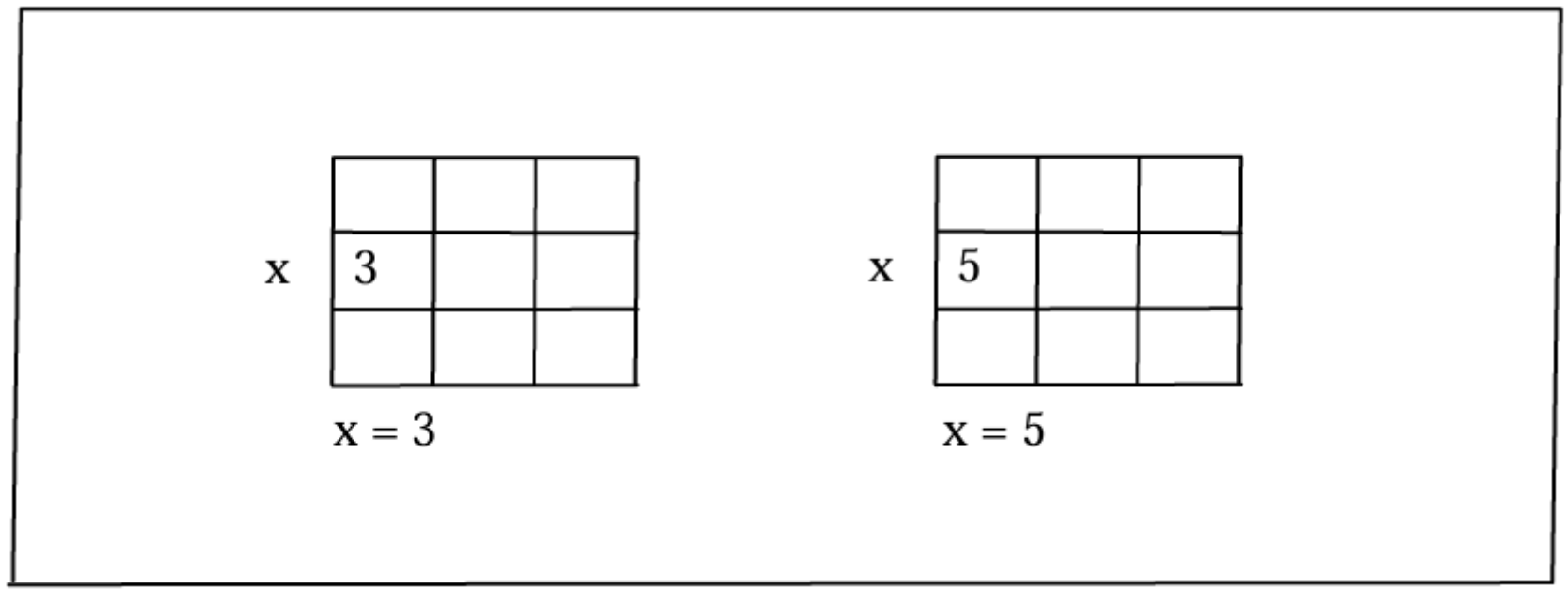
The C character set

- A character denotes any alphabet, digit or special symbol used to represent information
- valid alphabets, numbers and special symbols allowed in C are given below:

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

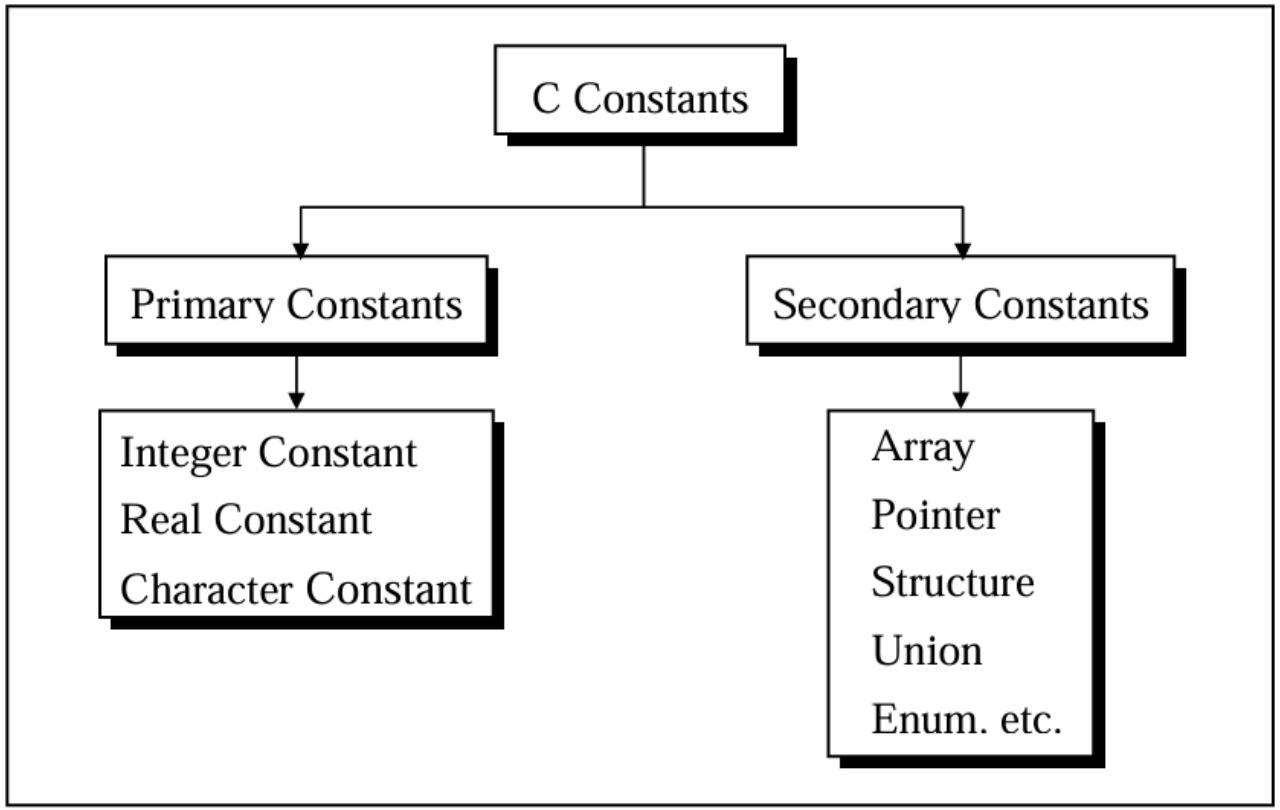
Constants, Variables and Keywords

- The alphabets, numbers and special symbols when properly combined form constants, variables and keywords
- A constant is an entity that doesn’t change whereas a variable is an entity that may change.
- In any program we typically do lots of calculations. The results of these calculations are stored in computers memory.
- Like human memory the computer memory also consists of millions of cells.
- The calculated values are stored in these memory cells.
- To make the retrieval and usage of these values easy these memory cells (also called memory locations) are given names.
- Since the value stored in each location may change, the names given to these locations are called variable names.


x
3
x = 3
x
5
x = 5

Types of C Constants

- C constants can be divided into two major categories:
  - Primary Constants
  - Secondary Constants



Rules for Constructing Integer Constants

- An integer constant must have at least one digit.
- It must not have a decimal point.
- It can be either positive or negative.
- If no sign precedes an integer constant it is assumed to be positive.
- No commas or blanks are allowed within an integer constant.
- The allowable range for integer constants is -32768 to 32767
- Truly speaking the range of an Integer constant depends upon the compiler. For a 16-bit compiler like Turbo C or Turbo C++ the range is –32768 to 32767. For a 32-bit compiler the range would be even greater
- E.g. 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
- A real constant must have at least one digit.
- It must have a decimal point.
- It could be either positive or negative.
- Default sign is positive.
- No commas or blanks are allowed within a real constant.

E.g.
+325.34
426.0
-32.76
-48.5792

Exponential Form

- The exponential form of representation of real constants is usually used if the value of the constant is either too small or too large.
- It however doesn’t restrict us in any way from using exponential form of representation for other real constants.
- In exponential form of representation, the real constant is represented in two parts. The part appearing before ‘e’ is called mantissa, whereas the part following ‘e’ is called exponent.

Exponential Form

- The mantissa part and the exponential part should be separated by a letter e.
- The mantissa part may have a positive or negative sign.
- Default sign of mantissa part is positive.
- The exponent must have at least one digit, which must be a positive or negative integer. Default sign is positive.
- 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

- 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.
- The maximum length of a character constant can be 1 character.
- Ex.: 'A’
- 'I’
- '5’
- '='

Types of C Variables

- As we saw earlier, an entity that may vary during program execution is called a variable.
- Variable names are names given to locations in memory.
- These locations can contain integer, real or character constants.
- In any language, the types of variables that it can support depend on the types of constants that it can handle.
- This is because a particular type of variable can hold only the same type of constant. For example, an integer variable can hold only an integer constant

Rules for Constructing Variable Names

- A variable name is any combination of 1 to 31 alphabets, digits or underscores. Some compilers allow variable names whose length could be up to 247 characters. Still, it would be safer to stick to the rule of 31 characters. Do not create unnecessarily long variable names
- The first character in the variable name must be an alphabet or underscore.
- No commas or blanks are allowed within a variable name.
- No special symbol other than an underscore (as in gross_sal) can be used in a variable name.

Rules for Constructing Variable Names

- Ex.: si_int
- m_hra
- pop_e_89
- It is compulsory for you to declare the type of any variable name that you wish to use in a program.
- This type declaration is done at the beginning of the program
- Ex.: int si, m_hra ; float bassal ; char code ;

C Keywords

- Keywords are the words whose meaning has already been explained to the C compiler
- The keywords cannot be used as variable names
- Some C compilers allow you to construct variable names that exactly resemble the keywords. However, it would be safer not to mix up the variable names and the keywords.
- The keywords are also called ‘Reserved words’.

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

- Before we begin with our first C program do remember the following rules that are applicable to all C programs
- Each instruction in a C program is written as a separate statement. Therefore a complete C program would comprise of a series of statements.
- The statements in a program must appear in the same order in which we wish them to be executed; unless of course the logic of the problem demands a deliberate ‘jump’ or transfer of control to a statement, which is out of sequence

- Blank spaces may be inserted between two words to improve the readability of the statement. However, no blank spaces are allowed within a variable, constant or keyword.
- All statements are entered in small case letters.
- C has no specific rules for the position at which a statement is to be written. That’s why it is often called a free-form language.
- Every C statement must end with a ;. Thus ; acts as a statement terminator.

- Let us now write down our first C program. It would simply calculate simple interest for a set of values representing principle, number of years and rate of interest.

```
/* Calculation of simple interest */
/* Author gekay  Date: 25/05/2004 */
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 ) ;

}
```

About Comments

- Comment about the program should be enclosed within /* */.
- Though comments are not necessary, it is a good practice to begin a program with a comment
- Any number of comments can be written at any place in the program.
- The normal language rules do not apply to text written within /* .. */. Thus we can type this text in small case, capital or a combination.
- Comments are completely ignored by the compiler
- Comments cannot be nested. For example,
- /* Cal of SI /* Author sam date 01/01/2002 */ */ is invalid.

main()

- A comment can be split over more than one line, as in,
- /* This is

  a jazzy

  comment */
- main( ) is a collective name given to a set of statements. This name has to be main( ), it cannot be anything else. All statements that belong to main( ) are enclosed within a pair of braces { }
- Any variable used in the program must be declared before using it. For example,
- int p, n ;
- float r, si ;
- Any C statement always ends with a ;

Print function

- Once the value of si is calculated it needs to be displayed on the screen.
- Unlike other languages, C does not contain any instruction to display output on the screen.
- All output to screen is achieved using readymade library functions.
- One suchfunction is printf( ). We have used it display on the screen the value contained in si.
- 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
- In addition to format specifiers like %f, %d and %c the format string may also contain any other characters. These characters are printed as they are when the printf( ) is executed.
- 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) ;
- The output of the last statement would look like this...
- Prin = 1000
- Rate = 8.5

- printf( ) can not only print values of variables, it can also print the result of an expression.
- printf ("%d %d %d %d", 3, 3 + 2, c, a + b * c – d) ;

Receiving Input

- scanf() function can be used

```
/* Calculation of simple interest */
/* Author gekay Date 25/05/2004 */
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 ) ;
}
```

scanf()

- & is an ‘Address of’ operator.
- It gives the location number used by the variable in memory.
- When we say &a, we are telling scanf( ) at which memory location should it store the value supplied by the user from the keyboard.
- Note that a blank, a tab or a new line must separate the values supplied to scanf( )

C Instructions

- There are basically three types of instructions in C:
- Type Declaration Instruction - To declare the type of variables used in a C program.
- Arithmetic Instruction - To perform arithmetic operations between con stants and variables.
- Control Instruction - To control the sequence of execution of various state ments in a C program.

Type Declaration Instruction

- This instruction is used to declare the type of variables being used in the program.
- Any variable used in the program must be declared before using it in any statement.
- The type declaration statement is written at the beginning of main( ) function.
- int bas ;
- float rs, grosssal ;
- char name, code ;

- While declaring the type of variable we can also initialize it as shown below
- int i = 10, j = 25 ;
- float a = 1.5, b = 1.99 + 2.4 * 1.44 ;
- The order in which we define the variables is sometimes important sometimes not.
- int j = 25, j = 10 ;
- float a = 1.5, b = a + 3.1 ; => OK
- float b = a + 3.1, a = 1.5 ; => not OK
- int a, b, c, d ;
- a = b = c = 10 ; => OK
- int a = b = c = d = 10 ; => not OK, we are trying to use b (to assign to a) before defining it.

Arithmetic Instruction

- A C arithmetic instruction consists of a variable name on the left hand side of = and variable names & constants on the right hand side of =.
- The variables and constants appearing on the right hand side of = are connected by arithmetic operators like +, -, *, and /
- int ad ;
- float kot, deta, alpha, beta, gamma ;
- ad = 3200 ;
- kot = 0.0056 ;
- deta = alpha * beta / gamma + 3.2 * 2 / 5 ;

Operands and Operator

- The variables and constants together are called ‘operands’
- that are operated upon by the ‘arithmetic operators’
- and the result is assigned, using the assignment operator, to the variable on left hand side.

Important Points regarding arithmetic in C

- C allows only one variable on left-hand side of =. That is, z = k *1 is legal, whereas k * 1 = z is illegal
- In addition to the division operator C also provides a modular division operator. This operator returns the remainder on dividing one integer with another. Thus the expression 10 / 2 yields 5, whereas, 10 % 2 yields 0
- modulus operator (%) cannot be applied on a float
- on using % the sign of the remainder is always same as the sign of the numerator
- –5 % 2 yields –1, whereas, 5 % -2 yields 1

- An arithmetic instruction is often used for storing character constants in character variables
- char a, b, d ;
- a = 'F’ ;
- b = 'G’ ;
- d = '+’ ;
- When we do this the ASCII values of the characters are stored in the variables. ASCII values are used to represent any character in memory. The ASCII values of ‘F’ and ‘G’ are 70 and 71

Arithmetic on characters

- Arithmetic operations can be performed on ints, floats and chars.
- char x, y ;
- int z ;
- x = 'a’ ;
- y = 'b’ ;
- z = x + y ;
- The ASCII values of ‘a’ and ‘b’ are 97 and 98, and hence can definitely be added

Operators must be written explicitly

- No operator is assumed to be present. It must be written explicitly.
- a = c.d.b(xy) => usual arithmetic statement
- b = c * d * b * (x * y) => C statement

Exponentiation

- Unlike other high level languages, there is no operator for performing exponentiation operation.
- Thus following statements are invalid
- a = 3 ** 2 ; b = 3 ^ 2 ;
- If we want to do the exponentiation we can get it done this way:

```
#include <math.h>
main( )
{
    int a ;
    a = pow ( 3, 2 ) ;
    printf ( "%d", a ) ;
}
```

Integer and Float Conversions

- An arithmetic operation between an integer and integer always yields an integer result.
- An operation between a real and real always yields a real result.
- 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

- It may so happen that the type of the expression and the type of the variable on the left-hand side of the assignment operator may not be same.
- In such a case the value of the expression is promoted or demoted depending on the type of the variable on left-hand side of =
- For example,
- int i ;
- float b ;
- i = 3.5 ; => 3 gets stored after demotion
- b = 30 ; => 30.000000 gets stored after promotion
- Instead of a simple expression used in the above examples if a complex expression occurs, still the same rules apply.

Type Conversion in Assignments

- float a, b, c ;
- int s ;
- s = a * b * c / 100 + 32 / 4 - 3 * 1.1 ;
- during evaluation of the expression the ints would be promoted to floats
- and the result of the expression would be a float.
- But when this float value is assigned to s it is again demoted to an int and then stored in s

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

Hierarchy of Operations

Priority	Operators	Description
1st	* / %	multiplication, division, modular division
2nd	+ -	addition, subtraction
3rd	=	assignment

Hierarchy of Operations

- if there are more than one set of parentheses, the operations within the innermost parentheses would be performed first, followed by the operations within the second innermost pair and so on
- We must always remember to use pairs of parentheses.

Example 1

- Determine the hierarchy of operations and evaluate the following expression:
- i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8
- i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8
- i = 1 + 4 / 4 + 8 - 2 + 5 / 8
- i = 1 + 1+ 8 - 2 + 5 / 8
- i = 1 + 1 + 8 - 2 + 0
- i = 2 + 8 - 2 + 0
- i = 10 - 2 + 0
- i = 8 + 0
- i = 8

Example 2

- Determine the hierarchy of operations and evaluate the following expression:
- kk = 3 / 2 * 4 + 3 / 8 + 3
- kk = 3 / 2 * 4 + 3 / 8 + 3
- kk = 1 * 4 + 3 / 8 + 3
- kk = 4 + 3 / 8 + 3
- kk = 4 + 0 + 3
- kk = 4 + 3
- kk = 7

Conversion of arithmetic statement to C statement

Algebric Expression	C Expression
$a \times b - c \times d$	a * b – c * d
$(m + n)\ (a + b)$	(m + n) * (a + b)
$3x2 + 2x + 5$	3 * x * x + 2 * x + 5
$\frac{a+b+c}{d+e}$	(a + b + c) / (d + e)
$\left[\frac{2BY}{d+1} - \frac{x}{3(z+y)}\right]$	2 * b * y / (d + 1) – x / 3 * (z + y)

Associativity of Operators

- When an expression contains two operators of equal priority the tie between them is settled using the associativity of the operators
- Associativity can be of two types—Left to Right or Right to Left
- Left to Right associativity means that the left operand must be unambiguous
- Unambiguous means that it must not be involved in evaluation of any other sub-expression
- in case of Right to Left associativity the right operand must be unambiguous

Example

- Consider the expression
- a = 3 / 2 * 5 ;
- Since both / and * have L to R associativity and only / has unambiguous left operand (necessary condition for L to R associativity) it is performed earlier.

Operator	Left	Right	Remark
/	3	2 or 2 * 5	Left operand is unambiguous, Right is not
*	3 / 2 or 2	5	Right operand is unambiguous, Left is not

Example

- a = b = 3 ;
- Here both assignment operators have the same priority and same associativity (Right to Left)
- Since both = have R to L associativity and only the second = has unambiguous right operand (necessary condition for R to L associativity) the second = is performed earlier.

Operator	Left	Right	Remark
=	a	b or b = 3	Left operand is unambiguous, Right is not
=	b or a = b	3	Right operand is unambiguous, Left is not

Example

- z = a * b + c / d ;
- Here * and / enjoys same priority and same associativity (Left to Right)
- Here since left operands for both operators are unambiguous Compiler is free to perform * or / operation as per its convenience

Operator	Left	Right	Remark
*	a	b	Both operands are unambiguous
/	c	d	Both operands are unambiguous

Control Instructions in C

- As the name suggests the ‘Control Instructions’ enable us to specify the order in which the various instructions in a program are to be executed by the computer.
- In other words the control instructions determine the ‘flow of control’ in a program
- There are four types of control instructions in C.
- Sequence Control Instruction
- Selection or Decision Control Instruction
- Repetition or Loop Control Instruction
- Case Control Instruction

Control Instructions in C

- Sequence control instruction ensures that the instructions are executed in the same order in which they appear in the program.
- Decision and Case control instructions allow the computer to take a decision as to which instruction is to be executed next.
- The Loop control instruction helps computer to execute a group of statements repeatedly.