Advantages of writing algorithm

Effective communication: Algorithm is written using English like language,algorithm is better way of communicating the logic to the concerned people

Effective Analysis: with the help of algorithm , problems can be analyzed effectively , There can be number of algorithms for solving given problem

Proper Documentation: The selected algorithm has to be documented . The algorithm serves as a good documentation which is required to identify the logical errors

Easy and Efficient Coding: The algorithm act as blueprint during program development

Program Debugging: Debugging is nothing but identifying the logical errors in algorithm /program. Algorithms help in debugging so that we can identify the logical errors easily

Program maintenance: Maintaining the software becomes much easier

Advantages of writing algorithm

Effective communication: Flowcharts are better way of communicating the logic to the concerned people i.e. programmers and business people

Effective Analysis: with the help of Flowchart , problems can be analyzed effectively

Proper Documentation: Flowcharts play an important role in understanding the logic of complicated and lengthy problems. Hence it is very much required to document the flowcharts along with the algorithms

Easy and Efficient Coding: By looking the flowchart, writing programs is a straight forward method .

Program Debugging: Flowcharts help in debugging process

Program maintenance: Maintaining the software becomes much easier

Disadvantages/Drawbacks of writing algorithm

Complex Logic: For complicated logic ,the flowcharts becomes complex.

Alteratations and Modifications: If there is a change in the logic , the flowchart has to be completely rewritten and requires lot of time.

Reproduction: In case of any problem, reproduction of flowchart becomes problem since the flowchart symbols cannot be typed

Examples:

1.Write a algorithm & flowchart to Check whether the given number is odd or even

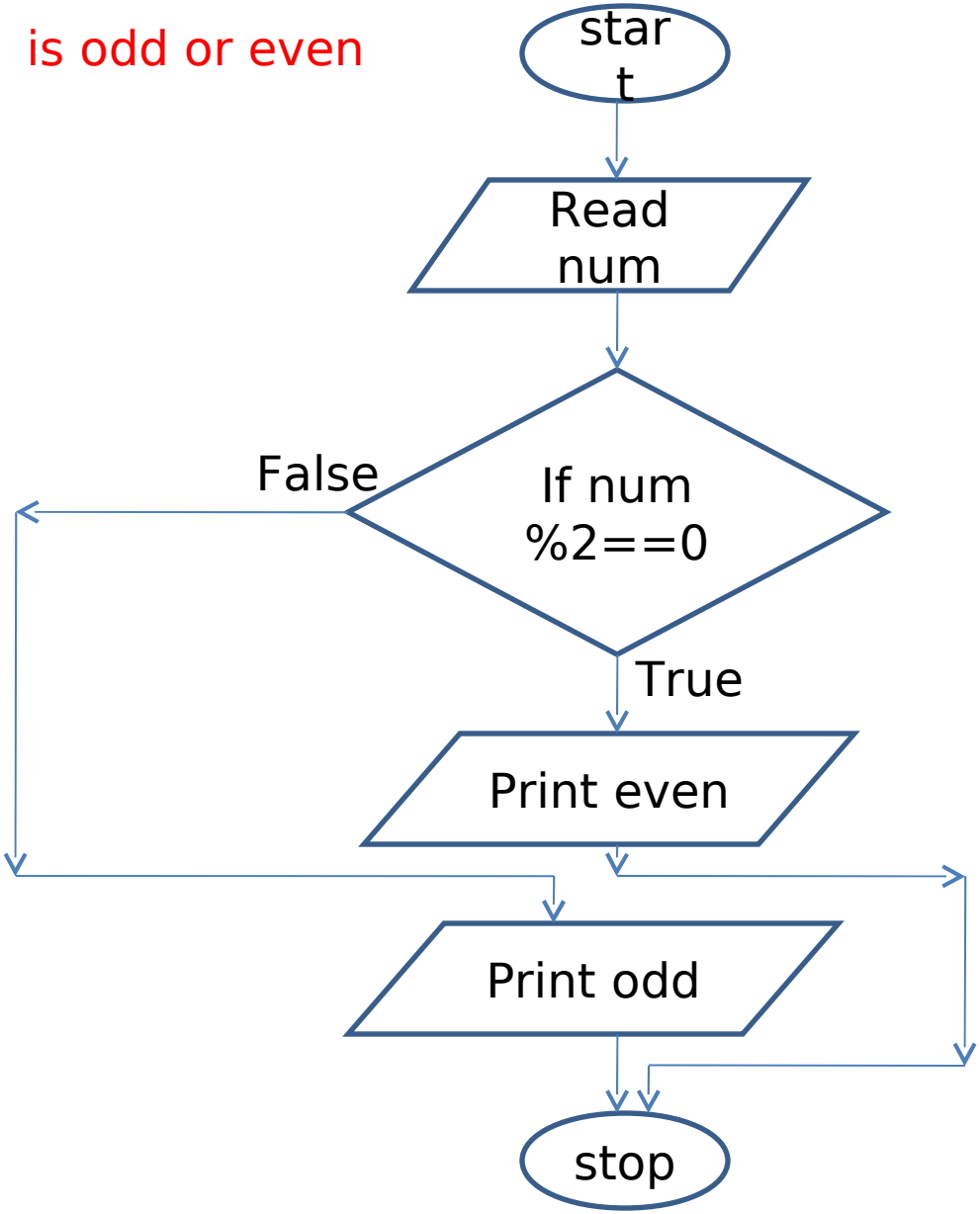
2.Write a algorithm & flowchart to Check Whether the number is positive or negative or zero

3.Write a algorithm & flowchart to find the larger of two numbers

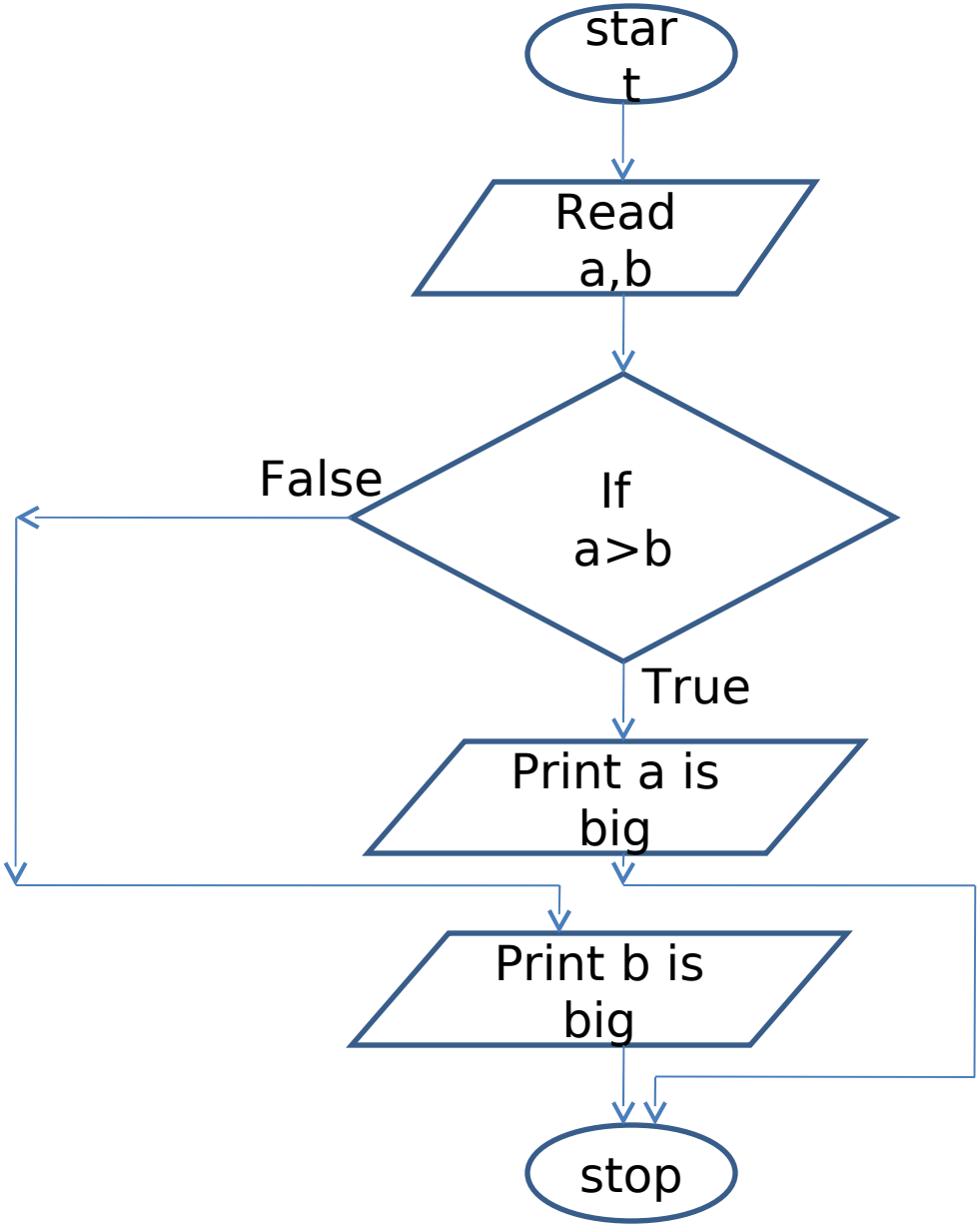
4.Write a algorithm & flowchart to find the larger of 3 numbers

5.Write a algorithm & flowchart to Solve a quadratic equation

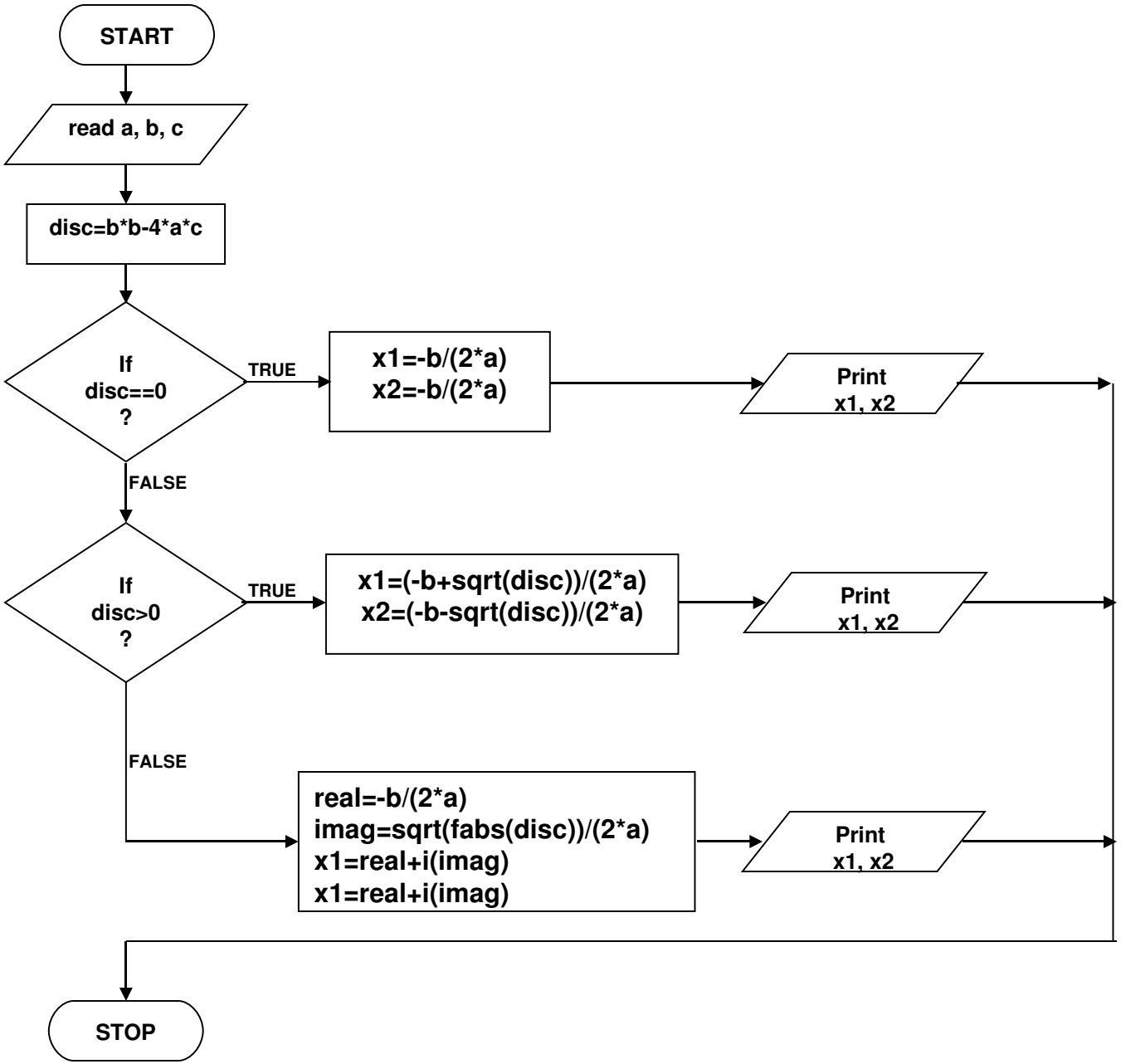
1. Write a algorithm & flowchart to Check whether the given number is odd or even



1. Write a algorithm & flowchart to find the larger of two numbers



5. QUADRATIC EQUATION



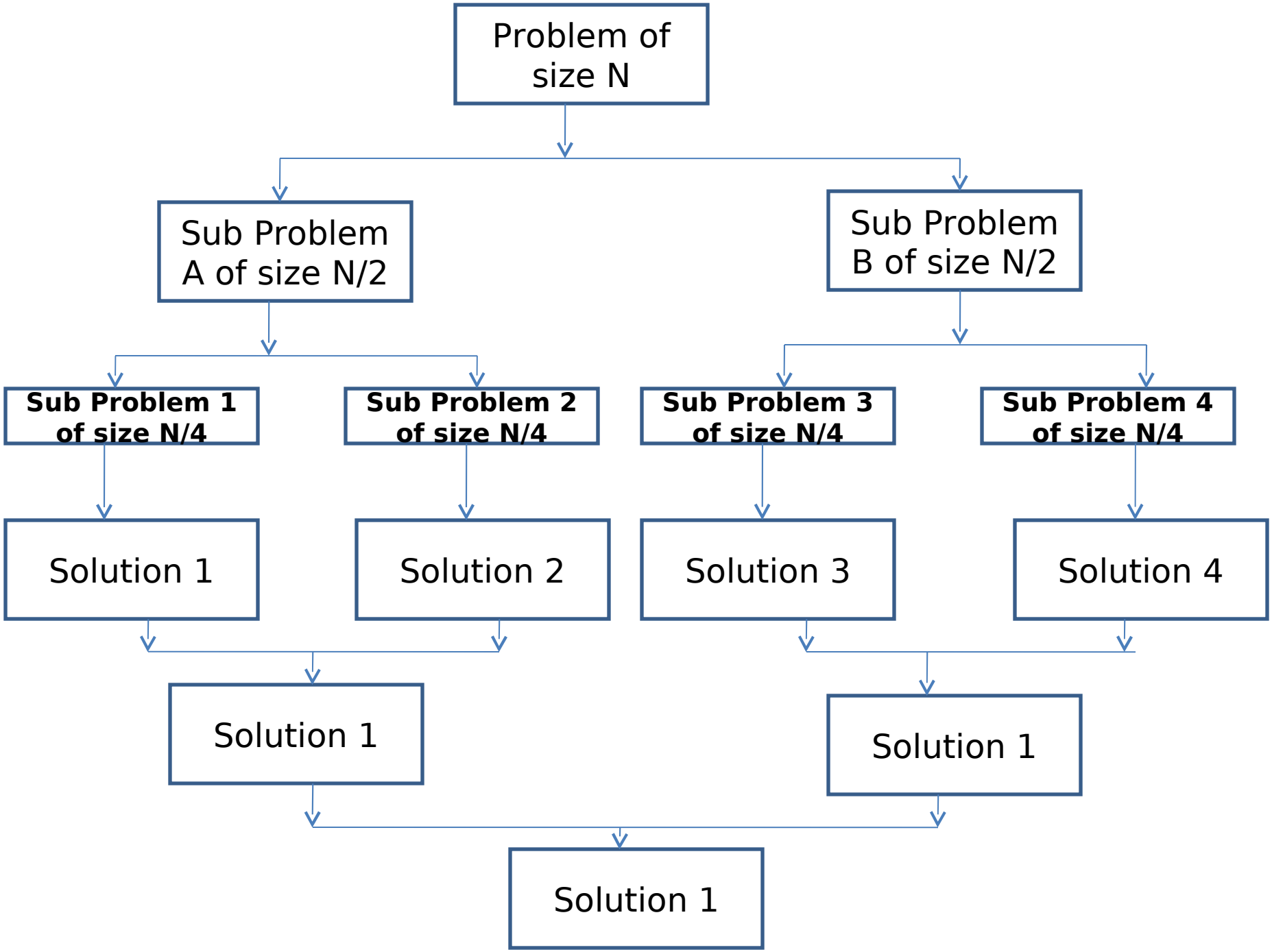
Divide and conquer Strategy

The divide and conquer strategy involves three steps

1. Divide the problem into a number of subproblems

2. Conquer(solve) the subproblem. If the subproblem sizes are small enough,solve them in a straight forward manner. If the subproblems are large,divide them into still number of subproblems

3. Combine the solutions of various subproblems into a single solution


Problem of size N
Sub Problem A of size N/2
Sub Problem B of size N/2
Sub Problem 1 of size N/4
Sub Problem 2 of size N/4
Sub Problem 3 of size N/4
Sub Problem 4 of size N/4
Solution 1
Solution 2
Solution 3
Solution 4
Solution 1
Solution 1
Solution 1

Advantages

1.Difficult and complex problems can be solved easily

2.This Technique simple and very efficient algorithm

3.Since problems are divided into smaller sub problems of same type as the original problem, all the sub problems can be solved in parallel thus increasing the efficiency

DisAdvantages

1. Since many problems can be solved using recursion they are slightly slower in execution speed

2. These problems tend to occupy more memory

3. For simple problems,this technique may become more complicated than the straightforward iterative technique

What are the difference between algorithms and flowcharts

Flowchart	Algorithms
1.It is a graphical representation along with instructions	1.Algorithms are not represented graphically. Instead expressed in English like language along with mathematical expressions
2.For complex problems ,flowchart become complex	2.Algorithms can be used for simple or complex problems
3.Modifications is difficult	3.Modified easily
4.We can easily understand the logic for a given problem	4.Understanding may be slightly difficult

Constants, Variables and Data types

Objective:

To learn about keywords available in C language,

1.keywords, identifiers and constants with example,

2.variables, conventions used to write variables, local and global scope of variables and character set.

Why ‘C’ Language

Computers can understand machine language written in 0’s and 1’s

Computers have to be instructed or programmed using 0’s and 1’s is called machine language

Characters set

1. The set of characters is called character set

2. The Characters that can be used to form words, numbers

3. The characters in C are grouped in to

    1. Letters
    2. Digits
    3. Special characters
    4. White spaces

letter A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m
n o p q r s t u v w x y z

digit 0 1 2 3 4 5 6 7 8 9

Special Symbols

! " # % & ' ( ) * + , - . / :
; < = > ? [\ ] ^ { | } ~

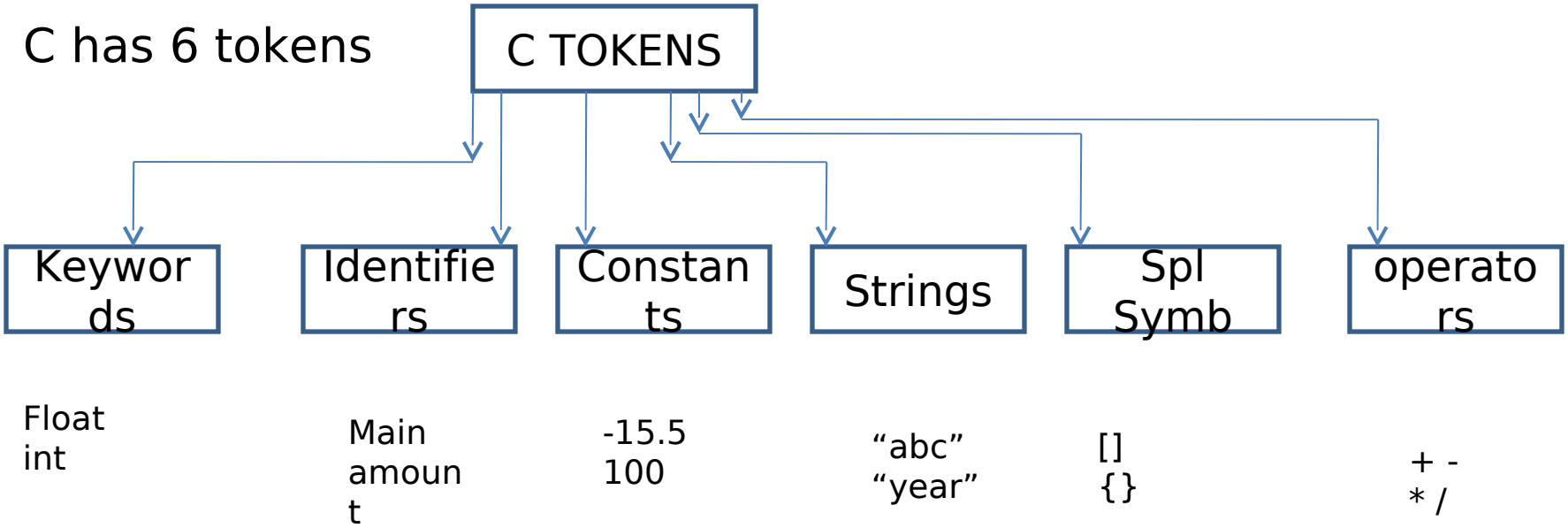
White Spaces

C TOKENS

Individual words are called tokens

In C Program the smallest individual units are known as C Tokens

C has 6 tokens



KEYWORDS

1. All Keywords have fixed meaning and these meanings cannot be changed

2. All Keywords must be written in small letters

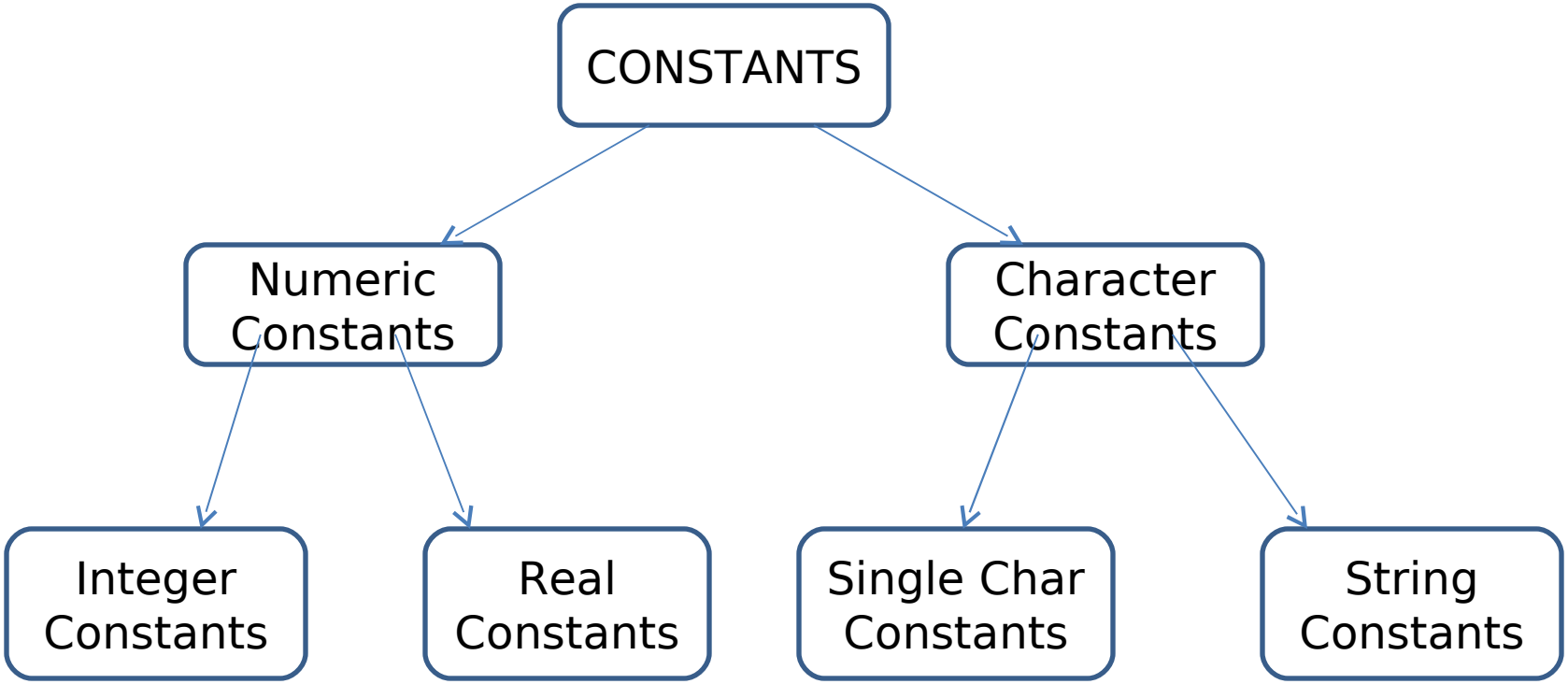
3. Total 32 Keywords

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

CONSTANTS

1. Fixed Values that do not change during the execution of a program

2. Types of Constants



VARIABLES

1. A Variable is a data name that may be used to store a data value

2. A variable may take different values at different times during execution

3. A Variable name can be chosen by the programmer In a meaningful way

4. Each Variable has a **Type**

Rules / Conditions

1.They Must begin with a letter and also permit underscore as the first character

2.It recognizes a length of 31 characters . However ,length should not be normally more than 8 characters

3.Uppercase and lowercase are significant

4.It should not be a keyword

5.White space is not allowed

Some examples of valid variable

John	value	T_raise
Delhi	X1	Ph_Value
_sum		

Some examples of Invalid variable names

123	(area)
%	25th

DATA TYPES

1. Storage representation and machine instructions to handle constant differ from machine to machine

2. ANSI C Supports 3 types classes datatypes
   1. Fundamental data types
   2. Derived data types
   3. User-defined data types

1.Fundamental data types

1.Int
2.Float
3.Double
4.Char
5.Void

Integer Types (int)

1. Integers are whole numbers with range of values supported by particular machine

2. Integers occupy one(1) word of storage and word size of machines vary the size of an integer that can be stored depending on computer

3. If its 16 bit word length
   Size of the integer value is limited to the range **-32768 to 32767**
4. Signed integers uses one bit for sign and 15 bits for magnitude

Floating point types

1. Floating point numbers are stored in 32bits
   with 6 digits of precision
2. Float
1. Double
   1. more accuracy
   2. 64 bit and 14 bits precision

1. Long double
   80 bits

Examples:

```
float money;
money = 0.12;
```

Void Types

1. Void type has no values

2. Used to specify the type of functions

3. The type of function is said to be void when it does not return any value to the calling functions

Char type

1. A single character
2. Characters are usually stored in 8bits

3. Range :
   unsigned characters : 0 to 255
   signed characters : -128 to +127
4. Example

char letter;

letter = ‘A’;

OPERATORS AND EXPRESSIONS

1. An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations

2. Operators are used in programs to manipulate data and variables

3. C operators can be classified into a number of categories
   1.Arithmetic operators
   2.Relational operators
   3.Logical operators
   4.assignment operators
   5.Increment or decrement operators
   6.Conditional operators
   7.Bitwise operators
   8.Special operators

EXPRESSION

1. Expression is a sequence of operands and operators that reduces to single value
   Ex: 7+8

1. Arithmetic operators

+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

1.Integer arithmetic
2.Real arithmetic
3.Mixed mode arithmetic

2.Relational Operators

Operator	**Meaning**
<	is less than
<=	is less than or equal to
>	Is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

3.Logical operators

Operator	**Meaning**
&&	meaning logical AND
||	meaning logical OR
!	Meaning logical NOT

4.Assignment operators

== equal operator

1.Shorthand assignment operator

Syntax:

v op=exp;

Where

V is variable

op is arithmetic operator

Shorthand Assignment operators

Statement with simple Assignment operator	Statement with shorthand operator
1.a= a +1	a+=1
2.a= a -1	a-=1
3.A=a*(n+1)	a*=n+1

Advantages

1.What appears on the left hand side need not be repeated and therefore it becomes easier to write

2.The statement is more easier to read

3.The statement is more efficient

5.Increment or decrement operators

++ Increment operator

-- Decrement operator

The operator ++ adds 1 to the operand and – subtracts 1

Both are unary operators it can take the following form

++m; or m++

--m ; or m--

Rules for ++ and -- operators

1. Increment or decrement operators are unary operators and they require variable as their operands
2. When postfix ++(or --) is used with a variable in an expression ,the expression is evaluated first using the original value of the variable and then variable is incremented or decremented by one
3. When prefix ++(or --) is used in an expression ,the variable is incremented or decremented first then expression is evaluated using new value of the variable
4. The precedence and associatively of ++ and – operators are same as those of unary + and unary -

6.Conditional Operator /Ternary operator

Syntax:

exp1? exp2: exp3

Where

exp1,exp2,exp3 are expressions

Example:

```
a=10;
B=15;
X=(a>b)?a:b;
```

7.Bitwise operators

1. All data items are stored in the computers memory as a sequence of bits(0’s & 1 ‘s)
2. Manipulation of individual bits is carried out in machine language or assembly language

3. C provides six operators

Operator	Symbol Name	Meaning
&	Ampersand	Bitwise AND
|	Pipeline	Bitwise OR
^	Caret	Exclusive OR
~	Tilde	1’s complement
<<	Double less than	Left shifting of bits
>>	Double greater than	Right shifting of bits

SPECIAL OPERATORS

1. Comma operator
2. Sizeof operator
3. Pointer operator
4. Member selection operator(. and ->)

Comma operator

1. It can be used to link the related expressions together

2. Comma linked list of expressions are evaluated from left to right and the value of right most expression is the value of the combined expression

Ex:

```
Value=(x=10,y=5,x+y);

for(n=1,m=10,n<=m ; n++,m++)
```

The Sizeof operator

1. The sizeof operator is a compile time operator

2. When used with an operand ,it returns the number of bytes the operand occupies

3. The operand may be a variable , a constant or a data type

Example:
m=sizeof(sum);
N=sizeof(int)

Its also used to allocate memory space dynamically to variable during execution of a program

Arithmetic Expressions

1. Arithmetic expression is a combination of variables, constants and operators
2. It takes one or more operands and connects them by an arithmetic operators

Mathematical Expression	**C Equivalent**
1.(2x+1)(3y+z)	(2*x+1)(3*y+z)

Evaluation of arithmetic expression

1. Arithmetic expressions are evaluated from left to right
2. Expressions involving high priority operators evaluated first
3. Expressions are evaluated using an assignment statement of the form

Variable=expressions

Precedence of Arithmetic operators

1. High priority * / %
2. Low priority + -

Rules for evaluation of expression

1. First ,parenthesized sub expression from left to right are evaluated
2. If parentheses are nested, the evaluation begins with the innermost sub expression
3. The precedence rule is applied in determining the order of application of operators is evaluating sub expression
4. The associativity rule is applied when two or more operators of the same precedence level appear in a sub expression
5. Arithmetic expressions are evaluated from left to right using the rules of precedence
6. When parentheses are used the expressions within parenthesis assume highest priority

Type conversions in expressions

1. Implicit type conversion
2. Explicit type conversion

Implicit type conversion

1. C permits mixing of constants and variables of different types in an expression

2. C automatically converts any intermediate values to the proper type so that the expression can be evaluated without loosing any significance this is called as implicit type conversion

Rules that are applied while evaluating expressions

1. If one of the operands is long double,the other will be converted to long double and the result will be long double

2. Else,if one of the operands is double ,the other will be converted to double and result will be double

3. Else,if one of the operands is float ,the other will be converted to float and result will be float

4. Else,if one of the operands is unsigned long int ,the other will be converted to unsigned long int and result will be unsigned long int

5. Else,if one of the operands is long int and other is unsigned int,then

a)if unsigned int can be converted to long int,unsigned int operand will be converted as such and result will be long int

b)else both operands will be converted to unsigned long int and result will be unsigned long int

6. Else, if one of the operands is long int ,the other will be converted to long int and the result will be long int

7. Else if one of the operands is unsigned int,the other will be converted to unsigned int,the other will be converted to unsigned int and result will be unsigned int

Explicit conversion

Syntax:

(type-name)expression

Where

Type-name: is one of the any standard data type

Expression: it may be a constant variable or an expression

Examples:

1) X=(int)7.5;

2) A=(int)21.3/(int)4.5;

3) B=(double)sum/n

4) Y=(int)(a+b)

Operator precedence and associativity

1. The precedence is used determine how an expression involving more than one operator is evaluated

2. The operator at the higher level precedence are evaluated first

3. The operator of the same precedence are evaluated either from **right to left** or from **left to right** depending on the level This is called as associativity property of an operator

UNIT-V

Managing input & output operations

Formatted Output

C Provides the printf() function to display the data on the monitor

The printf() is included in **stdio.h** header file

Syntax:

printf(“control string”,arg1,arg2….argn);

Control string consists of 3 types of items

1.Characters that will be printed on the screen as they appear

2.Format specifications that define the output format for display of each item

3.Escape sequence characters such as \n,\t

1. The control string indicates how many arguments follow and what their types are

2. The arguments arg1,arg2,....,argn are the variables whose values are formatted and printed according the the specifications of the control string

3. The arguments should match in number , order and type with the format specifications

Format specification has the following form

% w. p type-specifier

Where

1.W is an integer number that specifies the total number of columns for the output value

2.P is another integer number that specifies the number of digits to the right of the decimal point or number of character to be printed from the string

Examples:

```
printf(“programming is an art\n”);

Printf(“%d”,number);

Printf(“%f%f”,p,q);

Printf(“sum=%d”,sum);

Printf(“\n\n”);
```

1. Output of integer numbers
2. Output of real numbers
3. Printing of a single character
4. Mixed data output
5. Enhancing the readability of output

1. Output of integer numbers

Syntax: The format specification for printing an integer number is

% W d

Where

W = minimum field width for the output.

d = value to be printed is an integer

The number is written right-justified in the given field width

Format	**output**
Printf(“%d”,9876);	9876

printf

READING A CHARACTER

1. Reading a character from the ‘standard input’ unit and writing it to the ‘standard output’ unit

2. Reading a single character can be done by using the function **getchar()**

   Syntax:

   Variable_name=getchar();

   Where:

   •Variable name is a valid c name that has to be declared a char type

   •When this statement is encountered ,the computer waits until a key is pressed and assigns this character as a value to getchar()

• getchar() is used on the right hand side of an

Example:

```
main()
{
char letter;
 letter=getchar();
}
```

Program 2.

/*Shows the use of getchar() in an interactive environment.*/

```
main()
{

char letter;

printf(“would you like to know my name\n”);

printf((“type Y for YES and N for NO\n);

letter=getchar();

If(letter== ‘Y’ || letter==‘y’)

printf(“\n My name is Nagaraj\n”);

else

printf(“\n you are good for nothing\n”);

}
```

The getchar() may be called successively to read the characters contained in a line of text.

```
char  character;
character= ‘ ‘;
while(character != ‘\n’)
{
character=getchar();
}
```

/*Write a c pgm to enter a character and displays a message on the screen telling the user whether the character is an alphabet or digit ,or any spl character.*/

```
#include<stdio.h>
main()
{
char  character;
printf(“press any key\n”);
character=getchar();
If(isalpha(character)>0)
printf(“the character is a letter”);
else
If(isdigit(character)>0)
printf(“the character is a digit”);
else
printf(“the character is not alphanumeric”);
}
```

WRITING A CHARACTER

Writing character one at a time to the terminal

Syntax:
putchar(variable-name);

Example:

```
char letter=N;
putchar(letter);
```

/* write a c program that reads a character from keyboard and then prints it in reverse case */

```
#include<stdio.h>
main()
{
 char alp;
 printf(“enter an alphabet\n”);
 //putchar(‘\n’);
 alp=getchar();
 if(islower(alp))
 putchar(toupper(alp));
 else
 putchar(tolower(alp));
}
```

FORMATTED INPUT

scanf() ……. Scan formatted

Syntax:

scanf(“control string”,arg1,arg2,…….argn);

1.The control string specifies the field format in which the data is to be entered

2.Arguments arg1,arg2,..argn specify the address of locations where the data is stored

3.Control string and arguments Are separated by commas

Control string (format string) contains field specifications
It may include

1.Field specifications, consisting of the conversion character %,a data type character ,and optional number ,specifying field width

2.Blanks, tabs, or newlines

1. Inputting integer numbers
2. Inputting real numbers
3. Inputting character strings
4. Reading mixed data types
5. Detection of errors in input

Un-conditional branch statements

1.Transfer the control from one point to other

2.Control can be transferred from one statement to the specified statement without condition

Syntax:

```
goto  label;
```

Label is an identifier
Labels need not be declared

```
label: statement;
```

Ex:

```
main()
{
double  x, y;
read: scanf(“%f”,&x);
If(x<0) goto read
Y=sqrt(x);
Printf(“%f%f\n”,x,y);
goto read;
}
```

Assignment

1. What is a control statement?
2. Explain various types of control statements in C
3. Write the syntax of nested if with flow chart.
4. Write the general syntax of switch statement.
5. What is goto statement? when it can be used
6. Write a C program to check whether given year is leap year or not.
7. Write a C program to check given number is odd or even using Bitwise operator.
8. Write a C program to check given number is positive or negative using Bitwise operator.
9. Write a C program to design a simple calculator using switch of type character.

Decision making and looping

Program loop consists of 2 segments,

1. body of the loop

2. control statements

1. Statements that enable the programmer to execute a set of statements repeatedly till the required activity is completed is called **looping constructs** or looping control statements

Types of looping

1. for loop
2. while loop
3. do-while loop

for loop

Syntax:

```
 for(initialization; test-condition; increment)
{
        Body of the loop;
}
```

Explanation:

1.Initialization of the control variables is done first, using assignment statement such as i=1 and count=0.The variables i and count are known as loop control variables

2.The value of the control variables is tested using the test condition. i<10 that determines when the loop will exit. if the condition is true , the body of the loop is executed; otherwise the loop is terminated and execution continues with the statements that immediately follows the loop

3. when the body of the loop is executed, the control is transferred back to for statement after evaluating last statement in the loop.now the control variable is

Ex:

```
for(x=0;x<=9;x++)
{
printf(“%d”,x);
}
```

Additional features of for loop:

1. More that one variable can be initialized at a time in the for statement
   for(n=0,p=1;n<17;++n)

2. Increment section may also have more than one part
   for(n=1,m=50;n<=m;n=n+1,m=m-1)
   {
       p=m/n;
       printf("%d%d%d\n",n,m,p);
   }

3. Test condition may have any compound relation and testing need not to be limited only to loop control variable

4.In the for loop one or more sections can be omitted

```
m=5;
for(;m!=100;)
{
 printf(“%d\n”,m);
m=m+5;
}
```

```
#include<stdio.h>
#include<stdlib.h>

int n,i,a[10],key;

void input()
{
        printf("Enter the elements\n");
        for(i=0;i<n;i++)
        scanf("%d",&a[i]);
}
```

```
void linear_search() {
        for(i=0;i<n;i++)
        {
                if(a[i]==key)
                {
                        printf("found at pos=%d\n",i+1);
                        exit(0);
                }
        }
        printf("Element not found \n");
}
```

```
void main()
{

        printf("Enter the size of an array\n");
        scanf("%d",&n);
        input();
        printf("Enter the element to be searched \n");
        scanf("%d",&key);
        linear_search();
        getch();
}
```

Nesting of for loops

One for statement is within another for statement

Example:

```
        for(i=1;i<10;i++)
{
.....................
.....................
For(j=1;j!=5;j++)
{
..............
..............
}
..............
..............
}
```

Jumps in loop

Jumping out of a loop:

1. When a break statement is encountered inside a loop,the loop is immediately exited and the program continues with the statement immediately following the loop

2. When the loops are nested the break would only exit from the loop containing it

3. Break will exit only a single loop

```
While(…)
{
……..
……..
If(condition)
break;
……..
……..
}
……..


for(…..)
{
……..
……..
If(error)
break;
……..
……..
}
……..
```

```
do
{
.............
.............
If(condition)
break;
...........
.........
}
While(.........);
.........
```

```
for(....)
{
.........
        for(.......)
        {
        ...........
        if(condition)
        break;
        .....
        }
         .........
}
```