Algorithms

Contents:

Infix to postfix conversion algorithm

This is an algorithm to convert an infix expression into a postfix expression. It uses a stack; but in this case, the stack is used to hold operators rather than numbers. The purpose of the stack is to reverse the order of the operators in the expression. It also serves as a storage structure, since no operator can be printed until both of its operands have appeared.

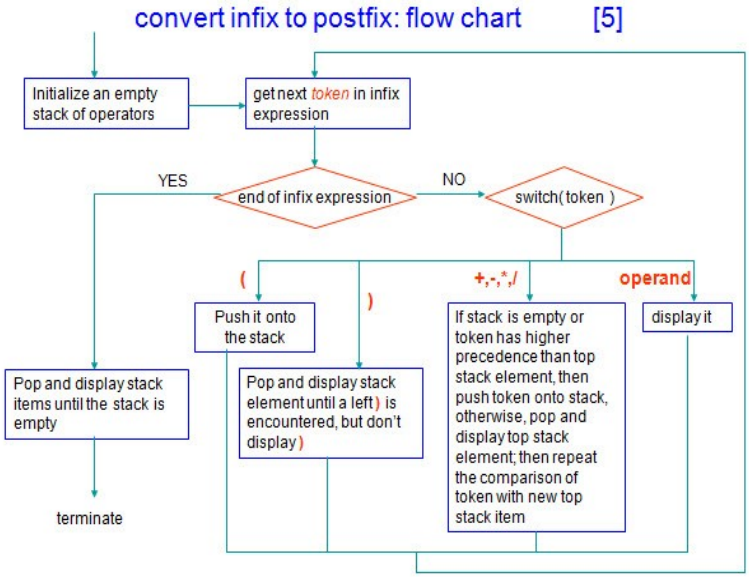
In this algorithm, all operands are printed (or sent to output) when they are read. There are more complicated rules to handle operators and parentheses.

Examples:

1. A * B + C becomes A B * C +

The order in which the operators appear is not reversed. When the '+' is read, it has lower precedence than the '*', so the '*' must be printed first.

We will show this in a table with three columns. The first will show the symbol currently being read. The second will show what is on the stack and the third will show the current contents of the postfix string. The stack will be written from left to right with the 'bottom' of the stack to the left.



Sr. no	Current Symbol	Operator Stack	Postfix string
1.	A		A
2.	*	*	A B
3.	B	*	A B *(pop and print before pushing the '+')
4.	+	+	A B * C
5.	C	+	A B * C +
6.			A B * C +

The rule used in lines 1, 3 and 5 is to print an operand when it is read. The rule for line 2 is to push an operator onto the stack if it is empty. The rule for line 4 is if the operator on the top of the stack has higher precedence than the one being read, pop and print the one on top and then push the new operator on. The rule for line 6 is that when the end of the expression has been reached, pop the operators on the stack one at a time and print them.

Example 2: A * (B + C ^D) + E becomes A B C D ^+ * E +

Sr. no	Current Symbol	Operator Stack	Postfix string
1.	A		A
2.	*	*	A
3.	(	* (	A
4.	B	* (	A B
5.	+	* (+	A B
6.	C	* (+	A B C
7.	^	* (+ ^	A B C
8.	D	* (+ ^	A B C D
9.	)	*	A B C D ^ +
10.	+	+	A B C D ^ + *
11.	E	+	A B C D ^ + * E
12.			A B C D ^ + * E +

A summary of the rules follows:

1. Print operands as they arrive.

2. If the stack is empty or contains a left parenthesis on top, push the incoming operator onto the stack.

3. If the incoming symbol is a left parenthesis, push it on the stack.

4. If the incoming symbol is a right parenthesis, pop the stack and print the operators until you see a left parenthesis. Discard the pair of parentheses.

5. If the incoming symbol has higher precedence than the top of the stack, push it on the stack.

6. If the incoming symbol has equal precedence with the top of the stack, use association. If the association is left to right, pop and print the top of the stack and then push the incoming operator. If the association is right to left, push the incoming operator.

7. If the incoming symbol has lower precedence than the symbol on the top of the stack, pop the stack and print the top operator. Then test the incoming operator against the new top of stack.

8. At the end of the expression, pop and print all operators on the stack. (No parentheses should remain.)

Matrices

Input Two matrices A and B
Output Output matrix C containing elements after addition of a and b

Matrix-Addition and substraction(A,B)
1 for i =1 to rows [A]
2 for j =1 to columns[A]
3 Input A[i,j];
4 Input A[i,j];
5 C[i, j] = A[i, j] + B[i, j]; (for addition)
6 C[i, j] = A[i, j] + B[i, j]; (for substration)
7 Display C[i,j];

Algorithm Description

To add two matrixes sufficient and necessary condition is "dimensions of matrix A = dimensions of matrix B".

Loop for number of rows in matrix A.
Loop for number of columns in matrix A.
Input A[i,j] and Input B[i,j] then add A[i,j] and B[i,j]
store and display this value as C[i,j];

Matrix-Multiply(A, B)
1 if columns [A] ≠ rows [B]
2 then error "incompatible dimensions"
3 else
4 for i =1 to rows [A]
5 for j = 1 to columns [B]
6 C[i, j] =0
7 for k = 1 to columns [A]
8 C[i, j]=C[i, j]+A[i, k]*B[k, j]
9 return C

Algorithm Description

To multiply two matrixes sufficient and necessary condition is "number of columns in matrix A = number of rows in matrix B".
Loop for each row in matrix A.
Loop for each columns in matrix B and initialize output matrix C to 0. This loop will run for each rows of matrix A.
Loop for each columns in matrix A.
Multiply A[i,k] to B[k,j] and add this value to C[i,j]
Return output matrix C.

The determinant of an n by n matrix -by recursion

Termination condition-minor of order 2

$$\det(A) = |A| = \begin{vmatrix} a11 & a12 & a13 & \dots & a1n \\ a21 & a22 & a23 & \dots & \\ a31 & a32 & a33 & \dots & a3n \\ \vdots & \vdots & \vdots & \vdots & \dots \quad \dots \\ \vdots & \vdots & \vdots & aij & \vdots \\ an1 & an2 & an3 & \dots & ann \end{vmatrix}$$

The solution is given by the so called "determinant expansion by minors". A minor Mij of the matrix A is the n-1 by n-1 matrix made by the rows and columns of A except the i'th row and the j'th column is not included. So for example M12 for the matrix A above is given below

$$M12 = \begin{vmatrix} a21 & a23 & a24 & \dots & a2n \\ a31 & a33 & a34 & \dots & a3n \\ a41 & a43 & a44 & \dots & a4n \\ \vdots & \vdots & \vdots & \vdots & \dots \quad \dots \\ \vdots & \vdots & \vdots & aij,\ i!=1,\ j!=2 & \vdots \\ an1 & an3 & an4 & \dots & ann \end{vmatrix}$$

The determinant is the given by the following where the sum is taken over a single row or column.

$$|A| = \text{pow}(-1,i+j)*aij*Mij$$

Any row or column can be chosen across which to take the sum, when computing manually the row or column with the most zeros is chosen to minimise the amount of work. If the first row is chosen for the sum then the determinant in terms of the minors would be

$$|A| = a11\ M11 - a12\ M12 + a13\ M13 - \dots + a1n\ M1n$$

Or expanded out as follows.

```
                |A| = (a11)
                      a22  a23   a24   . . .   a2n
                      a32  a33   a34   . . .   a3n
                      a42  a43   a44   . . .   a4n
                      an2  an3   an4   . . .   ann
        - (a12)
                      a21  a23   a24   . . .   a2n
                      a31  a33   a34   . . .   a3n
                      an1  an3   an4   . . .   ann
        + a13
                      a21  a22   a24   . . .   a2n
                      a31  a32   a34   . . .   a3n
                      a41  a42   a44   . . .   a4n
                      :    :     :     . . .   :
                       an1 an2   an4   . . .   ann

. . . . . +( a1n)
                      a21  a22   a23   . . .   a2(n-1)
                      a31  a32   a33   . . .   a3(n-1)
                      a41  a42   a43   . . .   a4(n-1)
                      :    :     :     . . .   :
                      an1  an2   an3   . . .   an(n-1)
```

The process is repreated for each of the determinants above, on each expansion the dimension of the determinant in each term decreases by 1. When the terms are 2 by 2 matrices the determinant is given as

```
a11    a12      =   (a11 a22    -      a12 a21)
a21    a22
```

this is the termination condition.

If the determinant is 0 the matrix said to be "singular". A singular matrix either has izero elements in an entire row or column, or else a row (or column) is a linear combination of other rows (or columns).

Matrix inversion

$$\mathbf{AB} = \mathbf{BA} = \mathbf{I}_n$$

If matrix is a non singular square matrix ,one can calculate the it's inverse
The algorithm is given below
First make an augmented matrix of size n by 2n by an unit matrix such as the left one is given matrix and the rest right part is unit matrix of size n by n
Then convert the right half of augmented matrix into row echelon form
When the given part of augmented matrix converts into row echelon form the right half gives the inverse of given matrix

Algorithm for Integration

By numerical methods - simpson's 1/3 method

In our source code,we have defined functions *f(x)* . The calculation using **C++ program for Simpson 1/3 rule** is based on the fact that the small portion between any two points is a parabola. The program follows the following steps for calculation of the integral.

As the program gets executed, first of all it asks for the value of lower boundary value of x i.e. x_0, upper boundary value of x i.e. x_n and width of the strip, h.

Then the program finds the value of number of strip as $n=(x_n - x_0)/h$ and checks whether it is even or odd. If theIn the source code, a function f(x) = 1/(1+x) has been defined. The calculation using C program for Simpson 1/3 rule is based on the fact that the small portion between any two points is a parabola. The program follows the following steps for calculation of the integral.

As the program gets executed, first of all it asks for the value of lower boundary value of x i.e. x0, upper boundary value of x i.e. xn and width of the strip, h.

Then the program finds the value of number of strip as n=(xn - x0)/h and checks whether it is even or odd. If the value of 'n' is odd, the program refines the value of 'h' so that the value of 'n' comes to be even.

After that, this C program for Simpson 1/3 rule, calculates value of f(x) i.e 'y' at different intermediate values of 'x' and displays values of all intermediate values of 'y'.

After the calculation of values of 'c', the program uses the following formula to calculate the value of integral in loop.

Integral = *((y0 + yn) +4(y1 + y3 + …….+ yn-1) + 2(y2 + y4 +…….+ yn-2))

Finally, it prints the values of integral which is 'stored as 'ans' in the program.

If f(x) represents the length, the value of integral will be area and if f(x) is area the output of C program for Simpson 1/3 rule will be volume. On balance, numerical integration can be carried out using the program below is very easy to use, simple to understand and gives reliable and accurate results.

value of 'n' is odd, the program refines the value of 'h' so that the value of 'n' comes to be even.
After that, this **C program for Simpson 1/3 rule,** calculates value of f(x) i.e 'y' at different intermediate values of 'x' and displays values of all intermediate values of 'y'.

After the calculation of values of 'c', the program uses the following formula to calculate the value of integral in loop.

Integral = *(($y_0 + y_n$) +4($y_1 + y_3 + \ldots\ldots\ldots + y_{n-1}$) + 2($y_2 + y_4 + \ldots\ldots\ldots + y_{n-2}$))

Finally, it prints the values of integral which is 'stored as 'ans' in the program.

Numerical differentiation-by newtons interpolation formula

Newtons forword interpolation formula

By Tayler's theorem

$$f(x) = f(x_0 + hu \approx y_0 + \Delta y_0 u + \frac{\Delta^2 y_0}{2!}(u(u-1)) + \cdots + \frac{\Delta^k y_0}{k!}\{u(u-1)\cdots(u-k+$$

$$+ \cdots + \frac{\Delta^n y_0}{n!}\{u(u-1)\ldots(u-n+1)\}.$$

Differentiating above tayler's formula , we get the approximate value of the first derivative at x as

$$\frac{df}{dx} = \frac{1}{h}\frac{df}{du} \approx \frac{1}{h}\left[\Delta y_0 + \frac{\Delta^2 y_0}{2!}(2u-1) + \frac{\Delta^3 y_0}{3!}(3u^2 - 6u + 2) + \cdots\right.$$

$$\left. + \frac{\Delta^n y_0}{n!}\left(nu^{n-1} - \frac{n(n-1)^2}{2}u^{n-2} + \cdots + (-1)^{(n-1)}(n-1)!\right)\right].$$

where, $u = \frac{x - x_0}{h}$.

Thus, an approximation to the value of first derivative at $x = x_0$ *i.e.* $u = 0$ is obtained as :

$$\left.\frac{df}{dx}\right|_{x=x_0} = \frac{1}{h}\left[\Delta y_0 - \frac{\Delta^2 y_0}{2} + \frac{\Delta^3 y_0}{3} - \cdots + (-1)^{(n-1)}\frac{\Delta^n y_0}{n}\right]. \qquad (13.2.3)$$

Algorithm for Differential equation

By runge-kutta 4th order method

One member of the family of Runge–Kutta methods is often referred to as "**RK4**", "**classical Runge–Kutta method**" or simply as "***the* Runge–Kutta method**". This method is used to solve ordinary differential equation.

Let the differential equation is:

$$\dot{y} = f(t, y), \quad y(t_0) = y_0.$$

Here, *y* is an unknown function (scalar or vector) of time *t* which we would like to approximate; we are told that $\dot{y}$, the rate at which *y* changes, is a function of *t* and of *y* itself. At the initial time t_0 the corresponding *y*-value is y_0. The function *f* and the data t_0, y_0 are given.

Now pick a step-size $h>0$ and define

$$y_{n+1} = y_n + \tfrac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$t_{n+1} = t_n + h$$

for $n = 0, 1, 2, 3, \ldots$, using

$$k_1 = f(t_n, y_n),$$
$$k_2 = f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}k_1),$$
$$k_3 = f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}k_2),$$
$$k_4 = f(t_n + h, y_n + hk_3).$$

$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i k_i,$$

where, $k_1 = hf(t_n, y_n),$

$$k_2 = hf(t_n + c_2 h, y_n + a_{21} k_1),$$
$$k_3 = hf(t_n + c_3 h, y_n + a_{31} k_1 + a_{32} k_2),$$
$$k_s = hf(t_n + c_s h, y_n + a_{s1} k_1 + a_{s2} k_2 + \cdots + a_{s,s-1} k_{s-1}).$$

k_1 is the increment based on the slope at the beginning of the interval, using $\dot{y}$, (Euler's

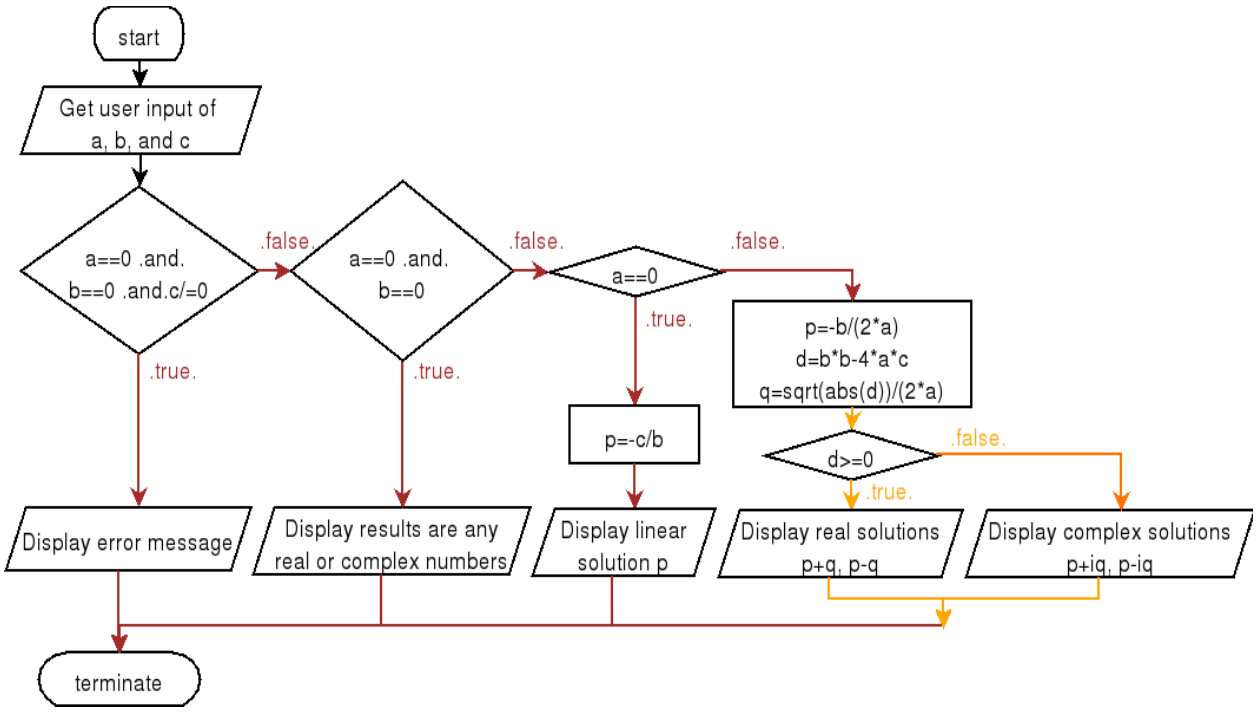
method) ;

k_2 is the increment based on the slope at the midpoint of the interval, using $\dot{y} + \frac{h}{2}k_1$;

k_3 is the increment based on the slope at the midpoint, but now using $\dot{y} + \frac{h}{2}k_2$;

k_4 is increment based on the slope at the end of the interval, using $\dot{y} + hk_3$.

Algorithm for Quadratic Equation:



Algorithm for Linear Equations:

Gaussian elimination is a method for solving matrix equations of the form

$$\begin{matrix} a_{1,1}\cdot x_1+a_{1,2}x_2+\cdots+a_{1,n}\cdot x_n=b_1 \\ a_{2,1}\cdot x_1+a_{2,2}x_2+\cdots+a_{2,n}\cdot x_n=b_2 \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ a_{n,1}\cdot x_1+a_{n,2}x_2+\cdots+a_{n,n}\cdot x_n=b_n \end{matrix}$$

above system of equation can be expressed in matrix form as given below

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}.$$

(1) To perform Gaussian elimination starting with the system of equations

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1\,k} \\ a_{21} & a_{22} & \cdots & a_{2\,k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k\,1} & a_{k\,2} & \cdots & a_{k\,k} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix},$$

(2) Compose the "augmented matrix equation"

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1\,k} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2\,k} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{k\,1} & a_{k\,2} & \cdots & a_{k\,k} & b_k \end{array}\right] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix}.$$

3) Here, the column vector in the variables X is carried along for labeling the matrix

Now,

perform elementary row operations to put the augmented matrix into the upper

triangular form

$$\left[\begin{array}{cccc|c} a'_{11} & a'_{12} & \cdots & a'_{1\,k} & b'_1 \\ 0 & a'_{22} & \cdots & a'_{2\,k} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a'_{k\,k} & b'_k \end{array}\right].$$

Solve the equation of the th row for , then substitute back into the equation of

(k-1)st row obtain a solution for Xk-1, etc., according to the formula

$$x_i = \frac{1}{a'_{i\,i}} \left(b'_i - \sum_{j=i+1}^{k} a'_{i\,j}\, x_j \right).$$