Prefix/Infix/Postfix Notation

(Prefix/Postfix = forma poloneză a unei expresii

Infix = forma algebrică cunoscută de la matematică)

Forma algebrică a unei expresii, în care folosim operațiile de bază (adunare, scădere, înmulțire, împărțire (rest), ridicare la putere) folosește aceste expresii sub forma

operand1 operator operand2 (ex: 5 + 7 => 5 și 7 se numesc "operanzi" iar + se numește "operator").

Forma aceasta este de tip "infix", in referindu-se la poziția operatorului față de operanzi.

Există și operatori UNARI, de forma operator operand (ex: -(3-5) => rezultatul parantezei, care la noi este operand, se va negativa. În algebră operatorii unari sunt + și -.

Ordinea operațiilor este cea știută de la matematică:

- se fac mai întâi parantezele; cea mai interioară paranteză este prima care se face.

- apoi se fac ridicările la putere

- apoi se fac înmulțirile, împărțirile și resturile (în ordinea scrierii)

- apoi se fac adunările și scăderile (în ordinea scrierii).

În exercițiile de la ACSL, vom avea operatorii:

^ sau ↑ (pentru ridicare la putere, gen 4^3)

\*, /, % (pentru înmulțire, împărțire, modulo=rest)

Obs. : / în acest capitol de teorie împarte CU ZECIMALE

+, - (adunare scădere).

Ceea ce înțelegem prin "forma poloneză" a unei expresii algebrice este transformarea acesteia astfel încât SĂ NU Mai utilizeze paranteze.

Mai precis, această formă are o ordine unică de efectuare a calculelor, chiar dacă NU utilizează paranteze.

Transformarea unei expresii algebrice de la forma normală (infix) se face fie ducând operatorul în fața operanzilor, caz în care avem de-a face cu forma PREfix, sau după, caz în care avem de-a face cu forma POSTfix.

Practic, ORICE limbaj de programare, în momentul în care are de calculat o expresie scrisă de programator într-o atribuire, de exemplu, o transformă la forma poloneză prefixată.

Iată câteva exemple de transformare:

|  |  |  |
| --- | --- | --- |
| Infix (algebric normal) | Prefix | Postfix |
| a+b | + a b | a b + |
| a+b\*c | + a \* b c | a b c \* + |
| 5+ | + 5 / 8 - 3 1 | 5 8 3 1 - / + |
| a+b+c | + + a b c | a b + c + |
| a+(b+c) | + a + b c | a b c + + |

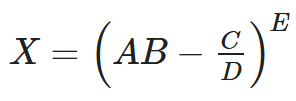
Dacă vă este mai ușor, putem construi un arbore asociat expresiei.

Adică, o structură formată din noduri (cercuri) în care înscriem operatorii și operanzii,

fiecare operator având două noduri fii, unul pe stânga, altul pe dreapta, care reprezintă operanzii.

|  |  |  |  |
| --- | --- | --- | --- |
| Infix (algebric normal) | Prefix | Postfix | Arborele expresiilor |
| a+b | + a b | a b + |  |
| a+b\*c | + a \* b c | a b c \* + |  |
| 5+ | + 5 / 8 - 3 1 | 5 8 3 1 - / + |  |
| a+b+c | + + a b c | a b + c + |  |
| a+(b+c) | + a + b c | a b c + + |  |

Exerciții:

Să convertim la prefix și postfix (inclusiv operatorul = va face parte din expresie, îl considerăm ca fiind operator binar

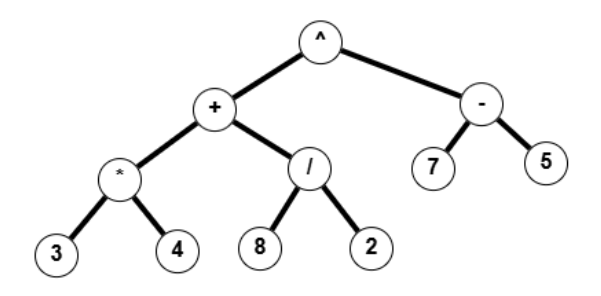
|  |  |  |
| --- | --- | --- |
| arbore: | prefix:  = X ^ - \* A B / C D E | postfix:  X A B \* C D / - E ^ = |

Exerciții:

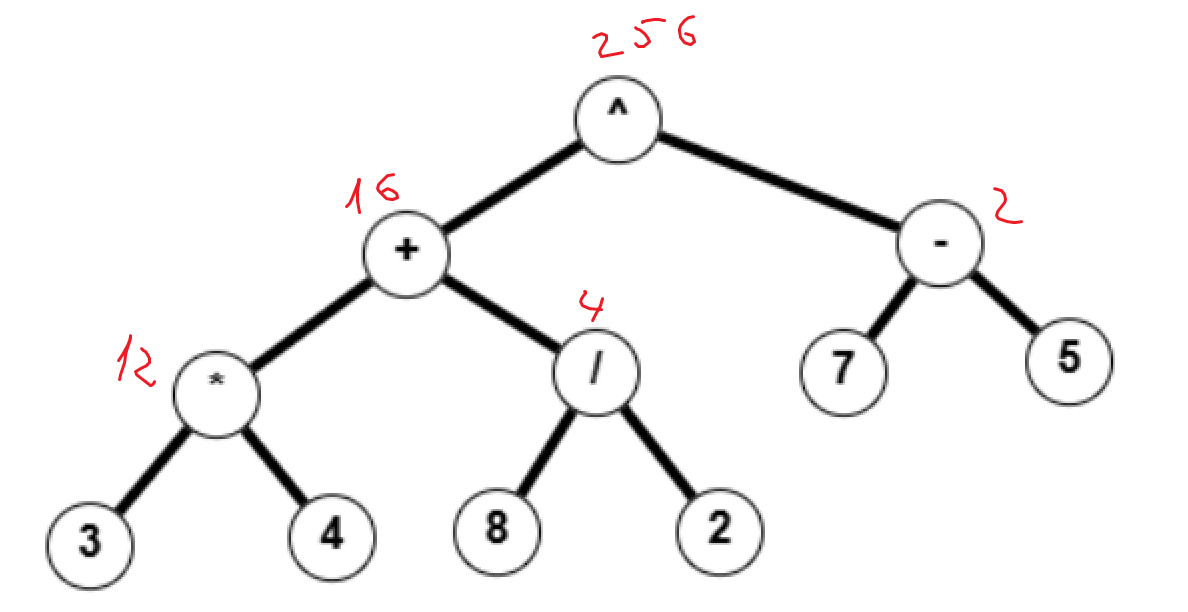
Să trecem expresia următoare de la prefix la infix (forma algebrică)

↑ + \* 3 4 / 8 2 – 7 5

Arborele:



Btw: pe baza arborelui putem calcula imediat valoarea expresiei:



Expresia algebrică (infix) corespunzătoare este:

(3\*4+8/2)^(7-5)

Bit String Flicking

Acest capitol tratează operații pe biți. În general ni se dau valori reprezentate pe un anumit număr de biți (fie ne dăm seama din exercițiu, fie se specifică în enunț) și operațiile care se pot face sunt:

- operații algebrice booleene care se fac bit cu bit (se scriu valorile una sub alta dacă sunt operații binare și se efectuează operațiile) acestea putând fi, în ordinea priorității:

NOT(~), AND (&), XOR (⊕), OR (|).

NOT

|  |  |
| --- | --- |
| a | ~a |
| 0 | 1 |
| 1 | 0 |

AND XOR OR:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | b | a & b | a⊕b | a | b |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |

- operații de shiftare (deplasare) care sunt simple (left-shift, right-shift) în care valorile se deplasează în sensul indicat, cu numărul de poziții indicate. Valorile care ies se pierd, din partea dinspre care se fac deplasările intră zerouri:

LSHIFT-3 abcdefg => defg000

LSHIFT-2 abcdefg => cdefg00

LSHIFT-6 abcdefg => g000000

LSHIFT-7 abcdefg => 0000000

RSHIFT-3 abcdefg => 000abcd

RSHIFT-2 abcdefg => 00abcde

- operații de shiftare (deplasare) care sunt circulare (left-circ și right-circ) în care valorile care ies într-o parte intră prin cealaltă parte

LCIRC-3 abcdefg => defgabc

LCIRC-2 abcdefg => cdefgab

LCIRC-6 abcdefg => gabcdef

LCIRC-7 abcdefg => abcdefg

LCIRC-142 abcdefg <=> LCIRC-2 abcdefg => cdefgab

RCIRC-3 abcdefg => efgabcd

RCIRC-2 abcdefg => fgabcde

De rețiunut că, dacă nr. de poziții este mai mare ca numărul de cifre (biți) deplasarea se face de fapt de un număr de poziții egal cu restul:

LCIRC-poz val <=> LCIRC-(poz%nrcif) val

Exerciții

1) Să se calculeze:

(RSHIFT-1 (LCIRC-4 (RCIRC-2 01101)))

(RSHIFT-1 (LCIRC-4 01011))

(RSHIFT-1 10101) = 01010

2) Exerciții de tip ecuație:

List all possible values of x (5 bits long) that solve the following equation.

(LSHIFT-1 (10110 XOR (RCIRC-3 x) AND 11011)) = 01100

Notăm x=abcde

(LSHIFT-1 (10110 XOR (RCIRC-3 abcde) AND 11011)) = 01100

(LSHIFT-1 (10110 XOR cdeab AND 11011)) = 01100

(LSHIFT-1 (10110 XOR cd0ab)) = 01100

(LSHIFT-1 d1b) = 01100

d1b0 = 01100

identificăm poziție cu poziție.

Literele care au dispărut înseamnă că acele variabile pot avea orice valoare

(în acest gen de ecuații ele se și numesc "don't care values")

Deci avem

d=0

1=1 (dacă am fi avut 1=0 => ecuația NU ar fi avut soluții)

=1 => a=0

b=0

0=0 (dacă am fi avut 0=1 => ecuația NU ar fi avut soluții)

Concluzie:

a=0, b=0, c=\*, d=0, e=\*

deci soluțiile sunt de forma 00\*0\*

adică avem 4 soluții:

00000, 00001, 00100, 00101