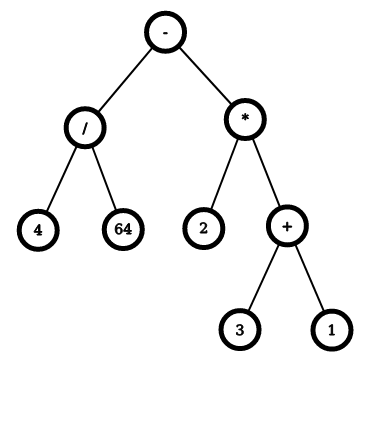
IOCLA

TEMA 1 – Prefix AST

README

1) Ideea de rezolvare a temei



Reprezentarea datelor ca arbore

Reprezentarea datelor ca vectori

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| symbol\_array | ‘-’ | | ‘/’ | 0 | 0 | ‘\*’ | 0 | ‘+’ | 0 | 0 |
| number\_array | 0 | | 0 | 4 | 64 | 0 | 2 | 0 | 3 | 1 |
| len | | 9 | | | | | | | | |
| max\_index | | 7 | | | | | | | | |
| ecx | | 0 | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| symbol\_array | ‘-’ | ‘/’ | 0 | 0 | ‘\*’ | 0 | ‘+’ | 0 | 0 |
| number\_array | 0 | 0 | 4 | 64 | 0 | 2 | 0 | 3 | 1 |
| ecx | 1 | | | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| symbol\_array | ‘-’ | 0 | ‘\*’ | 0 | ‘+’ | 0 | 0 |
| number\_array | 0 | 0 | 0 | 2 | 0 | 3 | 1 |
| len | 7 | | | | | | |
| max\_index | 5 | | | | | | |
| ecx | 4 | | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| symbol\_array | ‘-’ | 0 | ‘\*’ | 0 | 0 |
| number\_array | 0 | 0 | 0 | 2 | 4 |
| len | 5 | | | | |
| max\_index | 3 | | | | |
| ecx | 0 | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| symbol\_array | ‘-’ | 0 | ‘\*’ | 0 | 0 |
| number\_array | 0 | 0 | 0 | 2 | 4 |
| ecx | 2 | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| symbol\_array | ‘-’ | 0 | 0 |
| number\_array | 0 | 0 | 8 |
| len | 3 | | |
| max\_index | 1 | | |
| ecx | 0 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| symbol\_array | ‘-’ | 0 | 0 |
| number\_array | 0 | 0 | 8 |
| ecx | 0 | | |

|  |  |
| --- | --- |
| symbol\_array | 0 |
| number\_array | -8 |
| len | 1 |
| max\_index | -1 |

2) Ce variabile au fost folosite și cu ce scop?

Punem informațiile despre toate elementele din arborele sintactic abstract construit cu *getAST()* în doi vectori : **symbol\_array[401]** (prin simbol ma refer la un element al mulțimii {‘+’, ‘-’, ‘\*’, ‘/’}) și **number\_array[401]**. Informația necesara despre nodul *i* al AST-ului (nodul i la parcurgerea în preordine, adică rădăcina -> stânga -> dreapta) este reținută în cei doi vectori astfel:

* în symbol\_array[i], reținem valoarea numerica a simbolului dacă informația din nod reprezintă un simbol, sau 0, dacă informația din nod reprezintă un număr; [1]
* în number\_array[i], reținem valoarea numerica a informației din nod dacă aceasta reprezintă un număr nenul, sau 0, dacă informația din nod reprezintă un simbol sau numărul 0. [2]

*De ce 401 elemente?* O precizare suplimentara a temei spune : „Se vor efectua maximum 200 de operații.”, ceea ce implica 200 de simboluri + 201 numere = 401 noduri ale arborelui -> 401 elemente ale vectorilor.

Cei doi vectori vor avea întotdeauna lungimea egala cu **len**, iar când vom face parcurgerea lor, vom compara elementul curent cu următoarele 2 elemente; de aceea, trebuie sa reținem indicele la care vom opri parcurgerea și pentru asta vom folosi **max\_index**, variabila mereu egala cu (len – 2).

3) Ce funcții au fost folosite și cu ce scop?

a) **int** **string2num(char\*)** (echivalenta cu atoi(char\*)) - transforma parametrul dat într-un număr.

Rezultatul returnat este descris la [2].

b) **int** **operation\_type(char\*)** - returnează simbolul reprezentat de parametru.

Rezultatul returnat este descris, mai amplu, la [1].

c) **int execute\_operation(int, int, char)**

Rezultatul returnat este rezultatul operației

argument1 <valoare\_simbol\_argunent2> argument3.

d) **void iterate\_AST(node\*) -** parcurge arborele în preordine începând de la nodul *i* primit ca parametru și copiaza informația din nod în number\_array[i] prin *string2num* și în symbol\_array[i] prin *operation\_type*. Dacă nodul este nul, nu se executa stocarea datelor și revenim la ultimul nod valid. În caz contrar, după reținerea datelor, vom apela recursiv aceasta funcție pentru fiul din stânga și fiul din dreapta al nodului curent.

4) Implementarea programului principal

Primul pas consta în citirea arborelui în preordine de la tastatura și organizarea elementelor într-un arbore sintactic abstract cu ajutorul functiei *getAST()*. Se parcurge arborele rezultat cu funcția *iterate\_AST(node\*)*, iar informația nodului curent va fi adăugată pe ultima poziție a vectorilor **symbol\_array** și **number\_array** astfel: pe symbol\_array[i], punem rezultatul returnat de *operation\_type(dataNoduluiCurent)*, iar pe number\_array[i], punem rezultatul returnat de *string2num(dataNoduluiCurent)*.

După extragerea datelor din arbore, vom face mai multe parcurgeri ale vectorilor

de la 0 la (len-2) (**i\_loop**) și vom prelucra elementele lor pana când ajungem la len = 1. (**while\_loop**)

Dacă pe poziția curenta avem simbol valid, iar pe următoarele doua poziții avem numere valide (simboluri invalide), atunci modificam informația din poziția curenta astfel: (number\_array[i], symbol\_array[i]) = (*execute\_operation(number\_array[i+1], number\_array[i+2], symbol\_array[i])*, 0); de asemenea, se elimină elementele de pe pozițiile (i+1) și (i+2) și se actualizează lungimea și indicele maxim de parcurgere. (**j\_loop**).

Un exemplu al acestui algoritm este în primele 2 pagini ale acestui fișier,

la „1) Ideea de rezolvare a temei”.

P.S. : Sper sa nu fie vreo problemă prea mare dacă în programul principal, deși codul este în engleză , comentariile sunt în română. 😅