Proiect de atestat - Interpretor de Pseudocod

Realizator: Cîcu Timotei-Iosif

Detalii de implementare a interpretorului de pseudocod, detalii despre utilizarea interpretorului, si exemple de pseudocod valide.

1. Introducere

Am ales sa realizes acest proiect, deoarece sunt interesat in creerea limbajelor de programare compilate si interpretate.

Interpretorul a fost creat cu scopul de a invata despre limbajele de programare interpretate.

2. Prezentarea implementarii interpretorului de pseudocod

Am realizat interpretorul folosind in limbajul de programare C. Acesta are patru etape:

2.1. Etapa de lexare/tokenizare

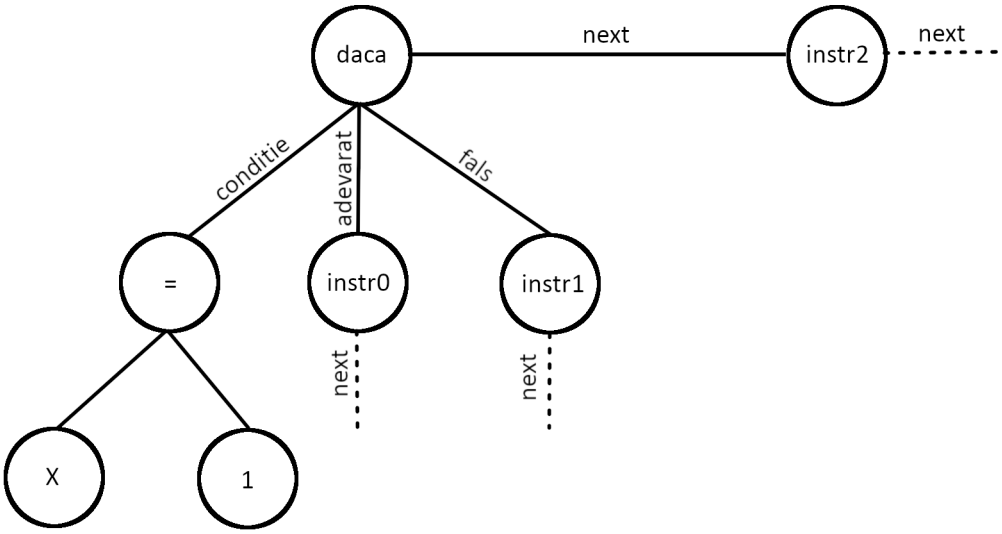
Interpretorul citeste fisierul de intrare, apoi parcurge caracter cu caracter codul citit, transformandu-l intr-o serie de tokenuri care vor fi utilizate in etapa urmatoare (etapa de parsare).

2.2. Etapa de parsare

Programul parcurge tokenurile obtinute din etapa anterioara si formeaza un arbore de executare care este utilizat la etapare urmatoare.

Expresiile aritmetice sunt transformate din scrierea infix, in scriere prefix.

Exemplu de parte de arbore de executare:



2.3. Etapa de executare

Programul parcurge arborele de executare pentru a executa instructiuniile scrise in fisierul de intrare.

Apeluri de functii

Atunci cand o functie este apelata, se copiaza intr-un hash map de variabile paramentrii functiei, iar acest hash map este pus (push) intr-un stack (stiva).

Atunci cand o functie returneaza sau se termina, se scoate (pop) din stack (stiva), memoria alocata pentru hash mapul nou de variabile este eliberata si memoria alocata pentru noul element in stack (stiva) este si ea eliberata.

Variabile

Variabilele locale sunt memorate in hash mapul de variabile asociat capului stivei, iar variabilele globale sunt memorate in hash mapul `global_var_map`.

Atunci cand nu s-a apelat nicio functie, hash mapul de variabile locale din stack (stiva) este acelasi cu hash mapul de variable globale `global_var_map`.

2.4. Etapa de garbage collection:

Variabilele de tip vector si de tip functie sunt alocate dinamic, iar atunci cand o variabila isi schimba valoarea si nu mai are referinta catre inceputul vectorului/catre functie, se poate creea un "memory leak".

Atunci cand un vector sau o functie sunt create, adresa lor de memorie este inserata intr-un hash map pentru a putea tine minte locatia lor.

Atunci cand o variabila primeste prin assignment, citire, sau apelare la functie o adresa de memorie care a fost alocata dinamic, hash mapul este actualizat si frecventa de cate variabile au referinta catre acea adresa creste.

Atunci cand o variabila care retinea o asemena adresa primeste prin assignment sau citire o noua valoare, sau se returneaza din functie si se elibereaza hash mapul de variabile locale, numarul de referinta catre acea parte de memorie scade.

Pointerele returnate de la functia `referinta(x)`, nu sunt contorizate.

Hash mapul utilizat pentru garbage collection este un hash map de frecventa

Atunci cand numarul de referinte a scazut pana la 0, se verifica cate alte adrese au un numar de referinte egal cu 0 si aceste adrese se elibereaza.

Exemple de cod in care s-ar elibera memorie folosind garbage collection

```
x <- vector(3)
x <- 10
```

sau

```
x <- vector(3)
citeste x
```

sau

```
subprogramul f() face
|    x <- vector(3)
f()
```

Implementarea hash mapului

Si variable_map si possible_garbage_map folosesc aceeasi structura de date generalizata, folosind functii pentru utilitatiile speciale (o functie de hash, de indexare, de dereferentiere o singura data, de inserare in lista inlantuita, si de cautare in lista inlantuita)

3. Compilare si utilizare a interpretorului

3.1. Compilare

Interpretorul se complileaza utilizand compilatorul `gcc` (sau echivalent) folosind urmatoarea comanda: `gcc main.c -lm -o pseudocod`

3.2. Utilizare

Interpretorul se apeleaza folosind comanda `pseudocod [fisier]` - unde fisier este fisierul care contine codul sursa

4. Sintaxa utilizata

4.1. Indentare

Indentarea sa face folosind caracterul |.

Exemplu:

```
daca 1 atunci
|    scrie "da"
```

Daca la sfarsitul liniei anterioare s-a folosit caracterul de escape \, atunci nu se va mai indenta folosind, deoarece interpretorul trateaza acest caz ca si cand nu ar fi o linie noua intre cele doua linii

Exemplu:

```
daca 1 si \
     2 atunci
|    scrie "da"
```

Instructiuniile de decizie, repetitive, si definirea functiilor au nevoie de indentare pentru a determina codul din interiorul structurii.

4.2. Expresiile (instructiunile aritmetice)

Expresiile sunt combinari valide de operatori unari si binari, variabile, valori, si apeluri la functii

Operaturul unar este `not`, iar operatorii binari sunt: `<-` (assignment), `sau`, `si`, `=`, `!=`, `<`, `>`, `<=`, `>=`, `+`, `-`, `*`, `/`, `%` (restul impartirii), `^` (ridicarea la putere).

Ordinea operatorilor

- Ordinul 9: `not`
- Ordinul 8: `^`
- Ordinul 7: `*`, `/`, `%` (restul impartirii)
- Ordinul 5: `+`, `-`
- Ordinul 4: `<`, `>`, `<=`, `>=`
- Ordinul 3: `=`, `!=`
- Ordinul 2: `si`
- Ordinul 1: `sau`
- Ordinul 0: `<-` (assignment)

Operatiile de ordin mai mare vor fi executate primele.

Exemple de expresii valide:

```
x <- 10 * 2 / 2 * f(x)
```

```
x = x != x
```

```
x <- -1
```

Exemple de expresii invalide

```
x <- 10 * 2 / 2 *
```

```
x = != x
```

```
x <- * 1
```

4.3. Variabilele

Variabilele sunt declarate prin asignare, citire, sau prin definirea unor functii (functiile definite de utilizator sunt variabile).

Numele variabilelor pot sa contina litere, caracterul. _ (underscore), si numere, dar nu pot sa inceapa cu numere.

4.4. Tipurile de variabile

Tipurile de variabile sunt: `TV_NUMAR` (numere de tip intreg si real), `TV_CHAR` (caractere), `TV_VECTOR` (vectori de alte tipuri de variabile), `TV_TIP` (returnat de functia `tip()`), `TV_ERROR` (returnat de functia `tip()`).

Prescurtarea `TV` vine de la "tip variabila".

Functia `tip()` este explicata la sectiunea despre functiile predefinite.

4.5. Declararea si apelarea functiilor

Functiile sunt definite utilizand cuvantul cheie `subprogramul`, urmat de numele subprogramul, lista de

parametrii in paranteze, si cuvantul cheie `face` astfel:

```
subprogramul nume_subprogram(p0, p1, p2, ...) face
|    cod
```

Functiile sunt apelate folosind numele subprogramul si lista de variabile sau valori in paranteze astfel:

```
nume_subprogram(v0, v1, v2, ...)
```

Daca functia nu are parametrii, atunci se va declara astfel:

```
subprogramul nume_subprogram() face
|    cod
```

Si se va apela astfel:

```
nume_subprogram()
```

4.5.1. Functii predefinite

- `sqrt(x)` - radical din x
- `floor(x)` - da partea intreaga a lui x
- `tip(x)` - returneaza tipul lui x, valoarea returnata are tip `TV_TIP`
- `vector(x)` - alocheaza memorie pentru a creea un vector de dimenisiune x
- `dimensiune(x)` - da dimensiunea vectorului x
- `referinta(x)` - returneaza o referinta catre x sub forma unui vector cu dimensiune 1, functie folosita pentru a modifica valoarea unei variabile neglobale in interiorul unui subprogram. se derefentiaza prin accesarea indexului 0
- `cast(x, tip) si schimbare_tip(x, tip)` - returneaza valoarea lui x, dar cu tip `tip`, `tip` este o variabila sau o constanta de tip `TV_TIP`

4.6. Instructiunea de decizie

Instructiunea de decizie se foloseste astfel:

```
daca expr0 atunci
|    cod daca expr0 este adevarat
```

Instructiunea de decizie cu doua ramuri:

```
daca expr0 atunci
|    cod daca expr0 este adevarat
|altfel
|    cod daca expr0 este fals
```

Instructiunea de decizie cu mai mult de doua ramuri:

```
daca expr0 atunci
|       cod daca expr0 este adevarat
|altfel daca expr1 atunci
|       |    cod daca expr0 este fals si expr1 este adevarat
|       |altfel
|       |    cod daca expr0 este fals si expr1 este fals
```

Observatie: Spre deosebire de celelalte structuri dupa care trebuie sa urmeze indentatie, dupa cuvantul cheie `altfel` poate urma o expresie/instructiune direct.

4.7. Instructiuni repetitive cu numar nedeterminat de pasi

4.7.1. Instructiuni repetitive cu test initial

Instructiunile repetitive cu test initial vor executa codul din interiorul structurii doar daca expresia este adevarata.

4.7.1.1. Instructiunea cat timp ... executa

Instructiunea `cat timp ... executa` se foloseste astfel:

```
cat timp expr0 executa
|    cod daca expr0 este adevarat
```

4.7.1.2. Instructiunea pana cand ... executa

Instructiunea `pana cand ... executa` se foloseste astfel:

```
pana cand expr0 executa
|    cod daca expr0 este fals
```

4.7.2. Instructiuni repetitive cu test final

Instructiunile repetitive cu test final vor executa cel putin o data codul din structura, indiferent daca expresia este adevarata, apoi vor executa codul din nou, doar daca expresia este adevarata

4.7.2.1. Instructiunea repeta ... cat timp

Instructiunea `repeta ... cat timp` se foloseste astfel:

```
repeta
|    cod daca expr0 este adevarat
|cat timp expr0
```

4.7.2.2. Instructiunea repeta ... pana cand

Instructiunea `repeta ... pana cand` se foloseste astfel:

```
repeta
|    cod daca expr0 este fals
|pana cand expr0
```

4.8. Instructiunea repetitiva cu numar determinat de pasi

4.8.1. Instructiunea pentru ... executa

Instructiunea `pentru ... executa` se foloseste astfel:

```
pentru expr0, expr1, expr2 executa
|    cod daca expr1 este adevarata
```

`expr0` va fi executata, apoi se va testa daca `expr1` este adevarata si daca este adevarata, se va executa codul din structura, la sfarsitul codului din structura, se va executa `expr2` si se va intoarce la testare.

Observatie: spre deosebire de pseudocodul invatat la scoala, aceasta instructiune are aceeasi functionalitate ca si un `for` din C.

4.9. Instructiuni de intrare/iesire

4.9.1. Instructiuni de intrare

4.9.1.1. Instructiunea citestetot

Instructiunea `citestetot` va fi urmata de o variabila si se va citi de la tastatura un sir de caractere pana la caracterul de linie noua si se va stoca in variabila sub forma unui vector de caractere.

Exemplu:

```
citestetot x
```

4.9.1.2. Instructiunea citeste

Instructiunea `citeste` va fi urmata de un numar nedeterminat de variabile si se va citi de la tastatura cuvant cu cuvant si se va determina daca poate fi interpretat ca numar. Daca poate fi interpretat ca numar, atunci va fi memorat ca numar (`TV_NUMAR`) in variabila corespunzatoare numarului cuvantului. Daca nu poate fi interpretat ca numar, atunci se va stoca in variabila corespunzatoare ca un vector de caractere.

Exemplu:

```
citeste x, y, z
```

Si se citeste de la tastatura:

```
abc 123 b1
```

Atunci x va memora vectorul de caratere “abc”, y va memora numarul 123, z va memora vectorul de caratere “b1”.

Exemplu:

```
citeste x
```

Si se citeste de la tastatura:

```
12.12
```

Atunci x va memora numarul 12.12 (doisprezece virgula doisprezece).

4.9.2. Instructiuni de iesire

4.9.2.1. Instructiunea scrie

Instructiunea scrie va fi urmata de un numar nedeterminat de variabile si valori care vor fi afisate in consola in functie de tipul lor:

- TV_NUMAR - va afisa numarul
- TV_CHAR - va afisa caracterul
- TV_VECTOR - daca este un sir de caractere, atunci se va afisa sirul de caractere. Daca nu este sir de caractere, atunci interpretorul va da eroare
- TV_TIP - va afisa TV_NUMAR , TV_CHAR , TV_VECTOR , TV_TIP , sau TV_ERROR in functie de valoarea variabilei
- TV_ERROR - se va trata precum variabila are tip de numar (TV_NUMAR)

4.10. Vectori

4.10.1. Declarea vectoriilor

Vectorii pot fi declarati in doua moduri.

```
x <- vector(3)
```

care va creea un vector de dimensiune 3

si

```
x <- (1, 2, 3)
```

care va creea un vector de dimensiune 3 cu valoriile 1, 2, 3.

Vectorii pot avea mai multe dimesiuni astfel:

```
x <- vector(3)
x[0] <- vector(3)
```

si

```
x <- ((1, 2, 3), 2, 3)
```

4.10.2. Indexarea vectoriilor

Vectorii pot fi indexati utilizand urmatoarea sintaxa:

```
x[expr0][expr1]...[exprn]
```

Exemplu:

```
x <- ((1, 2, 3), 2, 3)
scrie x[0][1], " ", x[2]
```

va afisa:

```
2 3
```

4.11. Cuvatul cheie sursa

Acest cuvant cheie este singurul care este interpretat in etapa de lexing, similar cu C. Codul din noul fisier este inserat in lista de tokenuri exact in locul in care apare, iar mai tarziu este tratat la fel ca si codul din sursa principala.

Dupa cuvantul cheie sursa, se va scrie locatia fisierului care sa fie inclus.

Exemplu:

Fisierul `main.pseudo`

```
sursa "alta_sursa.pseudo"
f()
```

Fisierul `alta_sursa.pseudo`

```
subprogramul f() face
|   scrie "123"
```

Va afisa:

```
123
```

4.12. Comentarii

Comentariile pot fi scrise utilizand caracterul # similar cu python. Tot ce urmeaza dupa caracterul # va fi ignorat pana la urmatoarea linie.

Exemplu:

```
# daca 1 atunci
# |  scrie 2
```

Nu va afisa nimic.

5. Exemple de cod in directiva "examples"

Am realizat cateva programe folosind pseudocodul/limbajul de programare inventat de mine pentru a afisa abilitatiile acestuia.

5.1. X si 0

Am implementat jocul de x si 0 in pseudocod folosind citirea si afisarea in consola. Tabla de joc este memorata intr-un vector/tablou bidimensional.

Atunci cand trebuie intrata casuta in care se va pune simbolul (x sau 0), se va afisa un prompt care evidentiaza a cui tura este. Inputul este reprezentat de un numar intre 1 si 3 pentru linie si un numar intre 1 si 3 pentru coloana, acestea fiind separate printr-un spatiu.

Programul va determina automat cazurile de castig sau egalitate.

5.2. Sah

Am implementat o versiune incompleta a jocului de sah in pseudocod folosind citirea si afisarea in consola. Tabla de sah este memorata intr-un vector/tablou bidimensional.

Acesta are aproape toate miscarile, cu exceptia miscarii "en passant". Acesta poate detecta egalitate doar in cazurile stalemate, si 50 move rule, dar nu si in cazurile dead position rule, sau threefold repetition. In rest, jocul este complet.

Programul va detecta automat sah-ul si sahmat-ul.

Miscariile se scriu la promptul `mutare>`. Miscariile nu se specifica folosind notatia de sah, ci se specifica casuta de inceput a piesei si unde trebuie mutata folosind sintaxa litera-numar (coloana-linie; exemplu: `D2 D4`). Programul va detecta automat daca mutarea este valida.

In cazul in care o miscare este invalida, programul v-a explica criteriul folosit pentru a detecta ca miscarea este invalida. Tabla de sah este afisata la fiecare tura din perspectiva persoanei care trebuie sa mute.

5.3. Probleme de eficienta

Am rezolvat o parte (10 probleme) din problemele propuse la proba practica pentru examenul de atestare a competentelor profesionale a absolventilor claselor de matematica-informatica si matematica-informatica, intensiv informatica, in pseudocod.

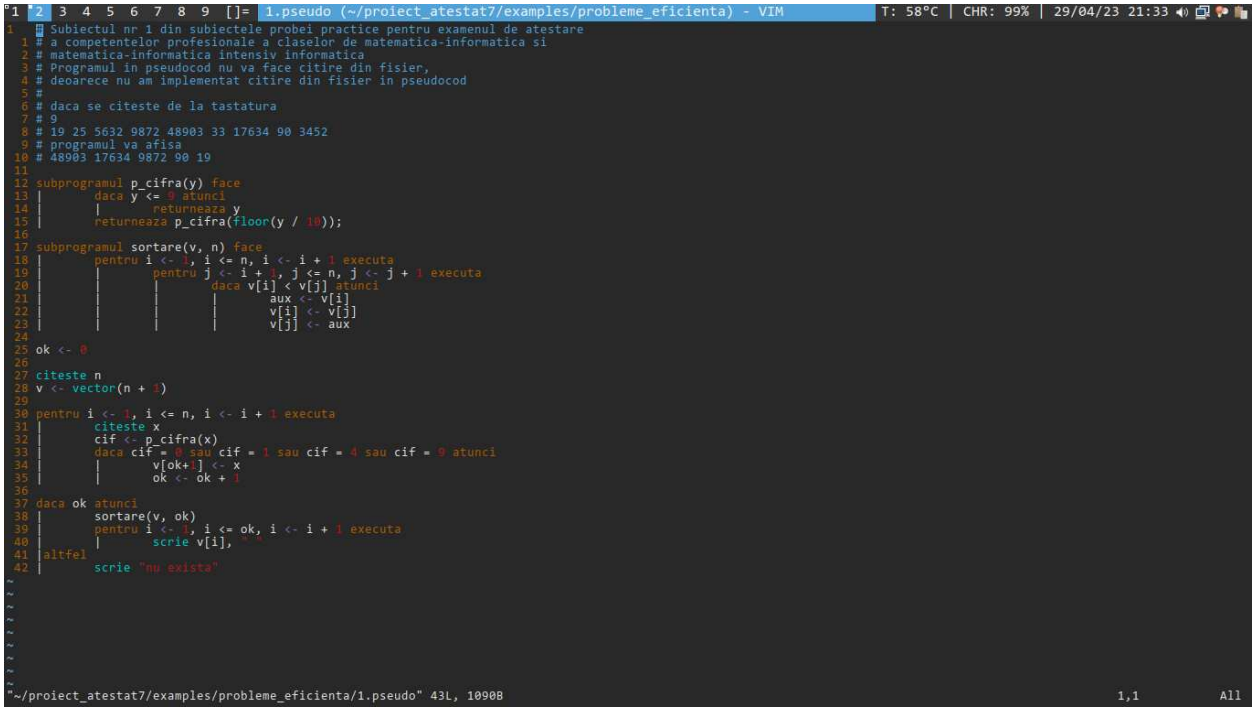
Deoarece nu am implementat citire si afisare in fisier in pseudocod, am modificat cerinta unor probleme, astfel incat sa foloseasca citirea de la tastatura si afisarea in consola (stdin si stdout).

6. Pentru utilizatorii de linux

Programul se poate instala folosind comanda `sudo make install`

7. Pentru utilizatorii de vim

In folderul/directiva `vim` am inclus fisierul `pseudo.vim` pentru sintaxa astfel incat vim sa foloseasca high-lightarea de sintaxa pentru pseudocod, de exemplu:



Fisierul trebuie plasat in `~/.vim/syntax` sau in alt folder echivalent.

Cuprins