Pseudocod

(explicat usor)

Cuprins:

11. Tablouri (inca nu exista)
    a. Sintaxa tablouri 1D
    b. Sintaxa tablouri nD
    c. Folosirea tablourilor impreuna cu string-uri
    d. Citirea tablourilor
    e. Exemplu program: se citesc n numere, sa se afiseze in ordinea inversa
12. Convertirea (inca nu exista)
    a. Din numar in string
    b. Din string in numar
13. Functii (inca nu exista)
    a. Sintaxa
    b. Tipuri de functii
14. Folosire functii matematice (inca nu exista)

1. Introducere

Inainte sa incep sa vorbesc despre pseudocod, as vrea sa spun cate ceva despre scopul acestei limbi de programare, de ce ar fi useful un astfel de interpret si cateva ganduri legate de development-ul ei.
Ideea de un interpret pentru pseudocod a aparut la una din primele ore de informatica, cand profesoara incerca sa ne invete conceptele de programare intr-o limba extrem de ciudata pe care o numea pseudocod. Si ce era si mai ciudat era ca nu se facea pe calculator nimic. Totul scris la tabla si in caiet. Mi s-a parut ciudat sa inveti concepte relativ abstracte ca arrays, loops si altele fara sa te atingi de calculator. Dar totusi, inteleg oarecum aceasta abordare. Pseudocod este o limba extrem de simpla, care contine doar concepte de baza (dar totusi este turing complete dupa cum se va demonstra mai tarziu). Totusi pastreaza putin (spre deosebire de Python) din sintaxa C++, ceea ce o face usor un candidat decent pentru a invata conceptele si a sari la C++ mai tarziu. Astfel mi-a venit o idee: de ce sa nu creez un interpret care poate fi folosit pentru a preda elevilor incepatori conceptele de programare de baza?

De ce? Pai din cauza ca nu e usor. Building a programming language is no joke, si am aflat asta dupa ce, naiv fiind, m-am aruncat cu capul inainte sperand sa fac unul fara probleme. Prima versiune, inceputa in iulie terminata la inceputul lui septembrie poate fi gasita aici https://github.com/vitelariu/pseudocode-legacy. (atentie, nu e utilizabila). Tot prin septembrie imi "promovam" produsul prin server-ul de discord RoAlgo (https://discord.gg/8bZdbJsbME), reusind cumva, nu stiu cum, sa atrag atentia unui moderator. El imi spusese ca facut ca lumea, ar putea fi o chestie destul de interesanta, si tot el, daduse anunt pentru a strange niste contributori la proiect (spoiler: au exista doar 2 fara mine). Dupa cum spuneam, in momentul cand scriu acest pdf, suntem 3 oameni care au contribuit la proiect. Eu, care am scris cea mai mare parte din cod, Alecu Stefan Iulian, care a scris script-ul de build si Dumitru Iliei, care a fixat cateva bug-uri, a formatat si scris ceva cod, plus mi-a dat cateva idei cand eram blocat.
In prezent, pseudocod este o limba de programare perfect utilizabila, care si-a atins si depasit scopurile initiale.
Acestea fiind spuse, sa nu mai zabovim si sa trecem la treaba :)

!!! Acesta nu este un tutorial. Daca citesti asta imi asum ca stii deja conceptele de baza de programare si esti aici doar ca sa vezi care e sintaxa pseudocodului. Probabil esti profesor, care vrea sa stie cum se foloseste pseudocod, ca mai apoi la scoala sa-i inveti pe elevi.
Aici nu explic conceptele, explic sintaxa!

2. Terminologie

Se definesc urmatoarele lucruri:
{expr} = orice tip de expresie
{var} = orice tip de nume de variabila
daca e folosit de mai multe ori, aceasta indica aceeasi variabila
{cod} = un bloc de cod, care poate fi orice

Identarea se face cu 4 spatii sau un tab.

3. Expresii

a. Baza

Orice expresie consista dintr-o combinatie de operanzi si operatori. Spre ex: *2 + 3*

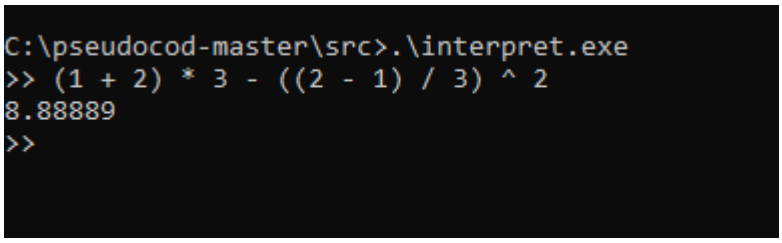
```
C:\pseudocod-master\src>.\interpret.exe
>> 2 + 3
5
>>
```

In aceasta expresie operanzii sunt 2 si 3, iar operatorul este “+”.
Operanzii nu sunt limitati, ei pot fi numere, string-uri, variabile. De asemenea operatorul poate lua multe forme ca “+”, “-”, “*” etc. Iata o lista intreaga:

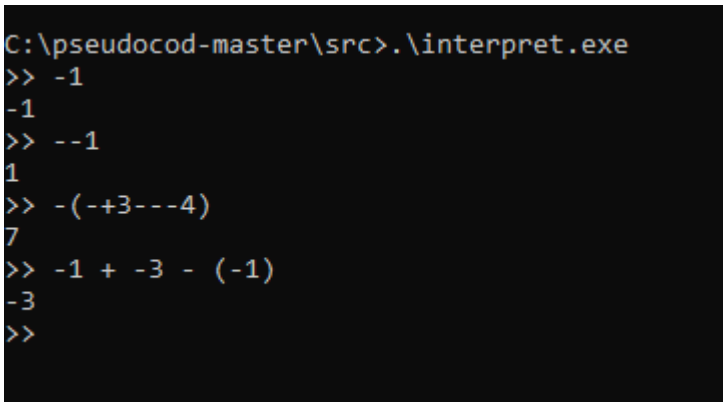
Operator	Semnificatie
+	suma
-	diferenta
*	produs
/	cat
%	modulo
^	putere

```
C:\pseudocod-master\src>.\interpret.exe
>> -2 ^ 2 + 10 / 5 * 2 + 3 % 1
0
>>
```

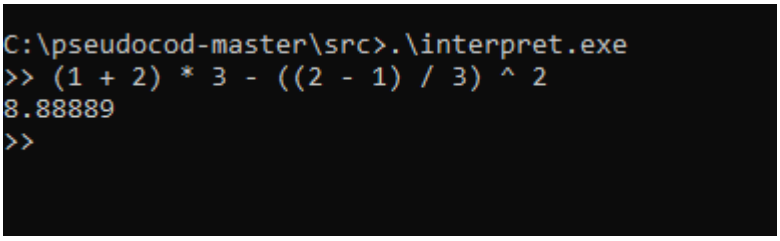
De asemenea putem include si parantezele “(”, “)”, pentru a manipula ordinea efectuarii operatiilor:



Exista si semnele “-” si “+” care schimba (sau nu schimba) semnul expresiei. Acestea pot fi folosite in mai multe moduri, dar cea mai readable este folosind parantezele:



De asemenea parantezele patrate “[]” care converteste din numar real in numar intreg:

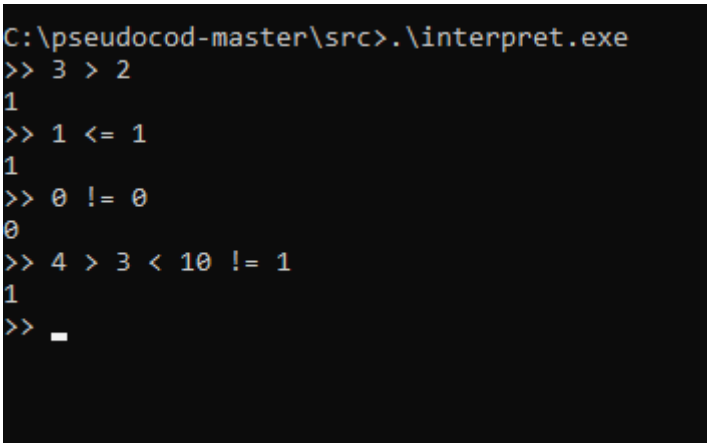


b. Comparari, logica si sintaxa mai ciudata

In pseudocod putem folosi si urmatoarele semne pe post de operatori: (ele vor returna 1 sau 0)

Operatori	Semnificatie
=	egal
!=	diferit
<	mai mic
>	mai mare
<=	mai mic egal
>=	mai mare egal

Folosirea lor:



Se poate observa ca in pseudocod compararile se pot scrie ca in Python.

Operatorii de logica mai returneaza 0 si 1:

Operator	Semnificatie
si	returneaza 1 daca “a and b”
sau	returneaza 1 daca “a or b”

Exemplu:

```
C:\pseudocod-master\src>.\interpret.exe
>> !0 si 1
1
>> 1 si 1 > 2
0
>> 0 sau 3
1
>>
```

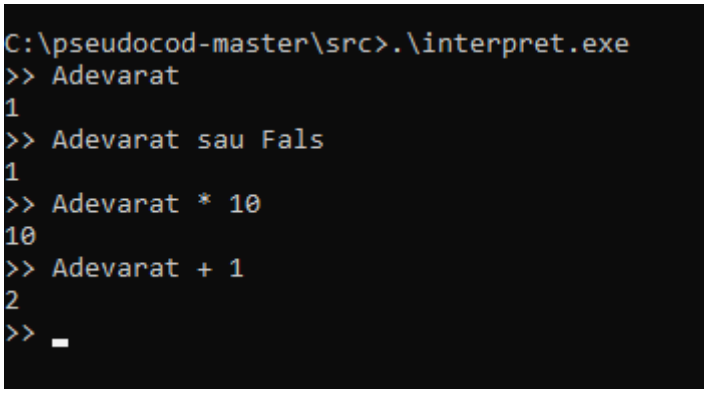
c. String-uri si operatii cu string-uri

Putem crea expresii folosind doar string-uri.
Operatiile suportate sunt adunarea a doua string-uri si inmultirea unui string cu un numar:

```
C:\pseudocod-master\src>.\interpret.exe
>> "Hello world"
"Hello world"
>>
>> "Hello" + " " + "world"
"Hello world"
>> ("Hello" + " " + "world") * 2
"Hello worldHello world"
>>
```

d. Booleans

Pentru booleans exista “Adevarat” si “Fals” (sunt tot numerele 1 si 0 doar ca scrise sub alta forma):



e. Ordinea operatiilor in pseudocod

Mai jos aveti un tabel care contine toate operatiile posibile descrise mai sus. Cu atat o operatie este mai sus, cu atat are mai multa prioritate:

!, +, - (semnele negativ pozitiv)
^
*, /, %
+, - (adunari, scaderi)
>, <, >=, <=, =, !=
si
sau

4. Scrie

a. Sintaxa

Poate ati observat deja ca expresiile “merg” doar cand executam interpretul din terminal. Cand scrie o expresie intr-o fila si o executa din ea, nu se intampla nimic. Pai, de fapt, se intampla. Expresia este calculata, dar spre deosebire de interpretarea din terminal, expresia nu este scrisa direct.

In schimb folosim comanda *scrie,* urmat de {expr} separate prin virgula daca este cazul:

```
C:\pseudocod-master\src>type file.txt
scrie "Hello World!\n"
scrie "Rezultatul lui 1 + 1 = ", 1 + 1, "\n"
C:\pseudocod-master\src>.\interpret.exe file.txt
Hello World!
Rezultatul lui 1 + 1 = 2
```

5. Variabile

a. Sintaxa

Operatorul de asignare este “←”.
{var} ← {expr}

```
C:\pseudocod-master\src>type file.txt
x <- 69
scrie x, "\n"
C:\pseudocod-master\src>.\interpret.exe file.txt
69
```

6. Citeste

a. Sintaxa

Putem sa citim si de la tastatura variabila folosind keyword-ul *citeste,* urmata de {var} separati prin virgula:

```
C:\pseudocod-master\src>type file.txt
citeste n
citeste x, y, z
scrie n, " ", x + y + z, "\n"
C:\pseudocod-master\src>.\interpret.exe file.txt
69
1 2 3
69 6

C:\pseudocod-master\src>.\interpret.exe file.txt
69 1 2 3
69 6
```

b. Diferite tipuri de “citeste”

Dupa cum puteti vedea, tipul variabilei default la citire este numarul real. Putem schimba tipul adaugand la sfarsit una din urmatoarele:

(numar natural) / (numere naturale)
(numar intreg) / (numere intregi)
(numar real) / (numere reale)
(string) / (text)
(bool) / (logica)

! Doar pentru ca se citeste un nr intreg spre exemplu in variabila x, nu inseamna ca variabila x nu va putea fi modificata pentru a deveni de tip real. Arunca o privire la exemplul urmator:

```
C:\pseudocod-master\src>type file.txt
citeste x (numar intreg)
x <- x + 0.1
scrie x, "\n"
C:\pseudocod-master\src>.\interpret.exe file.txt
0.9

In fisierul: file.txt: eroare -> S-a citit gresit valoarea!
   1 | citeste x (numar intreg)

C:\pseudocod-master\src>.\interpret.exe file.txt
1
1.1
```

7. Bucle “cat timp”

1. Sintaxa

Buclele “cat timp” pot fi create folosind keyword-ul *cat timp*, urmat de {expr}, si de keyword-ul *executa.* Pe urmatoarele linii, identat de un tab (sau 4 spatii) mai mult decat identarea statement-ului

```
C:\pseudocod-master\src>type file.txt
i <- 0
cat timp i <= 10 executa
        scrie i, " "
        i <- i + 1
C:\pseudocod-master\src>.\interpret.exe file.txt
0 1 2 3 4 5 6 7 8 9 10
C:\pseudocod-master\src>
```

b. Exemplu program: citeste n numere, sa se afiseze patratul lor

```
C:\pseudocod-master\src>type file.txt
citeste n
i <- 0

cat timp i < n executa
        citeste x
        scrie x * x, " "
        i <- i + 1
C:\pseudocod-master\src>.\interpret.exe file.txt
5
1 2 3 4 5
1 4 9 16 25
C:\pseudocod-master\src>
```

8. Bucle “pentru”

a. Sintaxa de baza

Putem crea o bucla “pentru” folosind keyword-ul *pentru*, urmat de o asignare, urmat de o virgula, o expresie (care reprezinta upper bound-ul sau lower bound-ul lui {var}) si keyword-ul *executa*.

pentru {var} ← {expr}, {expr2} executa
    {bloc}

```
C:\pseudocod-master\src>type file.txt
pentru i <- 0, 10 executa
        scrie i, " "
scrie "\n"
pentru i <- 10, 0 executa
        scrie i, " "
C:\pseudocod-master\src>.\interpret.exe file.txt
0 1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1 0
C:\pseudocod-master\src>
```

Dupa cum puteti vedea, {var} creste, sau descreste in functie de caz. Dar daca vrem sa crestem sau descrestem cu alti “pasi”?

b. Bucla “pentru” cu numar diferit de pasi

Putem face asta folosind un al treilea parametru la bucla “pentru”.
pentru {var} ← {expr}, {expr2}, {expr3} executa
    {bloc}

```
C:\pseudocod-master\src>type file.txt
pentru i <- 0, 100, i+20 executa
        scrie i, " "
C:\pseudocod-master\src>.\interpret.exe file.txt
0 20 40 60 80 100
C:\pseudocod-master\src>
```

Echivalenta in bucla “cat timp” a programului de sus:

{var} ← {expr}
cat timp {var} <= {expr2} executa
    {bloc}
    {var} ← {expr3}

De asemenea se poate folosi o variabila declarata inainte:

```
C:\pseudocod-master\src>type file.txt
i <- 0
pentru i, 10 executa
        scrie i, " "
C:\pseudocod-master\src>.\interpret.exe file.txt
0 1 2 3 4 5 6 7 8 9 10
C:\pseudocod-master\src>
```

9. Bucle “executa pana cand”

a. Sintaxa

Putem folosi bucla “executa pana cand” utilizand sintaxa:

executa
    {bloc}
pana cand {expr}

```
C:\pseudocod-master\src>type file.txt
i <- 0
executa
        scrie i, " "
        i <- i + 1
cat timp i < 10
C:\pseudocod-master\src>.\interpret.exe file.txt
0 1 2 3 4 5 6 7 8 9
C:\pseudocod-master\src>
```

10. Afirmatiile “daca / altfel daca / altfel”

a. Sintaxa

Putem scrie o afirmatie “daca” foarte simplu folosind keyword-ul *daca*, urmat de [expr] si *atunci*.

daca [expr] atunci
    {bloc}

De asemenea putem folosi si altfel “altfel daca”, “altfel” tot la fel, dupa o afirmatie “daca”.

```
C:\pseudocod-master\src>type file.txt
daca Adevarat atunci
        scrie "salut"
altfel daca 1 > 0 atunci
        scrie "salut2"
altfel daca 0 atunci
        scrie "salut3"
altfel
        scrie "salut4"
C:\pseudocod-master\src>.\interpret.exe file.txt
salut
C:\pseudocod-master\src>
```