Tema LFA 2020-2021
Interpretor Glypho

Andrei-Gabriel POPESCU, Andrei DUMITRESCU, George-Daniel MITRA

Decembrie 2020

Rezumat

Tema constă în implementarea, într-un limbaj de programare la alegere, a unui interpretor pentru limbajul Glypho. Un interpretor este un program care execută un fișier sursă scris într-un limbaj de programare, fără a necesita compilarea acestuia în prealabil.

1 Specificații temă

1.1 Cerință

Să se implementeze un interpretor pentru limbajul Glypho, care citește codul sursă dintr-un fișier de intrare si afișează la ieșirea standard rezultatul.

Arhiva trebuie să fie zip. Nu rar, 7z, ace sau alt format ezoteric. Fișierul Makefile și fișierul README trebuie să fie în rădăcina arhivei, nu în vreun director.

Fișierul trebuie să se numească README, nu readme, ReadMe, README.txt, readme.txt, read-me.doc, rEADME, README.md sau alte variante asemănătore sau nu.

Nerespectarea oricărui aspect menționat mai sus va duce la nepunctarea temei.

1.2 Specificații program

1.2.1 Rularea

Programul primește următoarele argumente din linia de comandă:

- input: obligatoriu, reprezintă numele fișierului sursă
- base: opțional, reprezintă baza în care sunt reprezentate numerele pentru intrări/ieșiri. Implicit se folosește baza 10

Pentru a permite apelarea executabilului cu argumente, regula de run din Makefile trebuie să aibă următoarea formă:

```
run: build
    ./<exec> $(input) $(base)
```

Exemplu de rulare:

```
make run input=code.gly base=16
```

1.2.2 Intrări

Programul va citi dintr-un fișier de intrare codul sursă scris in limbajul Glypho, specificat, pe larg, în secțiunea 2. Fișierul de intrare va avea extensia .gly.

Codul este format din glife, simboluri afișabile, cu valori ASCII între 33 și 126. Orice alt caracter se ignoră.

1.2.3 Ieșiri

Programul afișează răspunsul la ieșirea standard dacă programul este corect sau afișează eroarea la ieșirea de eroare.

Formatul mesajului de eroare este următorul (fără spații):

Error:<instructiune>

sau

Exception:<instructiune>

<instructiune>reprezintă a câta instrucțiune cauzează eroarea.

Tipurile de erori sunt documentate în secțiunea 2.6.

Valoarea de retur a programului este 0 dacă nu există erori, -1 dacă există erori sintactice, -2 dacă există excepții(erori la rulare).

În cazul în care detectează o eroare sintactică, programul nu începe execuția, deci nu va afișa nimic la ieșirea standard.

În cazul unei erori de execuție, programul va afișa normal până la întâlnirea erorii. În momentul în care detectează eroarea, programul încetează execuția.

1.2.4 Suport

Pentru această temă nu există schelet de cod.

Vi se pune la dispoziție un model de Makefile și un executabil care rezolvă ocmplet tema.

1.2.5 Versiuni

Mașina de test are instalat Ubuntu 18.04 LTS si următoarele versiuni:

- gcc/g++: 9.2.1 20191102
- clang: 7.0.0
- bison: 3.0.4
- javac: 11.0.5
- python: 2.7.17, 3.8.0
- python numpy: 1.16.6(2.7), 1.18.1(3.8)
- python ply: 3.11

1.2.6 Limbaje acceptate

Limbajele acceptate inițial sunt C, C++, Java și Python, urmând ca ulterior lista sa poată fi extinsă, dacă este cazul.

1.2.7 Observație

Dacă aveți nevoie de modificări, puteți posta pe forum (moodle). Dacă cererea va fi aprobată de către cineva din echipă, vom actualiza mașina pe care sunt testate temele și vom anunța acest lucru pe forum în momentul în care se fac modificarile.

1.2.8 Încărcare temă

Tema trebuie încărcată pe vmchecker [1].

2 Descrierea limbajului

2.1 Glypho: Glyph-O(riented)

Considerăm un limbaj de programare care este independent de simbolurile utilizate de la o instrucțiune la alta. Pentru fiecare punct din execuție, interpretorul limbajului trebuie sa descifreze intrucțiunea pe care simbolurile o reprezintă. Dacă restrângem lungimea unei instrucțiuni la un număr fixat n, putem enumera toate instrucțiunle posibile. În implementarea temei, interpretorul va citi fiecare instrucțiune in acest limbaj, o va procesa și va afisa rezultatul. Despre formatul fiecărei instrucțiuni, aveti detalii mai jos.

2.2 Instrucțiuni

O instrucțiune în Glypho are lungime 4, semnificația ei fiind determinată de poziția simbolurilor unice. Instrucțiunile nu se suprapun.

Asociem fiecărui simbol dintr-un bloc de 4 un număr de la 0 la 3 în felul următor:

- Primul simbol e întotdeauna 0
- Pentru fiecare apariție a unui simbol întâlnit anterior, folosim valoarea asociată lui
- Pentru fiecare simbol nou, folosim următoarea cifră nefolosită anterior

2.2.1 Exemple

Fie șirul ASAP. A va avea asociată valoarea 0. S este un simbol nou, îi asociem 1. A a mai fost întâlnit, folosim 0. P este nou, folosim 2. Codul asociat lui ASAP este 0102.
Similar, pentru următoarele șiruri:

- OKAY are codul asociat 0123
- COOL are codul asociat 0112
- CASA are codul asociat 0121

O altă metodă de a obține codul asociat e să eliminăm caracterele duplicatele din șirul original și să folosim pentru fiecare simbol indicele în șirul procesat.

- $ASAP \rightarrow ASP$

Asocierea simbolurilor cu indicii corespunzători:

A	S	P
0	1	2

Deci ASAP → 0102.

- $OKAY \rightarrow OKAY$

Asocierea simbolurilor cu indicii corespunzători:

O	K	A	Y
0	1	2	3

Deci OKAY → 0123.

- $COOL \rightarrow COL$

Asocierea simbolurilor cu indicii corespunzători:

C	O	L
0	1	2

Deci COOL → 0112.

- $CASA \rightarrow CAS$

Asocierea simbolurilor cu indicii corespunzători:

C	A	S
0	1	2

Deci CASA → 0121.

2.2.2 Optimizarea reprezentării (opțional)

Codul rezultat poate fi interpretat ca fiind un număr în baza 4 [2]. Valoarea maximă este: $0123_{(4)} = 4^0 * 3 + 4^1 * 2 + 4^2 * 1 = 27$.

Observație Unele numere în baza 4 mai mici decât $0123_{(4)}$ nu reprezintă coduri valide.

- 0020: 2 nu poate apărea fără să fi apărut 1 înaintea lui
- 0132: 3 nu poate apărea înaintea lui 2.

Pentru optimizare, puteți să eliminați valorile inutile. Singurul număr care conține un 3 este 0123, ceea ce înseamnă că restul ar putea fi interpretate ca fiind în baza 3. Valoarea dinaintea lui este: $0122_{(3)} = 3^0 * 2 + 3^1 * 2 + 3^2 * 1 = 17$. Deși nu este un număr în baza 3, dacă l-am interpreta ca fiind, ar rezulta: $0123_{(3)} = 3^0 * 3 + 3^1 * 2 + 3^2 * 1 = 18$ [3].

2.3 Stiva

Accesul la memorie în Glypho se face prin intermediul unei stive. Pentru a asigura funcționalitate mai mare decât a unui automat cu stivă, se adaugă niște instrucțiuni care permit accesul la toate elementele stivei. Stiva poate avea orice număr de elemente la un moment dat. Elementele stivei sunt numere întregi.

2.4 Setul de intrucțiuni

- 0000 ..... NOP - operația nulă
- 0001 ..... Input - citește un număr de la intrarea standard și îl va pune în vârful stivei
- 0010 ..... Rot - scoate un element din vârful stivei și îl va pune la coadă
- 0011 ..... Swap - interschimbă elementele din vârful stivei
- 0012 ..... Push - va pune în vârful stivei un nou element cu valoarea 1
- 0100 ..... RRot - scoate un element de la coada stivei și îl va pune în vârf
- 0101 ..... Dup - duplică elementul din vârful stivei
- 0102 ..... Add - scoate două elemente din vârful stivei, le calculează suma și pune rezultatul în vârful stivei
- 0110 ..... L-brace - va ignora codul până la paranteza corespunzătoare, dacă elementul din vârful stivei este 0
- 0111 ..... Output - scoate elementul din vârful stivei și îl afișează la ieșirea standard
- 0112 ..... Multiply - scoate două elemente din vârful stivei, le calculează produsul și pune rezultatul în vârful stivei
- 0120 ..... Execute - scoate patru elemente din vârful stivei și le interpretează ca pe o instrucțiune
- 0121 ..... Negate - scoate elementul din vârful stivei și pune înapoi inversul lui
- 0122 ..... Pop - scoate un element din vârful stivei
- 0123 ..... R-brace - execuția revine la elementul L-brace asociat

2.5 Instrucțiunea Execute

La întâlnirea instrucțiunii execute, se extrag patru valori de pe stivă (a_0, a_1, a_2, a_3). Ordinea în care sunt interpretate este ordinea de extragere, nu ordinea de introducere.

Algoritmul de interpretare este cel descris la secțiunea 2.2, cu mențiunea că se poate aplica pe valori numerice, nu doar pe simboluri.

Pentru a permite verificarea statică a erorilor, instrucțiunea execute nu va executa instrucțiunile L-brace și R-brace, pentru că schimbă structura codului.

Execute poate executa o instrucțiune execute.

2.6 Erori posibile

2.6.1 Erori sintactice

Există următoarele cazuri de eroare, enumerate în ordinea priorității:

1. Lungimea codului (numărul de simboluri afișabile) nu este divizibilă cu 4. Dacă se întâmpla asta, nu se poate construi ultima instrucțiune.
2. Există o paranteză închisă care nu are corespondent.
3. Există o paranteză deschisă care nu e închisă până la finalul programului

2.6.2 Erori de execuție (excepții)

Excepțiile apar în momentul în care o instrucțiune nu poate fi executată din cauza configurației stivei:

- Operații care au nevoie de cel puțin un element, dar au stiva vidă (pop, rot, rrot, dup, L-brace, output)
- Operații care au nevoie de mai multe elemente, dar nu au suficiente (add, multiply, swap, execute)
- Date la intrare care nu respectă formatul (nu sunt numere valide în baza curentă)

3 Checker

Vi se pune la dispoziție un checker cu teste publice, care dă punctaje de până la 256(200 + 56 bonus).

200 de puncte (fără bonus) pot fi obținute fără a implementa citire și afișare în altă bază diferită de 10.

160 din cele 200 de puncte(80%) pot fi obținute folosind reprezentări pe 64 de biți (long long int).

32 de puncte (20 + 5 + 7 bonus) din cele 256 sunt acordate pentru verificarea erorilor.

Timp de lucru: 16 zile.

Deadline: 18 Decembrie 2020 , 23:59. Upload-ul va rămâne deschis pâna la ora 05:00 a doua zi.

Bibliografie

[1] vmchecker LFA

[2] Interpretarea unui program Glypho

[3] Invoker

[4] elf tema

[5] Laborator 1 SO: Makefile

[6] Glypho