Tema 1 - Prefix AST

- Deadline: 22.11.2019 23:55
- Data publicării: 08.11.2019, 23:55
- Responsabili:
  - Teodor Dutu
  - Radu Nicolau

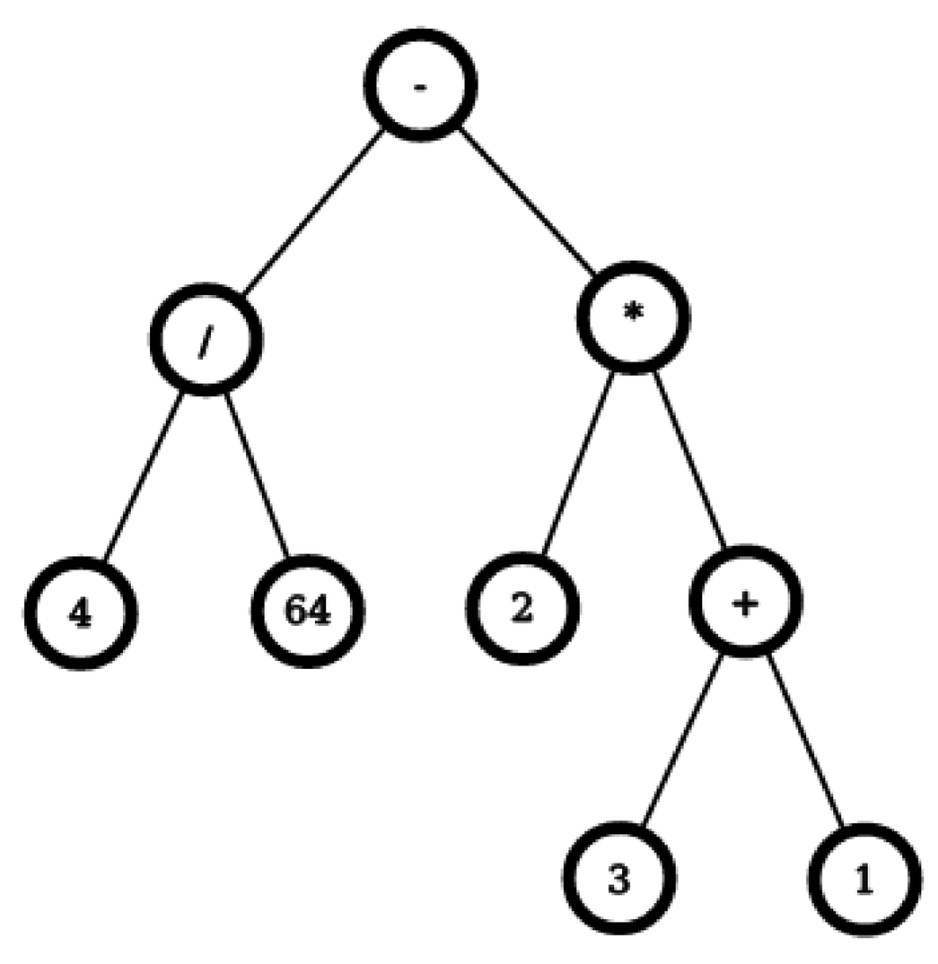
Enunț

Să se implementeze un program în limbaj de asamblare care efectuează evaluarea unei expresii matematice prefixate și apoi afișează rezultatul la `stdout`. Numerele ce apar în expresie sunt numere întregi cu semn, pe 32 de biți, iar operațiile ce se aplică lor sunt: `+, -, /, *`. Expresia prefixată va fi primita sub forma unui AST (abstract syntax tree) în urma apelului unei funcții externe - `getAST()`. Citirea se va face de catre fucția `getAST()` de la tastatură.

Arbore sintactic abstract

Arborii sintactici abstracți sunt o structură de date cu ajutorul căreia compilatoarele reprezintă structura unui program. În urma parcurgerii AST-ului, un compilator generează metadatele necesare transformării din cod de nivel înalt în cod assembly. Puteți găsi mai multe informații despre AST aici.

Reprezentarea sub forma unui AST a unui program/expresii are avantajul de a defini clar ordinea evaluarii operațiilor fară a fi necesare paranteze. Astfel expresia `4 / 64 - 2 * (3 + 1)` poate fi reprezentată sub forma:



Implementare

Programul va folosi ca input un string în care se află parcurgerea preordine a arborelui, în ordinea, **rădăcină, stânga, dreapta**, ce poarta numele de Forma poloneza prefixată. Această expresie este citită și transformată în arbore de către functia `getAST()` din fișierul `ASTUtils.o`, funcție care este apelată și în schelet. De asemenea, de eliberarea memoriei utilizate pentru reținerea arborelui se ocupă funcția `freeAST()` din același fișier, care este de asemenea apelată în schelet.

Astfel, ce vă revine de facut este să implementați parcurgerea și evaluarea arborelui deja construit. Urmăriți comentariile din schelet pentru detalii.

De asemenea, structura folosită pentru a stoca un nod din arbore arată astfel:

```
struct __attribute__((__packed__)) Node
{
    char* data;
    struct Node* left;
    struct Node* right;
};
```

Prototipurile funțiilor definite în fișierul ASTUtils.o sunt:

```
struct Node* getAST();
void freeAST(struct Node* root);
```

Stringul `data` conține fie un *operator* (`+, -, *, /`), fie un *operand* (număr). În ambele cazuri, stringul se termină cu caracterul `\0`.

După cum puteti afla si de pe acest link, urmatorul cod:

```
__attribute__((__packed__)
```

îi interzice compilatorului să adauge padding in cadrul unei structuri, distanțele față de începutul structurii la care se află campurile acesteia fiind astfel cele asteptate și nevariind de la o masină la alta.

Găsiți în arhiva cu resursele temei un fișier schelet de la care puteți începe implementarea.

O explicație a evaluării expresiei găsiți aici.

Exemple de rulare

```
$ ./tema1
* -  5 6 7
-7
$ ./tema1
+ + * 5 3 2 * 2 3
23
$ ./tema1
- * 4 + 3 2 5
15
```

Testare

Tema se poate testa pe platforma vmchecker sau local folosind checkerul `checker.py` din arhiva cu resursele temei.

Arhiva conține o serie de inputuri în directorul `inputs/input*.txt` și rezultatele așteptate pentru fiecare test, în directorul `outputs`. Verificarea acestor teste este făcută automat de către checker (`checker.py`).

Pentru a putea realiza tema in SASM este necesar să realizati linkarea dintre fisierul obiect rezultat în urma asamblării temei voastre împreuna cu fișierul obiect ASTUtils.o. Puteți edita opțiunile linker-ului din Settings → Settings → Build → Linking options. Adăugati la finalul opțiunilor existente calea absolută catre fisierul obiect ASTUtils.o. Opțiunile ar trebui să arate:

```
$PROGRAM.OBJ$ $MACRO.OBJ$ -g -o $PROGRAM$ -m32 /cale/absoluta/ASTUtils.o
```

În cazul in care nu realizați acest pas veți primi erorile:

```
undefined reference to `getAST'
undefined reference to `freeAST'
```

Trimitere și notare

Temele vor trebui încărcate pe platforma vmchecker (în secțiunea IOCLA) și vor fi testate automat. Arhiva încărcată trebuie să fie o arhivă `.zip` care să conțină:

- fișierul sursă ce conține implementarea temei, denumit `tema1.asm`
- fișier `README` ce conține descrierea implementării

Punctajul final acordat pe o temă este compus din:

- punctajul obținut prin testarea automată de pe vmchecker - 90%
- fișier README - 10%

A fost facut un update al regulamentului de realizare a temelor - s-a introdus o secțiune pentru depunctări, vă rugăm să o parcurgeți. De asemenea daca nu ați parcurs regulamentul de realizare a temelor deja vă recomandăm sa o faceți.

Mașina virtuală folosită pentru testarea temelor de casă pe vmchecker este descrisă în secțiunea Mașini virtuale din pagina de resurse.

Precizări suplimentare

- Rezultatul trebuie afișat la `stdout`. Pentru aceasta, puteți folosi macroul `PRINT_DEC` din SASM.
- Aici puteţi găsi un cheatsheet, recomandări, o serie de buguri frecvente, etc.
- Arborele citit de la tastatură este valid (nu se efectuează împățiri la 0, nu se citesc caractere diferite de [0-9] si ”-+/* ”, etc)
- Se vor efectua maximum 200 de operații
- Eliberarea memoriei realizata de functia freeAST() trebuie sa se execute cu succes. Puteti altera arborele cât timp toata memoria alocată este și eliberată la sfârșitul execuției. În caz contrar implementarea va fi depunctată.
- Operanzii pot avea valori negative, va trebui sa folosiți imul pentru înmulțire și idiv și cdq pentru împarțire.
- Răspunsul oricărei operații nu va depăși 31 de biți (nu va seta flag-ul de Overflow)
- În cazul operației de împărțire, doar câtul va fi luat în considerare. Spre exemplu, pentru inputul `+ 2 / 7 6` programul va trebui să afișeze 3:

```
$ ./tema1
+ 2 / 7 6
3
```

- Pentru orice subarbore cu mai mult de un nod, rădăcina subarborelui este operatorul, iar fiii sunt operanzii.
- Ordinea efectuării operațiilor este **de la stânga la dreapta**.

```
$ ./tema1
- 2 1
1
```

- Observați că într-un arbore sintactic abstract prioritatea operatiilor matematice este dată exclusiv de poziția acestora în cadrul arborelui. Astfel, pentru inputul: `* + 2 1 + 3 4` se va afișa `21` si nu `9`, operațiile executându-se în ordinea `(2 + 1) * (3 + 4)`, și nu `2 + 1 * 3 + 4`.
- Dacă întâmpinați probleme în rularea modului de debug în SASM, asigurațivă că fișierul `~/.gdbinit` nu conține următoarea linie `source ~/peda/peda.py`.
- Parsarea stringurilor pentru a obține numere trebuie realizată în limbaj de asamblare, nu cu o funcție externă (cum ar fi atoi)

Resurse

Arhiva ce conține checkerul, testele și fișierul de la care puteți începe implementarea este aici.

Table of Contents