




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 primită sub forma unui AST (abstract syntax tree) în urma apelului unei funcții externe - `getAST()`. Citirea se va face de către funcția `getAST()` de la tastatură.

Arbore sintactic abstract

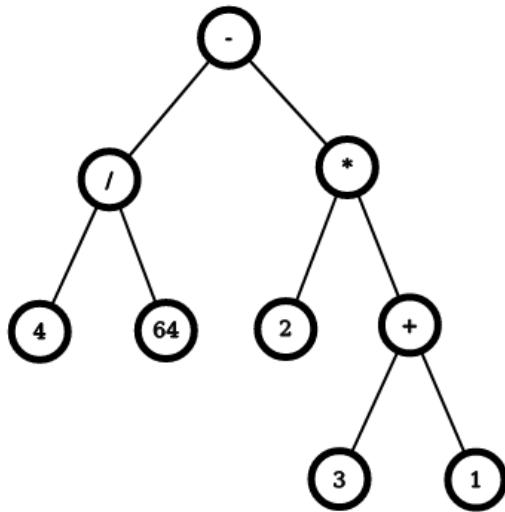
Arborii sintactici abstracți sunt o structură de date cu ajutorul căreia compilatoarele reprezintă structura unui program. În urma parcurgerii AST-ului, un compilator generează metadatele necesare transformării din cod de nivel înalt în cod assembly. Puteți găsi mai multe informații despre AST  aici.

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

- Anunțuri
- Bune practici
- Calendar
- Catalog
- Feed RSS
- IOCLA Need to Know
- Reguli și notare
- Resurse utile

Cursuri

- Curs 01: Introducere
- Curs 01 - 02: Arhitectura sistemelor de calcul
- Curs 02 - 03: Arhitectura x86
- Curs 04: Reprezentarea datelor în sistemele de calcul
- Curs 05: Reprezentarea datelor în sistemele de calcul - C2
- Curs 06 - 07: Setul de instrucțiuni
- Curs 07: Declararea datelor
- Curs 08 - 09: Moduri de adresare
- Curs 09: Stiva
- Curs 10 - 11: Funcții
- Curs 12: C + asm
- Curs 13: Unelte, utilitare



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 funcția `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 funcțiilor 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:

- Curs 13 - 15:
Buffer overflows, securitate
- Curs 16 - 17:
Optimizări
- Curs 18 - 19:
Virgulă flotantă


Laboratoare

- Laborator 01:
Introducere
- Laborator 02:
Toolchain
- Laborator 03:
First baby steps
- Laborator 04:
Rolul registrelor, adresare directă și bazată
- Laborator 05:
Structuri, vectori. Operatii pe șiruri
- Laborator 06:
Lucrul cu stiva
- Laborator 07:
Apeluri de funcții
- Laborator 08:
Interacțiunea C-assembly
- Laborator 09:
Analiza statică și dinamică a programelor. GDB
- Laborator 10:
Gestiunea bufferelor. Buffer overflow
- Laborator 11:
Optimizări
- Laborator 12:
Calcul în virgulă mobilă

Teme

- Tema 1 - Prefix AST
- Tema 2 - Stegano
- Tema 3 - Exploit ELF's, not elves

```
__attribute__((__packed__))
```

îi interzice compilatorului să adauge  padding în cadrul unei structuri, distanțele față de începutul structurii la care se află campurile acestea fiind astfel cele asteptate și nevariind de la o mașină 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 în SASM este necesar să realizați linkarea dintre fișierul obiect rezultat în urma asamblării temei voastre împreună cu fișierul obiect ASTUtils.o. Puteți edita opțiunile linker-ului din Settings → Settings → Build → Linking options. Adăugați la finalul opțiunilor existente calea absolută către fișierul obiect ASTUtils.o. Opțiunile ar trebui să arate:

```
$PROGRAM.OBJ$ $MACRO.OBJ$ -g -o $PROGRAM$
```


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

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

Table of Contents

- Tema 1 - Prefix AST
 - Enunț
 - Arbore sintactic abstract
 - Implementare
 - Exemple de rulare
 - Testare
 - Trimitere și notare
 - Precizări suplimentare
 - Resurse

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 făcut un update al regulamentului de realizare a temelor - s-a introdus o secțiune pentru **depunțări**, vă rugăm să o parcurgeți. De asemenea dacă nu ați parcurs **regulamentul de realizare a temelor** deja vă recomandăm să 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 puteți găsi un cheatsheet, recomandări, o serie de buguri frecvente, etc.
- Arborele citit de la tastatură este valid (nu se efectuează împărțiri la 0, nu se citesc caractere diferite de `[0-9]` și `"-+/*"`, etc)
- Se vor efectua maximum 200 de operații
- Eliberarea memoriei realizată de funcția `freeAST()` trebuie să se execute cu succes. Puteți altera arborele cât timp toată memoria alocată este și eliberată la sfârșitul execuției. În caz contrar implementarea va fi depunctată.
- Operanzii pot avea valori negative, va trebui să folosiți  `imul` pentru înmulțire și  `idiv` și  `cdq` pentru împărțire.
- 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 operațiilor matematice este dată exclusiv de poziția acestora în cadrul arborelui. Astfel, pentru inputul: `* + 2 1 + 3 4` se va afișa `21` și 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, asigurați-vă 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)

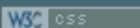




Resurse

Arhiva ce conține checkerul, testele și fișierul de la care puteți începe implementarea este [aici](#).

iocla/teme/tema-1.txt · Last modified: 2019/11/12 14:01 by radu.nicolau

 Old revisions

 Media Manager  Back to top

 BY-SA  CHIMERIC  DE  W3C CSS  DOKUWIKI  GET FIREFOX  RSS  XML FEED  W3C XHTML 1.0