Laborator 6 și 7

Prelucrarea matricilor

Pe parcursul laboratoarelor 6 și 7 vom realiza un analizor lexico-sintactic pentru a prelucra matrici și operații pe matrici. Așadar, vom crea o nouă pereche de fișiere lex și yacc.

Inputul poate fi o atribuire de matrice unei variabile sau o operație aritmetică între variabile de tip matrice. Exemple:

```
A = 1 0 2
    0 3 1 ;

B = 6 11 16
    0  1  2 ;

C = 0 1 0
    2 4 2 ;
```

```
A + B + C ;
> 7 12 18
  2  8  5

A + B – C ;
> 7 10 18
  -2  0  1
```

Scopul este să reținem matricile citite în variabile, pentru a putea aplica operații pe ele. Dacă variabilele sunt scrise în input cu majuscule, putem crea un vector de 26 de poziții în care să stocăm matricile atribuite variabilelor. Tradițional, în domeniul compilatoarelor, acest vector se numește tabelă de simboluri. Totuși, dintre simboluri (mai exact, dintre simbolurile terminale/tokeni), tabela de simboluri stochează doar identificatorii (variabile și nume de funcții) împreună cu informații legate de acestea: valoarea curentă păstrată într-o variabilă, respectiv un pointer spre începutul codului unei funcții.

```
matr *mem[26];
```

Pentru a stoca o matrice, putem crea două structuri: una pentru a stoca o înșiruire de numere întregi compunând un rând dintr-o matrice, iar alta pentru a stoca o înșiruire de rânduri care formează o matrice. Fiecare dintre cele două structuri trebuie să conțină și numărul total de elemente dintr-un rând (care practic definește numărul total de coloane din matrice), respectiv numărul total de rânduri dintr-o matrice.

```
typedef struct _line{
        int elems[MAX];
        int no_columns_used;
} line;

typedef struct _matr {
        line *rows[MAX];
        int no_rows_used;
} matr;
```

Având în vedere că pe stiva de valori pot să coexiste atât numere, cât și înșiruiri de numere sau înșiruiri de rânduri, înseamnă că yylval trebuie să fie declarat drept un %union cu trei câmpuri.

```
%union {
  struct _matr *mat;
  struct _line *lin;
  int ival;
}
```

Setul de producții este format dintr-un set de reguli care descriu inputul (*stmt*), un set de reguli care descriu operațiile aritmetice aplicate pe matrici (*expr*) și reguli pentru construcția rândurilor (*row*) și a matricilor (*matrix*).

```
stmt : VAR '=' matrix ';'
     | expr ';'
     ;
expr : expr '+' expr
     | expr '-' expr
     | VAR
     ;
matrix : matrix '\n' row
       | row
       ;
row : row NUMBER
    | NUMBER
    ;
```

Tokenul (literalul) ‘**;**’ indică finalul unei instrucțiuni (statement – stmt), după care ar urma un ‘**\n**’. Tokenul ‘**\n**’ apare și ca un separator între rândurile dintr-o matrice. După ultimul rând dintr-o matrice nu urmează direct ‘\n’, ci urmează tokenul ‘;’ și abia apoi ‘\n’ după finalul instrucțiunii.

Putem observa o comparație între gramatica matricilor și gramatica G1 de la curs:

```
E -> E + T
   | T
```

În gramatica G1, o expresie aritmetică E este definită recursiv ca o enumerare de termeni aritmetici T, separați prin tokenul ‘+’. Asemănător, în cazul matricilor, o matrice (*matrix*) este definită recursiv ca o enumerare de rânduri (*row*), separate între ele prin tokenul ‘\n’. Ca și aspect vizual, în G1 o expresie E este o sumă de termeni T (separați prin ‘+’) de la stânga la dreapta, iar o matrice (*matrix*) este o succesiune de rânduri (*row*) de sus în jos, efectul vertical fiind dat de ‘\n’-urile dintre rânduri.

În final, în secțiunea de user subroutines a fișierului **.y** vom defini funcțiile necesare pentru lucrul pe matrici.

Exerciții propuse:

1. Adăugați o funcție pentru calculul determinantului unei matrice.
2. Adăugați o funcție pentru realizarea înmulțirii a două matrici.