**Pismeni ispit iz JIP 02. 09. 2013.**

**Zadatak 1:**

a) Koji tipovi varijabli se mogu koristiti u jeziku Scheme?
b) Napišite funkciju u jeziku Scheme kojom se iz poznate liste L1 stvara nova lista L2 koja sadrži članove liste uvećane za 1.
c) Koristeći funkciju **filter**, napišite Scheme izraz (ili funkciju) kojim se formira lista od elemenata liste **L** koji su veći od 10.

**Zadatak 2.** Napišite prijevod u asemblerski jezik sljedećeg C programa:

```
int GetSmaller (int a, int b, int c)
/* funkcija vraća vrijednost manjeg od tri argumenta */
{
      if (a < b) {
            if (a < c ) return a;
      }
      else {
            if (b < c) return b;
      }
      return c;
}

void main ()
{
      int x = 5, y = 4, z = 6;
      printf ("Manji je %d", GetSmaller(x, y, z));
}
```

**Zadatak 3:** Napišite specifikaciju LEX programa za slučaj da funkcija yylex() treba prepoznati sljedeće tokene:

```
FLOAT – realni broj koji može biti u običnom i eksponencijalnom formatu
INT – integer u decimalnom (123) ili heksadecimalnom formatu (0x123)
ID – identifikator varijable – niz znakova (prvi je slovo, a ostali mogu biti
slovo ili broj)
PLUS - '+'  (operator zbrajanja)
MINUS – '-' (operator oduzimanja)
MUL – '*'   (operator množenja)
DIV – '/'   (operator dijeljenja)
EXP – '^'   (operator potenciranja)
NL – '\n'   (nova linija)
LEFT - '('  (lijeva zagrada)
RIGHT - ')' (dasna zagrada)
PRINT - ?
EQU – '='
```

Kostur specifikacije je datoteka "spec.l"

```
{%
enum tokendef {FLOAT=255, INT, ID, PRINT, EQU, PLUS, MINUS, MUL, DIV, EXP, NL,
LEFT, RIGHT};
%}
%%   specifikacija tokena
'\n' return NL;
......ovdje dopunite..........
%%
```

**Zadatak 4.** Napišite rekurzivni parser (bez semantičkih akcija) prema sljedećoj EBNF definiciji gramatike

```
Naredba : (ID EQU Izraz | PRINT Izraz) NL;

Izraz   : Clan ((PLUS | MINUS) Clan)*;

Clan    : Faktor ((MUL | DIV ) Faktor)*;

Faktor  : INT
        | FLOAT
        | ID
        | LEFT Izraz RIGHT
        ;
```

Ova gramatika dozvoljava zapis naredbi oblika:

```
y = x * (7 + y * 7.1e-3)
?y
```

Prvo je naredba pridjele vrijednosti, a drugo je naredba da se ispiše vrijednost izraza iza znaka '?'.
Unos završava znakom nove linije (NL token). U izrazima se mogu pojavljivati realni i cijeli brojevi, identifikatori, operatori (+, -, *, /, ^) i separatori (zagrade, prazna mjesta i tabulatori).

Pretpostavite da vam je na raspolaganju leksički analizator, iz prethodnog zadatka, u obliku funkcije **`int yylex()`**, koja vraća token iz ulaznog niza, a njegov leksem sprema u globalnu varijablu **`char yytext[]`**), kako je zadano specifikacijom u prethodnom zadatku.

Funkcije i konstante rekurzivnog parsera imaju prototip:

```
enum tokendef {FLOAT=255, INT, ID, PRINT, EQU, PLUS, MINUS, MUL, DIV, EXP, NL,
LEFT, RIGHT};

void Naredba();         // ispisuje vrijednost
void Izraz();           // vraća vrijednost izraza
void Clan();            // vraća vrijednost člana
void Faktor();          // vraća vrijednost faktora

extern int yylex();     // funkcije za dobavu tokena
extern char yytext[];   // i pripadni string leksema
```

## Rješenja

### Zadatak 1.

```
a) Brojevi, stringovi, boolean, znakovi, liste, vektori, strukture…
b) (define dodaj (lambda
                        (lista)
                              (map
                                    (lambda
                                          (li)
                                          (+ 1 li)
                                    )
                                    lista
                              )
                  )
   )
a)  (filter (lambda (li) (> li 10)) L)
```

### Zadatak 2.

```
#include "asmc.c"

#define  a  DWORD(M_[esp+4])
#define  b  DWORD(M_[esp+8])
#define  c  DWORD(M_[esp+12])

PROC(GetSmaller)
       MOV(eax, a)               //prebaci a u registar eax

       CMP(eax, b)               //usporedi a i b
       JL(a_je_manji)            //ako je a < b skoci na labelu a_je_manji
       MOV(eax, b)               //u protivnom, upiši b u eax

       CMP(eax, c)               //usporedi b i c (b je manji od a)
       JL(kraj)                  //ako je b y c skoči na kraj jer je b najmanji
       MOV (eax, c)              //u protivnom, prebaci c u eax
       JMP(kraj)                 //skoči na kraj (c je najmanji)

a_je_manji:
       CMP(eax, c)               //usporedi a i c
       JL(kraj)                  //ako je  a < c skoči na kraj (a je najmanji)
       MOV (eax, c)              //u protivnom, prebaci c u eax (c je najmnaji)

kraj:
       RET(0)
ENDP

#define        x  DWORD(M_[ebp-4])
#define        y  DWORD(M_[ebp-8])
#define        z  DWORD(M_[ebp-12])

PROC(MAIN)
       PUSH(ebp)
       MOV(ebp, esp)

       SUB(esp, 12)              //alokacija na stogu za 2 lokalne varijable
       MOV(x, 5)                 //x = 5
       MOV(y, 4)                 //y = 4
       MOV(z, 6)                 //z = 6

       PUSH(z)                   //stavljanje parametara funkcije GetSmaller na stog
       PUSH(y)
       PUSH(x)
```

```
	CALL(GetSmaller)		//poziv procedure
	ADD(esp, 12)			//očisti stog

	PUTS("Manji je ")		//ispis teksta
	PUTI(eax)			//ispis sadržaja eax (povratna vriednost funkcije)

	MOV(esp, ebp)
	POP(ebp)

	RET(0)
ENDP
```

## Zadatak 3.

```
%{
	#include <stdio.h>
	enum tokendef {FLOAT=255, INT, ID, PRINT, EQU, PLUS, MINUS, MUL, DIV, EXP, NL, LEFT, RIGHT};

%}

DIGIT		[0-9]
DOT		\.
DECIMAL		{DOT}{DIGIT}+
EXP		[eE][-+]?{DIGIT}*{DECIMAL}?
LETTER		[a-zA-Z]

%%

[ \t]+		;
({DIGIT}+{DOT}?)|({DIGIT}*{DECIMAL}){EXP}?  return FLOAT;
{DIGIT}+|('0x'{DIGIT}+)		return INT;
{LETTER}({DIGIT}|{LETTER})+	return ID;
'+'				return PLUS;
'-'				return MINUS;
'*'				return MUL;
'/'				return DIV;
'^'				return EXP;
'('				return LEFT;
')'				return RIGHT;
'\n'				return NL;
'='				return EQU;
'?'				Return PRINT;
.				;

%%

int yywrap()
{
	return 1;
}
```

## Zadatak 4.

```
#include <stdio.h>
#include <stdlib.h>
enum tokendef {FLOAT = 255, INT, ID, PRINT, EQU, PLUS, MINUS, MUL, DIV, EXP, NL, LEFT, RIGHT};

extern int yylex();
extern char yytext[];

typedef int Token;
int lookaheadToken;

void Error(char* str)
{
	printf("%s", str);
	exit(1);
}

void match(Token t)
{
	if(lookaheadToken == t)
		lookaheadToken = yylex();
	else
		Error("Krivi token");
}

void Naredba();
void Izraz();
void Clan();
void Faktor();

void Naredba()
{
	if (lookaheadToken == ID)
	{
		match(ID);
		match(EQU);
		Izraz();
	}

	else if (lookaheadToken == PRINT)
	{
		match(PRINT);
		Izraz();
	}

	else
		Error("Krivi token na pocetku linije!");


	if (lookaheadToken == NL)
		match(NL);

	else
		Error("Krivi token. Ocekivan kraj linije!");
}

void Izraz()
{
	Clan();

	while (lookaheadToken == PLUS || lookaheadToken == MINUS)
	{
		if (lookaheadToken == PLUS)
			match(PLUS);
```

```
		else
			match(MINUS);

		Clan();
	}
}

void Clan()
{
	Faktor();

	while (lookaheadToken == MUL || lookaheadToken == DIV)
	{
		if (lookaheadToken == MUL)
			match(MUL);

		else
			match(DIV);

		Faktor();
	}
}

void Faktor()
{
	if (lookaheadToken == INT)
		match(INT);

	else if (lookaheadToken == FLOAT)
		match(FLOAT);

	else if (lookaheadToken == ID)
		match(ID);

	else if (lookaheadToken == LEFT)
	{
		match(LEFT);
		Izraz();
		match(RIGHT);
	}

	else
		Error("Krivi token!");
}


int main()
{
	lookaheadToken = yylex();
	Naredba();

	return 0;
}
```