Pismeni ispit iz JIP 15. 7. 2013.

Zadatak 1. U Scheme jeziku:

a) Definirajte listu imena L koja sadrži elemente: 10, 20, 30, 40, 50.

b) Napišite izraz kojim se dobije lista koja ne sadrži prvi, drugi i treći element prethodne liste L.

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)
/* funkcija vraća vrijednost manjeg od dva argumenta */
{
      if (a < b) return a;
      else return b;
}

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

Zadatak 3: Napišite specifikaciju leksičkog analizatora za automatsko generiranje funkcije int yylex() pomoću programa LEX, a koja prepoznaje slijedeće tokene i njima pripadne lekseme:

```
NUM – realni broj koji može biti u običnom i eksponencijalnom formatu
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)
```

Kostur specifikacije je datoteka "spec.l"

```
{%
enum tokendef {NUM=255, PLUS, MINUS, MUL, DIV, EXP, NL, LEFT, RIGHT};
%}
%%    specifikacija tokena
'\n'  return NL;
.......
%%
```

Zadatak 4. Napišite rekurzivni parser prema sljedećoj EBNF formi gramatike

```
Naredba : Izraz NL;

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

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

ExpFaktor: Faktor ( EXP Faktor)* ;

Faktor  : NUM_TOKEN
        | (' Izraz ')'
        ;
```

Ova gramatika dozvoljava zapis naredbi oblika:

7 * (7.8+6*7e-3)^2

Unos naredbe počinje matematičkim izrazom, a završava znakom nove linije (NL token). Nakon toge ispisuje se rezultat izraza. U izrazima se mogu pojavljivati realni brojevi, 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 {NUM=255, PLUS, MINUS, MUL, DIV, EXP, NL, LEFT, RIGHT};

void Naredba();         // ispisuje vrijednost
double Izraz();         // vraća vrijednost izraza
double Clan();          // vraća vrijednost člana
double ExpFaktor();     // vraća vrijednost faktora i potencije
double Faktor();        // vraća vrijednost faktora

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

U svakoj funkciji odredite i semantičke akcije koje rezultiraju vrijednošću koje funkcije vraćaju ili koja se ispisuje u funkciji Naredba().

Rješenja

Zadatak 1.

```
a) (define L '(10 20 30 40 50))
b) (cdr (cdr (cdr L)))
c) (filter (lambda (li) (> li 10)) L)
```

Zadatak 2.

```
#include "asmc.c"

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

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 kraj
	MOV(eax, b)			//u protivnom, upiši b u eax

kraj:
	RET(0)
ENDP


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

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

	SUB(esp, 8)			//alokacija stoga za 2 lokalne varijable
	MOV(x, 5)			//x = 5
	MOV(y, 4)			//y = 4

	MOV(eax, y)			//ubacivanje parametara funkcije GetSmaller na stog
	PUSH(eax)			//Ovo radimo pomoću eax, iako smo mogli i bez njega
	MOV(eax, x)
	PUSH(eax)

	CALL(GetSmaller)		//poziv procedure
	ADD(esp, 8)			//očisti stog

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

	MOV(esp, ebp)
	POP(ebp)

	RET(0)
ENDP
```

Zadatak 3.

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

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

%%

[ \t]+		;
({DIGIT}+{DOT}?)|({DIGIT}*{DECIMAL}){EXP}?  return NUM;
\+		return PLUS;
\-		return MINUS;
\*		return MUL;
\/		return DIV;
\^		return EXP;
\(		return LEFT;
\)		return RIGHT;
'\n'		return NL;    //NAPOMENA: može i bez navodnika... neznam zašto ih je stavia
.		;

%%

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

Zadatak 4.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

enum tokendef {NUM=255, 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();
double Izraz();
double Clan();
double ExpFaktor();
```

```
double Faktor();


void Naredba()
{
	double val = 0;

	val = Izraz();

	if (lookaheadToken == NL)
		match(NL);
	else
		Error("Ocekivan kraj linije, a nije se dogodio");

	printf("Rezultat je %f\n", val);
}

double Izraz()
{
	double val = 0;
	val = Clan();

	while (lookaheadToken == PLUS || lookaheadToken == MINUS)
	{
		if (lookaheadToken == PLUS)
		{
			match(PLUS);
			val += Clan();
		}

		if (lookaheadToken == MINUS)
		{
			match(MINUS);
			val -= Clan();
		}
	}

	return val;
}

double Clan()
{
	double val = 0;

	val = ExpFaktor();

	while (lookaheadToken == MUL || lookaheadToken == DIV)
	{
		if (lookaheadToken == MUL)
		{
			match(MUL);
			val *= ExpFaktor();
		}

		if (lookaheadToken == DIV)
		{
			double val2;

			match(DIV);
			val2 = ExpFaktor();

			if (val2 != 0)
				val /= val2;
			else
				Error ("Ne smijes dijeliti s nulom");
		}
	}

	return val;
}
```

```
double ExpFaktor()
{
	double val;

	val = Faktor();

	while (lookaheadToken == EXP)
	{
		double val2 = 0;

		match (EXP);
		val2 = Faktor();

		val = pow (val, val2);
	}

	return val;
}

double Faktor()
{
	double val;

	if (lookaheadToken == NUM)
	{
		val = atof(yytext);
		match (NUM);
	}

	else if (lookaheadToken == LEFT)
	{
		match(LEFT);
		val = Izraz();
		match(RIGHT);
	}
	else
		Error("Nedostaje faktor!");

	return val;
}

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

	return 0;
}
```

Malo vježbe - LEX i YACC za 3. i 4. zadatak sa ovog roka!

LEX (Potrebne su manje promjene jer se LEX sada kombinira sa YACC-om)

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

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

%%

[ \t]+		;
({DIGIT}+{DOT}?)|({DIGIT}*{DECIMAL}){EXP}? { yylval.broj = atof(yytext); return NUM; }
\+		return PLUS;
\-		return MINUS;
\*		return MUL;
\/		return DIV;
\(		return LEFT;
\^		return EXP;
\)		return RIGHT;
'\n'		return NL;
.		;

%%

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

Prije YACC-a treba pretvorit gramatiku iz 4. zadatka iz EBNF u BNF oblik. Treba promijenit izraze:

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

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

ExpFaktor: Faktor ( EXP Faktor)* ;
```

Prema relaciji iz skripte (lekcija 2, strana 16, naslov **Lijeva rekurzija**) ovo se lako pretvara u:

```
Izraz   : Clan | Izraz PLUS Clan | Izraz MINUS Clan;

Clan    : ExpFaktor | Clan MUL ExpFaktor | Clan DIV ExpFaktor;

ExpFaktor:  Faktor | ExpFaktor EXP Faktor;
```

YACC

```
%{
	#include <stdio.h>
	#include <math.h>
	void yyerror(char *s);
	double val = 0;	// Za konačnu vrijednost
%}

%union
{
	double broj;
}

%token <broj> NUM_TOKEN		// Napomena: U 3. zadatku se zove NUM, a u 4. NUM_TOKEN
%token PLUS MINUS MUL DIV NL EXP // Nema tokena za lijevu i desnu zagradu jer u 4. zadatku nisu zadani

%%
Naredba		:	Izraz NL		{$$ = $1; val = $$;} // Konačnu vrijednost pridruži u val
		;
Izraz		:	Clan			{$$ = $1;}
		|	Izraz PLUS Clan		{$$ = $1 + $3;}
		|	Izraz MINUS Clan	{$$ = $1 - $3;}
		;
Clan		:	ExpFaktor		{$$ = $1;}
		|	Clan MUL ExpFaktor	{$$ = $1 * $3;}
		|	Clan DIV ExpFaktor	{$$ = $1 / $3;}
		;
ExpFaktor	:	Faktor			{$$ = $1;}
		|	ExpFaktor EXP Faktor	{$$ = pow ($1, $3);}
		;
Faktor		:	NUM_TOKEN		{$$ = $1;}	// $$ postaje yylval iz LEX-a
		|	'(' Izraz ')'		{$$ = $2;}
		;
%%

int main()
{
	yyparse();
	printf („Vrijednost je %f", val);
}

void yyerror(char *s)
{
	printf("%s", s);
}
```