Pismeni ispit iz JIP 1. 07. 2013

Zadatak 1.

a) Napišite rezultat sljedećeg izraza u Sheme jeziku:

```
> ((lambda (x) (* x x x) ) 5)
> …
```

b) Napišite funkciju u jeziku Scheme imena **Kub** koja vraća vrijednost argumenta na potenciju 3, s tim da argument može biti jedan broj ili lista brojeva, (za obraditi listu koristite funkciju **map**)

Zadatak 2. Napišite prijevod u ASMC asemblerski jezik C funkcije Kubiraj(), koja svaki element niza A, od N elemenata, potencira kubno. Primijenite tu funkciju u main () funkciji na globalni niz Y. Za lokalne varijable n i x možete koristite registre procesora.

```
void Kubiraj (int A[], int N)
{
      register n, x;

      for (n = 0; n < N; n++) {
            x = A [n];
            A[n] = x * x * x;
      }
}

int Y[4] = {7, 21, 22, 7};

int main ()
{
      Kubiraj(Y, 4);
      printf("Treći element je: %d", Y[2]);
      return 0;
}
```

Zadatak 3. Napišite leksički analizator u obliku funkcije int GetToken() koja prepoznaje slijedeće tokene i pripadne lekseme:

```
NUM - realni broj koji može biti u običnom i eksponencijalnom formatu
PLUS - '+'
MINUS – '-'
MUL – '*'
DIV – '/'
NL – '\n' (nova linija)
LEFT - '(' (lijeva zagrada)
RIGHT - ')' (dasna zagrada)
```

Kada funkcija vrati prepoznati token, njegov leksem treba biti spremljen u globalnom stringu char lexeme[32];

Zadatak 4. Napišite rekurzivni parser prema sljedećoj EBNF formi gramatike

```
Naredba     : Izraz NL;

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

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

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

Ova gramatika dozvoljava zapis naredbi oblika: 7 * (7.8+6*7e-3) . 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 u obliku funkcije int GetToken(), koja vraća token iz ulaznog niza, a njegov leksem sprema u globalnu varijablu char lexeme[]), kao je zadano u prethodnom zadatku.

Funkcije rekurzivnog parsera imaju prototip:

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

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

Rješenja

Zadatak 1.

```
a) 125
b) (define Kub lambda (L)
       (if (atom? L) (* L L L)
            (map
                  (lambda (li) (* li li li) )
            L)
        )
      )
```

Zadatak 2.

```
#include "asmc.c"

VAR str DB(19) = "Treci element je: ";
VAR Y DD(4) = {D_(7), D_(21), D_(22), D_(7)};

#define A DWORD(M_[esp+4])                //koristimo esp jer nema
#define N DWORD(M_[esp+8])                //lokalnih varijabli

PROC(Kubiraj)
       MOV(edi,0)                         //edi je registar n (n = 0)
       MOV(ebx, A)                        //adresa niza A u ebx

loop:                                     //petlja
       CMP(edi, N)                        //usporedi n i N
       JGE(endloop)                       //ako je n >= N skoči na kraj

       MOV(eax, DWORD(M_[ebx + edi*4]))   //prebaci sadržaj A[n] u eax
       MUL(eax, eax)                      //x*x
       MUL(eax, DWORD(M_[ebx + edi*4]))   //x*x*x
       MOV(DWORD(M_[ebx + edi*4]), eax)   //vrati rezultat natrag u A[n]

       INC(edi)                           //uvećaj n
       JMP(loop)                          //skoči na petlju

endloop:
       RET(0)
ENDP

PROC(MAIN)
       PUSH(4)                            //stavi broj elemenata na stog
       LEA(eax,Y)                         //učitaj adresu niza u eax
       PUSH(eax)                          //stavi adresu niza iz eax na stog

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

       PUTS(str)                          //ispis teksta
       PUTI(DWORD(Y[8]))                  //ispis Y[2].

       RET(0)
ENDP
```

Zadatak 3.

```
#include <stdio.h>
#include <ctype.h>

#define NL     100
#define END    101
#define NUM    102
#define PLUS   103
#define MINUS  104
#define MUL    105
#define DIV    106
#define LEFT   107
#define RIGHT  108
#define BAD    -1

char lexeme[32];
int GetToken(void)
{
	int ch;

	while(1)
	{
		ch = getchar();

		if (ch == ' ' || ch == '\t')
			continue;
		if (ch == '\n')
			return NL;
		if (ch == EOF)
			return END;

		//Pitaj radi li se o broju (počima sa znamenkom, točkom ili eksponentom)
		if (isdigit(ch) || ch == '.' || ch == 'e' || ch == 'E')
		{
			lexeme[0] = (char)ch;
			lexeme[1] = '\0';
			int i = 1; //indeks za spremanje članova niza

			//Pitaj počima li broj sa znamenkom
			if (isdigit(ch))
			{
				ch = getchar();

				while (isdigit(ch) && i < 31)
				{
					lexeme[i++] = (char)ch;
					ch = getchar();
				}
			}

			//Pitaj nastavlja li se broj sa točkom ili je ona početak broja (svejedno!)
			if (ch == '.' && i < 31)
			{
				if (i != 1) //ako broj počima sa točkom ne treba je ponovno spremat u niz
jer je već spremljena u lexeme[0]. A ako nije treba je spremit
					lexeme[i++] = (char)ch;

				ch = getchar();

				while (isdigit(ch) && i < 31)
				{
					lexeme[i++] = (char)ch;
					ch = getchar();
				}
			}

			//Isto kao točka, samo opcionalni eksponent
			if ( (ch == 'e' || ch == 'E') && i < 31)
```

```
		{
			if (i != 1) //Ista napomena kao i za točku
				lexeme[i++] = (char)ch;

			ch = getchar();

			if ((ch == '-' || ch == '+') && i < 31)
			{
				lexeme[i++] = (char)ch;
				ch = getchar();
			}

			while (isdigit(ch) && i < 31)
			{
				lexeme[i++] = (char)ch;
				ch = getchar();
			}

			if (ch == '.' && i < 31)
			{
				lexeme[i++] = (char)ch;
				ch = getchar();

				while (isdigit(ch) && i < 31)
				{
					lexeme[i++] = (char)ch;
					ch = getchar();
				}
			}
		}

		lexeme[i] = '\0';

		if (ch != EOF)
			ungetc(ch,stdin);

		return NUM;
	}

	else if (ch == '/')
		return DIV;
	else if (ch == '*')
		return MUL;
	else if (ch == '+')
		return PLUS;
	else if (ch == '-')
		return MINUS;
	else if (ch == '(')
		return LEFT;
	else if (ch == ')')
		return RIGHT;
	else
		return BAD;
    }
}
```

Zadatak 4.

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

#define NL     100
#define END    101
#define NUM    102
#define PLUS   103
#define MINUS  104
#define MUL    105
#define DIV    106
#define LEFT   107
#define RIGHT  108
#define BAD    -1

char lexeme[32];
typedef int Token;
int lookaheadToken;

int GetToken(void)
{ Ovdje ide funkcija iz 3. zadatka… }

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

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

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

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

		if (lookaheadToken == DIV)
		{
			double val2;

			match(DIV);
			val2 = Faktor();

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

	return val;
}

double Faktor()
{
	double val;

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

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

	return val;
}

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

	return 0;
}
```

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

LEX

```
%{
	#include <stdio.h>
%}

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

YACC

```
%{
	#include <stdio.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	// 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	{$$ = $1 - $3;}
		;
Clan		:	Faktor			{$$ = $1;}
		|	Clan MUL Faktor		{$$ = $1 * $3;}
		|	Clan DIV Faktor		{$$ = $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);
}
```