Podstawy Kompilatorów

Laboratorium 13

Synteza kodu i środowisko czasu wykonania.

Zadanie 1:
W pliku wejściowym dany jest ciąg wywołań funkcji o postaci: nazwa funkcji, lewy nawias okrągły, (być może pusta) lista argumentów, prawy nawias i średnik. Lista argumentów składa się ze standardowych wyrażeń arytmetycznych (operatory '+', '-', '*' oraz liczby) rozdzielonych przecinkami. Proszę napisać generator kodu wynikowego tłumaczący wywołania funkcji na asembler X86 w następujący sposób:

- trzeba dokonać ewaluacji każdego wyrażenia, a obliczoną wartość położyć na stos instrukcją *push*
  zakładamy, że argumenty odkładane są na stos od lewej do prawej
- wywołać funkcję instrukcją *call*
- wyczyścić stos po wywołaniu funkcji wykonując rozkaz *add sp,larg*, gdzie *larg* jest liczbą argumentów wywołania funkcji
  jeżeli lista argumentów jest pusta należy pominąć rozkaz czyszczenia stosu.

Konstruując analizator składniowy nie wolno posługiwać się zmiennymi globalnymi. Pamięć alokowaną dynamicznie należy zwalniać od razu, gdy nie jest już potrzebna.

Dla pliku o postaci:

```
fun1(3+4,5);
fun2(4*5+2,9,6);
fun3();
```

powinniśmy otrzymać wynik:

```
push 7
push 5
call fun1
add sp,2

push 22
push 9
push 6
call fun2
add sp,3

call fun3
```

Analizator leksykalny ma następującą postać:

```
%{
  int yywrap(void);
  int yylex(void);
  #include <stdio.h>
  #include "y.tab.h"
%}
%%
\;                      { return(';'); }
\+                      { return('+'); }
\*                      { return('*'); }
\-                      { return('-'); }
\(                      { return('('); }
\)                      { return(')'); }
\,                      { return(','); }
[0-9]+                  { yylval.ival = atoi(yytext);
                          return(num);
                        }
[A-Za-z_][A-Za-z0-9_]*  { yylval.text = strdup(yytext);
                          return(id);
                        }
" "|\t|\n               { ; }
%%
int yywrap(void) { return 1;}
```

Zadanie 2:
Proszę napisać generator kodu trójadresowego dla języka źródłowego zawierającego niepusty ciąg instrukcji przypisania o postaci:

identyfikator := wyrażenie ;

gdzie wyrażenie ma standardową postać złożoną z liczb, identyfikatorów, binarnych operatorów +, -, *, /, unarnego operatora -, wyrażenie może być ujęte w nawiasy okrągłe.

Konstruując analizator składniowy nie wolno posługiwać się zmiennymi globalnymi. Pamięć alokowaną dynamicznie należy zwalniać od razu, gdy nie jest już potrzebna.

Dla pliku o postaci:

```
k := 3 + num ;
min := - (4 * 2) / 4 - 2;
```

powinniśmy otrzymać wynik:

```
t0 := 3
t1 := t0 + num
k := t1
t2 := 4
t3 := 2
t4 := t2 * t3
t5 := -t4
t6 := 4
t7 := t5 / t6
t8 := 2
t9 := t7 - t8
min := t9
```

Analizator leksykalny ma następującą postać:

```
%{
  int yywrap(void);
  int yylex(void);
  #include <stdio.h>
  #include "y.tab.h"
%}
%%
\;                      { return(';'); }
\+                      { return('+'); }
\*                      { return('*'); }
\-                      { return('-'); }
\/                      { return('/'); }
\(                      { return('('); }
\)                      { return(')'); }
":="                    { return OP_ASSIGN; }
[0-9]+                  { yylval.ival=atoi(yytext);
                          return(NUM);
                        }
[A-Za-z_][A-Za-z0-9_]*  { yylval.text=strdup(yytext);
                          return(ID);
                        }
" "|\t|\n               { ; }
%%
int yywrap(void) { return 1;}
```

Odpowiedzi do zadań

Zadanie 1:

```
%union
{
  char *text;
  int ival;
}

%{
  int yylex(void);
  void yyerror(const char *,...);
  int yyparse(void);
  #include <stdio.h>
  #include <stdlib.h>
  extern int yylineno;
%}
%left '+' '-'
%left '*'

%token <text> id
%token <ival> num

%type <ival> E LA ARGS
%%
P : F
  | P F
  ;
F : id '(' ARGS ')' ';'  { printf("\ncall %s",$1);
                           free($1);
                           if($3 > 0)printf("\nadd sp,%d\n",$3);
                         }
  ;
ARGS :     { $$ = 0; }
     | LA  { $$ = $1; }
     ;
LA : E         { $$ = 1;           printf("\npush %d",$1); }
   | LA ',' E  { $$ = $1 + 1; printf("\npush %d",$3); }
   ;
E : E '+' E   { $$ = $1 + $3; }
  | E '-' E   { $$ = $1 - $3; }
  | E '*' E   { $$ = $1 * $3; }
  | num       { $$ = $1; }
  ;
%%
void yyerror(const char *fmt,...) {
   printf("%s in line %d\n", fmt, yylineno); }
int main() { return yyparse(); }
```

Zadanie 2:

```
%union
{
  char *text;
  int ival;
}
%{
  int yylex(void);
  void yyerror(const char *,...);
  int yyparse(void);
  #include <stdio.h>
  #include <stdlib.h>
  #include <string.h>
  extern int yylineno;
  char *newTemp(void);
%}
%left '+' '-'
%left '*' '/'
%left UMINUS
%token <text> ID
%token OP_ASSIGN NUM
%type <text> E
%type <ival> NUM
%%
P : P S
  | S
  ;
S : ID OP_ASSIGN E ';'  { printf("%s := %s\n",$1,$3);
                          free($1); free($3);
                        }
  ;
E : E '+' E             { printf("%s := %s + %s\n",$$=newTemp(),$1,$3);
                          free($1); free($3);
                        }
  | E '-' E             { printf("%s := %s - %s\n",$$=newTemp(),$1,$3);
                          free($1); free($3);
                        }
  | '-' E  %prec UMINUS { printf("%s := -%s\n",$$=newTemp(),$2); }
  | E '*' E             { printf("%s := %s * %s\n",$$=newTemp(),$1,$3);
                          free($1); free($3);
                        }
  | E '/' E             { printf("%s := %s / %s\n",$$=newTemp(),$1,$3);
                          free($1); free($3);
                        }
  | '(' E ')'           { $$ = $2; }
  | NUM                 { printf("%s := %d\n",$$=newTemp(),$1); }
  | ID                  { $$ = $1; }
  ;
%%
void yyerror(const char *fmt,...) {
   printf("%s in line %d\n", fmt, yylineno); }
int main() { return yyparse(); }
char *newTemp(void)
{
  static int tempNo = 0;
  char buf[20];
  sprintf(buf,"t%d",tempNo++);
  return strdup(buf);
}
```