# Podstawy Kompilatorów

Laboratorium 10

**Translacja sterowana składnią w generatorze YACC.**

**Zadanie 1:**
Proszę napisać program, który dla danej liczby całkowitej *j* oraz niepustego ciągu liczb naturalnych $c_0, c_1, \ldots, c_j, \ldots, c_n$ podaje wartość elementu $c_j$. Jeśli j>n, to nie jest wyświetlana żadna liczba.
*Wejście:* plik składa się z wiersza, w którym najpierw podawana jest wartość j, a po niej niepusty ciąg liczb poprzedzony znakiem ':'. Elementy ciągu są rozdzielone przecinkami.
*Wyjście:* jeśli $j \leq n$, to na wyjściu otrzymujemy liczbę $c_j$ ujętą w nawiasy prostokątne. W przeciwnym wypadku pojawiają się same nawiasy prostokątne.

Konstruując analizator składniowy nie wolno posługiwać się zmiennymi globalnymi.
Proszę skonstruować parser w wersji:
- z atrybutem dziedziczonym
- S-atrybutowej

*Przykłady:*
Dla pliku o postaci:
```
2 : 5, 7, 9, 1
```
powinniśmy otrzymać wynik:
```
[9]
```

Dla pliku o postaci:
```
4 : 5, 7, 9, 11
```
powinniśmy otrzymać wynik:
```
[]
```

Analizator leksykalny ma następującą postać:

```
%{
  int yywrap(void);
  int yylex(void);
  #include <stdio.h>
  #include "y.tab.h"
%}
%%
\,          { return(','); }
\:          { return(':'); }
[0-9]+      { yylval = atoi(yytext);
              return(num);
            }
%%
int yywrap(void){  return 1;}
```

**Zadanie 2:**
Poprawny plik wejściowy zbudowany jest ze znaków * (gwiazdka), które tworzą w pliku kształt odwróconego trójkąta prostokątnego (wnętrze trójkąta też jest wypełnione znakami *). Oznacza to, że ostatnia linia w takim pliku zawiera dokładnie jeden znak *, przedostatnia linia zawiera dwa znaki *, trzecia linia od końca trzy znaki *, itd. aż do pierwszej linii w tym pliku. Reasumując, każda linia w tym pliku (za wyjątkiem ostatniej) ma zawsze o jeden znak * mniej niż linia, która występuje bezpośrednio po niej. Poprawnie zbudowany plik wejściowy może więc wyglądać następująco:

```
*****
****
***
**
*
```

Minimalny rozpatrywany trójkąt prostokątny składa się z dokładnie 1 linii i ma następującą postać:

```
*
```

Proszę napisać analizator, który będzie sprawdzał, czy plik wejściowy zbudowany jest zgodnie z powyższymi założeniami.

Konstruując analizator składniowy nie wolno posługiwać się zmiennymi globalnymi.

*Uwaga*: każda linia pliku zakończona jest znakiem końca wiersza.

Analizator leksykalny ma następującą postać:

```
%{
  int yywrap(void);
  int yylex(void);
  #include <stdio.h>
  #include "y.tab.h"
%}
%%
\*         { return('*'); }
\n         { return('\n'); }
%%
int yywrap(void){  return 1;}
```

**Zadanie 3.**
Na wejściu dana jest deklaracja niepustej listy identyfikatorów zgodna ze składnią języka Pascal. Załóżmy, że rozpatrujemy tylko zmienne typu całkowitego (*integer*) oraz znakowego (*char*). Taką listę należy rozbić na pojedyncze deklaracje.

Dla wejścia o postaci:
```
var a, b, c : InTeGeR;
```
powinniśmy otrzymać odpowiedź:
```
var a : integer;
var b : integer;
var c : integer;
```

Analizator leksykalny ma następującą postać:

```
%{
  int yywrap(void);
  int yylex(void);
  #include <stdio.h>
  #include "y.tab.h"
%}
%%
[vV][aA][rR]                    { return KWD_VAR; }
[iI][nN][tT][eE][gG][eE][rR]    { return KWD_INTEGER; }
[cC][hH][aA][rR]                { return KWD_CHAR; }
[a-zA-Z][a-zA-Z0-9]*            { yylval.text = strdup(yytext);
                                  return ID;
                                }
";"                             { return ';'; }
","                             { return ','; }
":"                             { return ':'; }
%%
int yywrap(void) { return 1;}
```

## Odpowiedzi do zadań

**Zadanie 1:**
wersja z atrybutem dziedziczonym:

```
%{
  int yylex(void);
  void yyerror(const char *,...);
  int yyparse(void);
  #include <stdio.h>
  extern int yylineno;
%}
%token num
%%
P : num ':' { printf("["); } L { printf("]"); }
  ;
L : num       { if($-2 == 0)printf("%d",$1);    $$=0; }
  | L ',' num { if($-2 == $1+1)printf("%d",$3); $$=$1+1; }
  ;
%%
void yyerror(const char *fmt,...) {
   printf("%s in line %d\n", fmt, yylineno); }
int main() { return yyparse(); }
```

wersja S-atrybutowa:

```
%{
  int yylex(void);
  void yyerror(const char *,...);
  int yyparse(void);
  #include <stdio.h>
  extern int yylineno;
%}
%token num
%%
S : { printf("["); } P { printf("]"); }
  ;
P : num ':' num { if($1 == 0)printf("%d",$3); $$=$1-1; }
  | P ',' num   { if($1 == 0)printf("%d",$3); $$=$1-1; }
  ;
%%
void yyerror(const char *fmt,...) {
   printf("%s in line %d\n", fmt, yylineno); }
int main() { return yyparse(); }
```

**Zadanie 2:**

```
%{
  int yylex(void);
  void yyerror(const char *,...);
  int yyparse(void);
  #include <stdio.h>
  extern int yylineno;
%}
%token num
%%
P : T       { if($1 != 1)puts("Error");
              else puts("OK");
            }
  ;
T : T L     { if($1 == $2+1)$$=$2;
              else $$ = -1;
            }
  | L       { $$ = $1; }
  ;
L : G '\n' { $$ = $1; }
  ;
G : G '*'  { $$ = $1 + 1; }
  | '*'    { $$ = 1; }
  ;
%%
void yyerror(const char *fmt,...) {
   printf("%s in line %d\n", fmt, yylineno); }
int main() { return yyparse(); }
```

**Zadanie 3:**
rozwiązanie (dla przejrzystości) zawiera tylko szkielety implementacji funkcji:

- addVar(char *id) dodającej identyfikatory do listy

oraz

- void printVars(const char *type) drukującej elementy z listy oraz zwalniającą zajmowaną przez listę pamięć

```
%union
{
  char *text;
}
%{
  int yylex(void);
  void yyerror(const char *,...);
  int yyparse(void);
  #include <stdio.h>
  extern int yylineno;
%}
%{
    struct var
    {
      char *name;
      struct var *next;
    } *head = NULL;
    void addVar(char *);
    void printVars(const char *);
%}
%token KWD_VAR
%token KWD_INTEGER KWD_CHAR
%token <text> ID
%%
P : KWD_VAR L ':' T ';'
  ;
L : ID           { addVar($1); }
  | L ',' ID     { addVar($3); }
  ;
T : KWD_CHAR     { printVars("char");    }
  | KWD_INTEGER  { printVars("integer"); }
  ;
%%
void yyerror(const char *fmt,...) {
   printf("%s in line %d\n", fmt, yylineno); }
int main() { return yyparse(); }
void addVar(char *id)
{ /* funkcja dodająca identyfikator do listy */ };
void printVars(const char *type)
{ /* funkcja drukująca identyfikatory z listy */ };
```