Podstawy Kompilatorów

Laboratorium 7

Translacja sterowana składnią w metodzie zstępującej.

Zadanie 1:
Proszę napisać program, który dla danej liczby całkowitej *j* oraz niepustego ciągu liczb naturalnych c_0, c_1, …, c_j, …, 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 dwóch wersjach:
- z wykorzystaniem języka C
- z użyciem generatora LLgen

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:

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

Zawartość pliku *Lpars.h* dla implementacji w C:

```
#define num 257
```

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:

```
*
```

Zadanie polega na skonstruowaniu analizatora, który będzie rozpoznawał, czy plik wejściowy zbudowany jest zgodnie z powyższymi założeniami.
Proszę napisać akceptor w dwóch wersjach:

- z wykorzystaniem języka C
- z użyciem generatora LLgen

Konstruując analizator składniowy nie wolno posługiwać się zmiennymi globalnymi.

Uwaga: każda linia pliku zakończona jest znakiem końca wiersza.

Analizator leksykalny ma następującą postać:

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

Odpowiedzi do zadań

Zadanie 1:
schemat translacji:

```
P  → num ':'   { L.n := num.val; } L
L  → num       { if L.n = 0 then writeln(num.val);
                    RL.n := L.n - 1;
                 }
       RL
RL → ','
       num     { if RL.n = 0 then writeln(num.val);
                 if RL.n <> -1 then RL1.n := RL.n - 1;
                 else RL1.n := -1;
                 }
       RL1
RL → ε
```

LLgen:

```
{
  #include <stdio.h>
  #include <stdlib.h>
  int LLlval;
}
%token num;
%start parse, P ;
RL(int n)    : ','
               num {if(n==0)printf("%d",LLlval); if(n!=-1)n--; }
               RL(n)
             |
             ;
L(int n)     : num { if(n==0)printf("%d",LLlval); }
               RL(n-1)
             ;
P { int n; } : num  { n = LLlval; }
               ':'  { printf("["); }
               L(n) { printf("]"); }
             ;
{
  int main()
  {
    printf("\n");
    parse();
    printf("\n");
    return 0;
  }
  void LLmessage(int tk)
  {
    if(tk == -1)printf("Oczekiwany koniec pliku\n");
    else if(tk == 0)
           if(LLsymb < 256)printf("Nieoczekiwany token (%c) usunięty\n",LLsymb);
           else printf("Nieoczekiwany token o numerze (%d) usunięty\n",LLsymb);
           else if(tk < 256)printf("Oczekiwano tokenu (%c)\n",tk);
         else printf("Oczekiwano tokenu o numerze (%d)\n",tk);
    exit(1);
  }
}
```

C:

```
#include <stdio.h>
#include <stdlib.h>
#include "Lpars.h"

int LLlval;
int yylex(void);
/* ================================================ */
int LLcsymb;
int LLlookAhead(void) { return LLcsymb; }
void LLread(void)   { LLcsymb = yylex(); }
void InitLexScanner(void) { LLcsymb = yylex(); }
/* ================================================ */
void RL(int n)
{
  switch(LLlookAhead())
  {
     case ',' : LLread();
                switch(LLlookAhead())
                {
                  case num : LLread();
                             if(n == 0)printf("%d",LLlval);
                             if(n == -1)RL(-1);
                             else RL(n-1);
                             return;
                  default  : puts("Brak liczby");
                             exit(1);
                }
     case 0   : return;
     default  : puts("Brak ,");
                exit(1);
  }
}
void L(int n)
{
  switch(LLlookAhead())
  {
    case num : LLread();
               if(n == 0)printf("%d",LLlval);
               RL(n-1);
               return;
    default  : puts("Brak liczby 2");
               exit(1);
  }
}
void P(void)
{
  switch(LLlookAhead())
  {
    case num : {
                 int n = LLlval;
                 printf("[");
                 LLread();
                 switch(LLlookAhead())
                 {
                   case ':' : LLread();
                              L(n);
                              printf("]");
                              return;
                   default  : puts("Brak liczby 1");
                              exit(1);
                 }
               }
    default  : puts("Brak liczby");
               exit(1);
  }
}
int main()
{
  InitLexScanner();
  P();
  return 0;
}
```

Zadanie 2:

schemat translacji:

```
T  → G RT            { if G.l=RT.len+1
                          then writeln('OK')
                          else writeln('Errror');
                     }
RT  →                { RT.len := 0; }
RT  →  G RT1         { if G.l = RT1.len+1
                        then RT.len:=G.l
                        else RT.len:=-1;
                     }
G  → '*' RG          { G.l := RG.len + 1; }
RG  → '\n'           { RG.len := 0; }
RG  → '*' RG1        { RG.len := RG1.len + 1; }
```

LLgen:

```
{
  #include <stdio.h>
  #include <stdlib.h>
}
%start parse, T ;
T { int l1, l2; } : G(&l1) RT(&l2) { if(l1==l2+1)printf("OK");
                                     else printf("Error");
                                   }
                  ;
RT(int *len) { int l1; } :                  { *len = 0;  }
                         | G(&l1) RT(len)   { if(l1 == (*len)+1)(*len)=l1;
                                              else (*len)=-1;
                                            }
                         ;

G(int *l) { int tmp_l; } : '*' RG(&tmp_l) { *l = tmp_l + 1; }
                         ;

RG(int *len) : '\n'          { *len = 0; }
             | '*' RG(len)   { (*len)++; }
             ;
{
  int main()
  {
    printf("\n");
    parse();
    printf("\n");
    return 0;
  }
  void LLmessage(int tk)
  {
    if(tk == -1)printf("Oczekiwany koniec pliku\n");
    else if(tk == 0)
           if(LLsymb < 256)printf("Nieoczekiwany token (%c) usunięty\n",LLsymb);
           else printf("Nieoczekiwany token o numerze (%d) usunięty\n",LLsymb);
           else if(tk < 256)printf("Oczekiwano tokenu (%c)\n",tk);
         else printf("Oczekiwano tokenu o numerze (%d)\n",tk);
    exit(1);
  }
}
```

C:

```
#include <stdio.h>
#include <stdlib.h>
int yylex(void);
/* ================================================= */
int LLcsymb;
int LLlookAhead(void) { return LLcsymb; }
void LLread(void)    { LLcsymb = yylex(); }
void InitLexScanner(void) { LLcsymb = yylex(); }
/* ================================================= */
void RG(int *len)
{
  switch(LLlookAhead())
  {
    case '\n' : LLread(); *len = 0;
                return;
    case '*'  : LLread(); RG(len); (*len)++;
                return;
    default   : puts("Oczekiwana * lub nowa linia\n");
                exit(1);
  }
}
void G(int *l)
{
  int tmp_l;
  switch(LLlookAhead())
  {
    case '*' : LLread(); RG(&tmp_l); *l = tmp_l + 1;
               return;
    default  : puts("Oczekiwana *\n");
               exit(1);
  }
}
void RT(int *len)
{
  int l1;
  switch(LLlookAhead())
  {
     case 0   : *len = 0;
                return;
     default  : G(&l1); RT(len);
                if(l1 == (*len)+1)(*len)=l1; else (*len)=-1;
                return;
  }
}
void T(void)
{
  int l1, l2;
  G(&l1);
  RT(&l2);
  if(l1==l2+1)puts("OK"); else puts("Error");
}
int main()
{
 InitLexScanner();
  T();
  return 0;
}
```