Podstawy Kompilatorów

Laboratorium 6

Generator LLgen.

Zadanie 1:
Proszę napisać akceptor dla języka $a^n b^m$ (n, m > 0).

Przykłady:
Dla pliku o postaci:
aaabb
powinniśmy otrzymać wynik:
OK

Dla pliku o postaci:
aaabba
powinniśmy otrzymać komunikat o błędzie.

Analizator leksykalny ma następującą postać:

```
%{
  #include <stdio.h>
  int yywrap(void);
  int yylex(void);
  #include "Lpars.h"
%}
%%
a   { return 'a'; }
b   { return 'b'; }
%%
int yywrap(void) { return 1; }
```

funkcja *LLmessage* ma następującą postać:

```
LLmessage(int tk)
{
  printf("\nError code: (%d)\n",tk);
  exit(1);
}
```

Zadanie 2:
Proszę napisać akceptor dla języka $a^n b^n$ (a, b ≥ 0).

Przykłady:
Dla pliku o postaci:
aabb
powinniśmy otrzymać wynik:
OK

Dla pliku o postaci:
aaabb
powinniśmy otrzymać wynik (w nawiasach liczba liter *a* i *b* napotkanych na wejściu):
Error (3,2)

Dla pliku o postaci:
aaabba
powinniśmy otrzymać komunikat o błędzie.

Analizator leksykalny ma następującą postać:

```
%{
  #include <stdio.h>
  int yywrap(void);
  int yylex(void);
  #include "Lpars.h"
%}
%%
a   { return 'a'; }
b   { return 'b'; }
%%
int yywrap(void) { return 1; }
```

funkcja *LLmessage* ma następującą postać:

```
LLmessage(int tk)
{
  printf("\nError code: (%d)\n",tk);
  exit(1);
}
```

Zadanie 3:
Proszę napisać akceptor dla języka $c_1|c_2$, gdzie c_1 jest łańcuchem cyfr oktalnych, a c_2 jest lustrzanym odbiciem c_1. Oba łańcuchy mogą być puste, a więc plik składający się tylko ze znaku '|' jest również poprawny.

Przykłady:
Dla pliku o postaci:
123|321
powinniśmy otrzymać wynik:
OK

Dla pliku o postaci:
123|323
powinniśmy otrzymać komunikat o błędzie.

Dla pliku o postaci:
123|3211
powinniśmy otrzymać komunikat o błędzie.

Analizator leksykalny ma następującą postać:

```
%{
  #include <stdio.h>
  int yywrap(void);
  int yylex(void);
  #include "Lpars.h"
%}
%%
[0-7]   { return yytext[0]; }
"|"     { return '|'; }
%%
int yywrap(void) { return 1; }
```

funkcja *LLmessage* ma następującą postać:

```
LLmessage(int tk)
{
  printf("\nError code: (%d)\n",tk);
  exit(1);
}
```

Zadanie 4:
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 akceptor, który będzie rozpoznawał, czy plik wejściowy zbudowany jest zgodnie z powyższymi założeniami.

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;}
```

funkcja *LLmessage* ma następującą postać:

```
LLmessage(int tk)
{
  printf("\nError code: (%d)\n",tk);
  exit(1);
}
```

Odpowiedzi do zadań

Zadanie 1:

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

%start parse, spec ;

spec : 'a'+ 'b'+ { puts("OK"); }
     ;
{
  int main()
  {
    printf("\n");
    parse();
    printf("\n");
    return 0;
  }
  LLmessage(int tk)
  {
    printf("\nError code: (%d)\n",tk);
    exit(1);
  }
}
```

Zadanie 2:

```
{
  #include <stdio.h>
  #include <stdlib.h>
  int l_a = 0, l_b = 0;
}

%start parse, spec ;

spec : LA LB { if(l_a != l_b)printf("\nError (%d,%d)\n",l_a,l_b);
               else printf("\nOK\n");
             }
     ;
LA   :
     | 'a' { l_a++; } LA
     ;
LB   :
     | 'b' { l_b++; } LB
     ;
{
  int main()
  {
    printf("\n");
    parse();
    printf("\n");
    return 0;
  }
  LLmessage(int tk)
  {
    printf("\nError code: (%d)\n",tk);
    exit(1);
  }
}
```

Zadanie 3:

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

%start parse, spec ;

spec : ciag
     ;
ciag : '0' ciag '0'
     | '1' ciag '1'
     | '2' ciag '2'
     | '3' ciag '3'
     | '4' ciag '4'
     | '5' ciag '5'
     | '6' ciag '6'
     | '7' ciag '8'
     | '|'
     ;
{
  int main()
  {
    printf("\n");
    parse();
    puts("OK");
    printf("\n");
    return 0;
  }
  LLmessage(int tk)
  {
    printf("\nError code: (%d)\n",tk);
    exit(1);
  }
}
```

Zadanie 4:

```
{
  #include <stdio.h>
  #include <stdlib.h>
  int l_g = 0, o_l_g = 1;
}

%start parse, spec ;

spec : T
     ;
T : L { o_l_g = l_g - 1; } RT
  ;
RT :
   | L { if(o_l_g != l_g)printf("Error"); o_l_g--;
       }
     RT
   ;
L : '\n'    { l_g = 0; }
  | '*' L   { l_g++; }
  ;
{
  int main()
  {
    printf("\n");
    parse();
    printf("\n");
    return 0;
  }
  LLmessage(int tk)
  {
    printf("\nError code: (%d)\n",tk);
    exit(1);
  }
}
```