Podstawy Kompilatorów

Laboratorium 3

Uwaga: Do wykonania poniższych zadań związanych z implementacją niezbędny jest program LEX oraz kompilator.

Dla środowiska Linux mogą to być:

- ***Darmowa wersja generatora analizatorów leksykalnych - FLEX, dostępna na stronie:*** **http://www.gnu.org/software/flex**
- ***Kompilator – GCC, do pobrania na stronie:*** **http://gcc.gnu.org/**

Dla środowiska Windows zalecane jest środowisko cygwin zawierające w swoich pakietach zarówno program FLEX, jak i kompilator GCC. Środowisko można pobrać za darmo ze strony: **http://www.cygwin.com/**

Wprowadzenie

Program LEX służy do pisania analizatorów leksykalnych. Generuje on z pliku przygotowanego przez użytkownika i zawierającego reguły przetwarzania - kod źródłowy analizatora w języku C. Wygenerowany plik analizatora należy skompilować a następnie uruchomić przesyłając jako strumień danych wejściowych plik z danymi do analizy. Przykładowa sekwencja poleceń prowadzących do wygenerowania analizatora może wyglądać następująco:

```
flex -oscan.c scan.l
gcc scan.c -lfl -o scan.exe
```

Dla analizatora o nazwie scan.exe plik z danymi wejściowymi dane.txt będzie przesłany na wejście następująco:

```
scan.exe < dane.txt
```

Oto szkielet pliku ze specyfikacją dla generatora analizatorów FLEX:

```
%{
#include <stdio.h>
int yywrap();
int yylex();
%}
%%
a     {puts("Znaleziono litere a");}
%%
int yywrap() { return 1;}
int main() { return yylex(); }
```

Zadania do wykonania

Zad. 1:
Proszę napisać wyrażenia regularne w konwencji programu LEX, do których będą pasowały następujące napisy:

- ciąg znaków „john doe”
- pojedyncza spacja
- ciąg znaków john lub ciąg znaków doe
- dowolny pojedynczy znak różny od symbolu \n
- napis utworzony z jednego lub więcej dowolnych znaków różnych od \n
- napis utworzony z zera lub więcej dowolnych znaków różnych od \n
- napis utworzony z zera lub jednego z dowolnych znaków różnych od \n

- napis utworzony z jednego lub więcej znaków z klasy składającej się ze znaków od a do z
- napis utworzony z zera lub więcej znaków z klasy składającej się ze wszystkich znaków oprócz liter A, B oraz C

Zad. 2:
Jakie dane wejściowe zostaną dopasowane do poniższych wyrażeń regularnych?
a) \++**
b) \n
c) [a-zA-Z_][a-zA-Z0-9_]*

Zad. 3:
Proszę napisać za pomocą generatora LEX program usuwania wszystkich początkowych spacji w każdym wierszu.

Na przykład, jeśli plik wejściowy ma postać:
12345 abc ;
  345 abc ;
:-) :-(;-)
   :-) :-(

to na wyjściu powinniśmy otrzymać:
12345 abc ;
345 abc ;
:-) :-(;-)
:-) :-(

Zad. 4:

Plik wejściowy ma postać ciągu wierszy. Jeśli w danym wierszu występuje tylko jedna liczba i ewentualnie spacje przed i po liczbie, to należy sprawdzić czy jest to liczba podzielna przez 5 i jeśli tak, to na końcu wiersza dopisać "(+)", a jeśli nie to "(-)". Jeśli wiersz ma inną konstrukcję, to należy go przepisać bez zmian.

Na przykład, jeśli plik wejściowy będzie miał postać (w drugim wierszu po liczbie 12 występują jeszcze trzy spacje):

```
15 12 20
   12
15
     20
a= 20
```

to na wyjściu powinniśmy otrzymać:

```
15 12 20
   12   (-)
15(+)
     20(+)
a= 20
```

Uwaga!
Program powinien działać poprawnie również dla wejścia:

```
15 12 20
  55555555555555555555555555555555555555555
11111111111111111111111111111111
```

Dla którego na wyjściu powinniśmy otrzymać:

```
15 12 20
  55555555555555555555555555555555555555555(+)
11111111111111111111111111111111(-)
```

Zad. 5:

Identyfikator to niepusty ciąg znaków zaczynający się od litery lub znaku "_". Dalej mogą następować litery, cyfry i znaki "_". Napisać w LEXie program prostego analizatora leksykalnego dla podzbioru języka Pascal rozpoznający identyfikatory i słowa kluczowe begin i end. Należy przyjąć, że duże i małe litery nie są rozróżniane. W przypadku rozpoznania identyfikatora należy wydrukować na wyjściu ID, a w przypadku słowa kluczowego KWD.

Na przykład dla pliku wejściowego:

```
begin abra ka := dabra EnD
```

powinniśmy otrzymać:

```
KWD ID ID := ID KWD
```

Odpowiedzi do zadań

Zad. 1:

- ”john doe”
- ” ”
- john|doe
- .
- .+
- .*
- .?
- [a-z]+
- [^ABC]*

Zad. 2:

a) +, ++, +++, +*, ++*, +++*, +**, ++**, ++***, ...
b) \n (symbol końca linii)
c) Napis będący identyfikatorem języka C, czyli np.: funkcja, id, _nazwa, zmienna_1, i, main, ...

Zad. 3:

```
%{
#include <stdio.h>
int yywrap();
int yylex();
%}
%%
^" "+
%%
int yywrap() { return 1;}
int main() { return yylex(); }
```

Zad. 4:

```
%{
#include <stdio.h>
int yywrap();
int yylex();
%}
%%
^[ \t]*[0-9]*[1-46-9][ \t]*$       printf ("%s(-)", yytext);
^[ \t]*[0-9]*[05][ \t]*$           printf ("%s(+)", yytext);
%%
int yywrap() { return 1;}
int main() { return yylex(); }
```

Zad. 5:

```
%{
#include <stdio.h>
int yywrap();
int yylex();
%}
%%
[a-zA-Z_][a-zA-Z0-9_]*              printf ("ID");
[Bb][Ee][Gg][Ii][Nn]|[Ee][Nn][Dd]   printf ("KWD");
%%
int yywrap() { return 1;}
int main() { return yylex(); }
```