J.NAWROCKI, M. ANTCZAK, H. ĆWIEK, W. FROHMBERG, A. HOFFA, M. KIERZYNKA, S. WĄSIK

JĘZYKI FORMALNE I GRAMATYKI

WPROWADZENIE

Napisz analizator leksykalny (*LEX*) i analizator składniowy (*YACC*), który będzie wykorzystywany przez akceptor łańcuchów o postaci: $c_1|c_2$ gdzie c_1 jest ciągiem cyfr oktalnych (`[0-7]`), a c_2 lustrzanym odbiciem c_1. Ciągi c_1 i c_2 mogą być puste, zatem łańcuch zawierający jedynie znak `'|'` jest poprawny. Łańcuch poprawny (np.: `123|321`) winien być zaakceptowany komunikatem *"Syntax OK"*, natomiast niepoprawny (np.: `1234|3321`) - odrzucony komunikatem *"Syntax error !"*.

W celu wykonania powyższego zadania należy:

- pobrać pakiet, który zawiera narzędzia, które pozwolą na skonfigurowanie środowiska uruchomieniowego, które będzie wykorzystywane podczas tworzenia akceptora,
- stworzenie analizatora leksykalnego (z wykorzystaniem narzędzia *LEX*) oraz analizatora składniowego (za pomocą narzędzia *YACC*),
- kompilacja i uruchomienie napisanych w poprzednim kroku analizatorów; wynik powinien zawierać ocenę poprawności poszczególnych wierszu znajdujących się w pliku wejściowym.

1. Instalacja i konfiguracja narzędzi

Pobierz i rozpakuj pakiet, w dowolnej lokalizacji nie zawierającej spacji, korzystając z adresu: `http://www.cs.put.poznan.pl/mantczak/teaching/itc/Lex&Yacc.zip`.
Po pomyślnym rozpakowaniu (np.: na dysku E:\) powinien się pojawić katalog Temp (np.: `E:\Temp`).
Katalog Temp zawiera następujące katalogi:

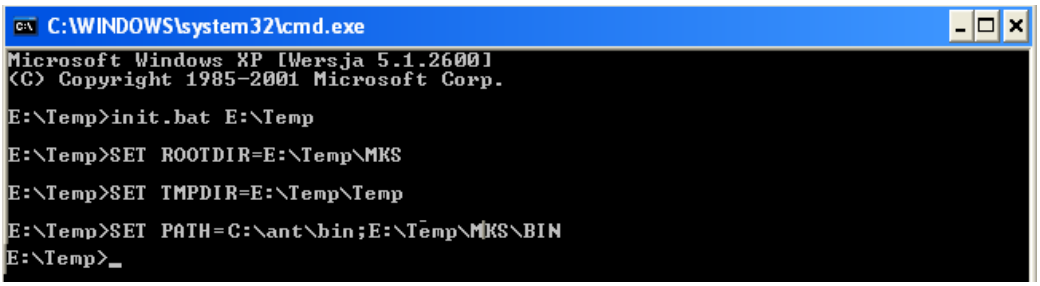
- MKS – pakiet *„MKS Lex and Yacc for DOS"* w wersji 3.2. Ukazał się na rynku w 1993 roku. Pomimo swojego wieku zawiera w pełni użyteczne wersje `LEXa` i `YACCa`. W późniejszych latach ukazały się poprawki do pakietu, które umożliwiły przetwarzanie większych specyfikacji i poprawiły kompatybilność narzędzi (teraz używamy 3.2a). Strona producenta pakietu: `http://www.mks.com`. W celu instalacji narzędzi dla innego systemu operacyjnego proszę skorzystać z informacji zawartych pod adresem `http://www.cs.put.poznan.pl/wcomplak/JF-tools.htm`.
- `TC` – w celu korzystania z pakietu potrzebny jest także kompilator języka C. Wypróbowaliśmy z powodzeniem (co nie oznacza, że pierwsza kompilacja zakończy się powodzeniem) dobrze znany nam już z ćwiczeń dotyczących programowania imperatywnego kompilator Turbo C.
- `Temp` – katalog wykorzystywany przez `Lex-a` i `Yacc-a` w celu przechowywania plików tymczasowych, które pojawiają się podczas korzystania z powyższych narzędzi.
- Plik `init.bat`, w którym zawarte są ustawienia zmiennych środowiskowych systemu Windows, które są wymagane podczas korzystania z powyższych narzędzi.

```
SET ROOTDIR=%1\MKS          // ścieżka do głównego katalogu z
                            // narzędziami
SET TMPDIR=%1\Temp          // ścieżka do katalogu, w którym
                            // tworzone są
```

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

```
                                    // pliki tymczasowe przez
                                    // wykorzystywane
                                    // narzędzia
SET PATH=%PATH%;%1\MKS\BIN          // dodanie ścieżki dostępu do
                                    // narzędzi do
                                    // zmiennej systemowej PATH w celu
                                    // możliwości odwoływania się do
                                    // narzędzi z
                                    // konsoli uruchomionej w dowolnym
                                    // katalogu.
```

Przejście do wnętrza katalogu `Temp` i uruchomienie w nim konsoli systemowej za pomocą polecenia `cmd`.
Uruchomienie w tej nowo utworzonej konsoli systemowej pliku `init.bat` z przekazaną jako parametr ścieżką do katalogu `Temp` (np.: `init.bat E:\Temp`).



W tym momencie w tej konkretnej konsoli systemowej jest zdefiniowane środowisko uruchomieniowe, które jest wystarczające w celu realizacji powyższego zadania. Ważną kwestią jest, że jeżeli zamknięte zostanie ta konsola systemowa i uruchomiona kolejna, wtedy w tej i każdej następnej musi zostać uruchomione polecenie z punktu 1.5 przed rozpoczęciem pracy z narzędziami Lex i Yacc.

2. Konstrukcja analizatora leksykalnego i składniowego

Celem analizatora leksykalnego jest sprawdzenie czy dane znajdujące się w wejściowym pliku są poprawnego typu (rodzaju).

Analizując treść powyższego zadania wiemy, że przykładem poprawnych danych wejściowych jest np.: `123|321`. W takim razie wiemy, że na wejściu mogą się pojawić tylko cyfry oraz znak `'|'`. Dodatkowo wiemy, że mogą to być tylko cyfry tworzące liczby oktalne, a więc ograniczamy zakres naszych cyfr od `0-7`. Wynik analizy leksykalnej zakończy się powodzeniem tylko wtedy, gdy w pliku wejściowym znajdować się będą tylko cyfry w zakresie od `0-7` oraz znak `'|'`. Na tym etapie niebadana jest ilość znaków, tzn. czy w pliku wejściowym znajduje się tylko jeden znak `'|'`. Taka analiza dokonywana jest podczas analizy składniowej w kolejnym etapie, do którego można przejść tylko wtedy, gdy etap analizy leksykalnej zakończy się sukcesem. Jeżeli na wejściu zostanie wyszczególniony, co najmniej jeden znak, który nie jest cyfrą z zakresu od `0-7` lub znakiem `'|'` wtedy analiza leksykalna kończy się niepowodzeniem. Poniżej znajduje się zawartość pliku `scan.l`, który zawiera implementację prostego analizatora leksykalnego zapisanego w języku Lex:

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

```
%%
[0-7]    {return yytext[0];}     // gdy na wejściu znajduje się
                                 // cyfra [0-7] przekaż ją do
                                 // dalszego etapu analizy
"|"      {return '|';}           // gdy na wejściu znajduje się
                                 // znak '|'przekaż go do dalszego
                                 // etapu analizy
.        {YY_FATAL("ERR!");}     // jeżeli na wejściu pojawi się
                                 // dowolny znak ('.'), który nie
                                 // został obsłużony przez żadną z
                                 // powyższych reguł bazujących na
                                 // wzorcach to przerwij analizę
                                 // leksykalną z błędem
                                 // sygnalizujących błędnie
                                 // przygotowaną zawartość pliku
                                 // wejściowego
```

Celem analizatora składniowego jest sprawdzenie czy dane wejściowe poprawnego typu są zgodne z przyjmowaną przez nas jako poprawną składnię języka.

W przypadku powyższego zadania przykładem poprawnych danych wejściowych jest np.: `123|321`. Zauważmy w takim razie, że `123|321 = 1 S 1 => S = 23|32 = 2 S 2 => S = 3|3 => 3 S 3 => S = |`, gdzie `S` to rodzaj podwyrażenia. Dla powyższego języka można zdefiniować następującą gramatykę, która może zostać zapisana z wykorzystaniem narzędzie Yacc (poniżej przedstawiona zostaje zawartość pliku `scang.y`):

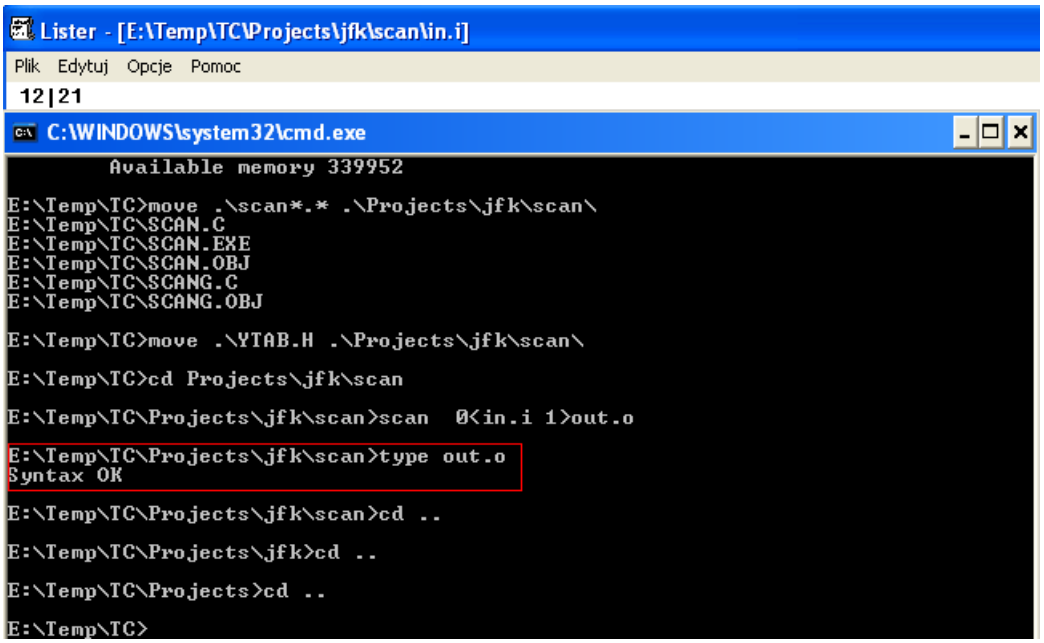
```
%{
#include <stdio.h>               // pliki nagłówkowe zawierające
#include <dos.h>                 // funkcje, które będą używane
#include <stdlib.h>              // w dalszych sekcjach kodu
#include <process.h>             // źródłowego
%}
%%                               // rozpoczyna się sekcja gramatyki
E : S {printf("Syntax OK\n");}   // potwierdź na wyjściu popr. skł.
  ;
S : '0' S '0'                    // kolejne reguły, które definiują
  | '1' S '1'                    // poprawną składnię języka
  | '2' S '2'                    // zdefiniowanego w treści zadania
  | '3' S '3'
  | '4' S '4'
  | '5' S '5'
  | '6' S '6'
  | '7' S '7'
  | '|'
  ;
%%                               // impl. pokrywanych funkcji yacca
void yyerror(char* msg)          // kod funkcji, który zostanie
{                                // wywołany w momencie, gdy
printf("Syntax error !\n");      // wyrażenia w pliku wejściowym
exit(1);                         // nie będą się zgadzać składniowo
}                                // z językiem z treści zadania
```

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

3. Kompilacja i uruchomienie i testowanie

Powyższe pliki wraz z plikiem zawierającym dane wejściowe `in.i` znajdują się w katalogu projektu `scan` w katalogu kompilatora `TC` (np.: `E:\Temp\TC\Projects\jfk\scan`). W celu uruchomienia powyższego projektu został napisany plik `makeJF.bat`, który znajduję się w katalogu głównym kompilatora (np.: `E:\Temp\TC`).

- Przechodzimy do katalogu kompilatora `TurboC` za pomocą polecenia `cd TC`.
- Kompilacja i uruchamianie projektu `scan` znajdującego się w katalogu `Projects\jfk`, w którym powinny się znajdować katalogi kolejnych tworzonych projektów (np.: `E:\Temp\TC\Projects\jfk\scan`) za pomocą polecenia `makeJF.bat nazwa_projektu (np.: makeJF.bat scan).`



Dodatkowe materiały

Zakładka *„materiały dla studentów”* dla przedmiotu *„języki formalne i kompilatory”* na stronie domowej dr inż. W. Complaka – `http://www.cs.put.poznan.pl/wcomplak`.

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.