Podstawy Kompilatorów

Laboratorium 9

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

Dla środowiska Linux mogą to być:

- ***Darmowa wersja generatora analizatorów leksykalnych - FLEX, dostępna na stronie:*** **http://www.gnu.org/software/flex**
- ***Darmowa wersja generatora analizatorów składniowych – BISON, dostępna na stronie:*** **http://www.gnu.org/software/bison/**
- ***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 YACC jest generatorem analizatorów składniowych. Na podstawie specyfikacji dostarczonej przez użytkownika można za jego pomocą wygenerować plik źródłowy w języku C zawierający kod analizatora składniowego. Kod taki należy skompilować do postaci binarnej a następnie uruchomić przekazując dane wejściowe jako strumień wejściowy. Przykładowo dla specyfikacji gram.y możemy wygenerować plik gram.c a następnie skompilować go do gram.exe. Przykładowa sekwencja poleceń prowadzących do wygenerowania analizatora może wyglądać następująco:

```
bison gram.y -d -o gram.c
gcc gram.c -o parser.exe
```

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

```
parser.exe < dane.txt
```

Jeżeli dodatkowo jako generator analizatora leksykalnego użyty zostanie program LEX to odpowiednia sekwencja wyglądać będzie następująco:

```
bison gram.y -d -o gram.c
flex -oscan.c scan.l
gcc scan.c gram.c –O3 –lfl -o parser.exe
parser.exe < dane.txt
```

Oto szkielety plików ze specyfikacją dla współpracujących ze sobą generatora analizatorów leksykalnych FLEX oraz generatora analizatorów składniowych BISON:

scan.l:

```
%{
#include "gram.h"
%}

%%
```

gram.y:

```
%{
#include <string.h>
#include <stdlib.h>
%}
%%

%%
int
yyerror ()
{
    printf ("ERROR!\n");
    exit(-1);
}

int
main ()
{
    yyparse ();
}
```

Zadania do wykonania

Przyjmijmy dla uproszczenia, że analizator leksykalny (plik scan.l) we wszystkich zadaniach ma następującą postać:

```
%{
#include "gram.h"
%}

%%
.    return yytext[0];
```

Zad. 1:
Proszę napisać specyfikację dla programu YACC będącą akceptorem języka
$a^n b^n$, gdzie n > 0. Jeśli na wejściu pojawi się napis należący do tego języka, program powinien wypisać „OK.”. W przeciwnym razie powinien wypisać „ERROR!”

Zad. 2:
Proszę napisać specyfikację dla programu YACC będącą akceptorem języka
$a^n b^m c^n$, gdzie n > 0, m >= 0. Jeśli na wejściu pojawi się napis należący do tego języka, program powinien wypisać „OK.”. W przeciwnym razie powinien wypisać „ERROR!”

Zad. 3:
Proszę napisać specyfikację dla programu YACC będącą akceptorem języka
$a^n b^n c^m d^m$, gdzie n > 0, m >= 0. Jeśli na wejściu pojawi się napis należący do tego języka, program powinien wypisać „OK.”. W przeciwnym razie powinien wypisać „ERROR!”

Zad. 4:
Napisać analizator sprawdzający czy dane wejściowe mają postać:

$c_1|c_2$

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. Także dane wejściowe: 123|321
są poprawne, ale 1234|3321
są błędne. Jeżeli dane wejściowe są poprawne analizator powinien dać odpowiedź "OK", a w przeciwnym przypadku "ERROR!".

Odpowiedzi do zadań

Zad. 1:

```
%{
#include <string.h>
#include <stdlib.h>
%}
%%

X: S {puts("OK");}
 | S error {yyerror();}
 ;
S: 'a' S 'b'
 | 'a' 'b'
 ;

%%
int
yyerror ()
{
    printf ("ERROR!\n");
    exit(-1);
}

int
main ()
{
    yyparse ();
}
```

Zad. 2:

```
%{
#include <string.h>
#include <stdlib.h>
%}
%%

X: S {puts("OK");}
 | S error {yyerror();}
 ;
S: 'a' S 'b'
 | 'a' C 'b'
 ;
C: C 'c'
 |
 ;

%%
int
yyerror ()
{
    printf ("ERROR!\n");
    exit(-1);
}

int
main ()
{
    yyparse ();
}
```

Zad. 3:

```
%{
#include <string.h>
#include <stdlib.h>
%}
%%

X: AB CD {puts("OK");}
 | AB CD error {yyerror();}
 ;
AB: 'a' AB 'b'
 | 'a' 'b'
 ;
CD: 'c' CD 'd'
 |
 ;

%%
int
yyerror ()
{
    printf ("ERROR!\n");
    exit(-1);
}

int
main ()
{
    yyparse ();
}
```

Zad 4.:

```
%{
#include <string.h>
#include <stdlib.h>
%}
%%

X: S {puts("OK");}
 | S error {yyerror();}
 ;

S : '0' S '0'
  | '1' S '1'
  | '2' S '2'
  | '3' S '3'
  | '4' S '4'
  | '5' S '5'
  | '6' S '6'
  | '7' S '7'
  | '|'
  ;

%%
int
yyerror ()
{
    printf ("ERROR!\n");
    exit(-1);
}

int
main ()
{
    yyparse ();
}
```