Istruzioni per l'utilizzo di Bison/Flex - C

Sia data la seguente grammatica:

S → aSb

S → ab

Definizione delle regole lessicali e sintattiche

1. Creare un file `MyGrammar.y` dove andremo a definire le regole per l'analisi sintattica:

- Questo file deve rispettare la seguente struttura:

```
%{
// Qui le dichiarazioni dei prototipi delle funzioni usate
%}

// Dichiarazione dei token

%%
// Grammatica
%%

// Codice C addizionale e Main
```

  - **Dichiarazioni dei prototipi**: in questa sezione vanno inseriti i prototipi delle funzioni che verranno utilizzate dal parser e le librerie necessarie.

    ```
    %define parse.error verbose

    %{
    #include <stdio.h>

    int yylex();
    int yyerror(const char *msg);
    %}
    ...
    ```

  - **Dichiarazione dei token**: In questo esempio, essendo i token singoli caratteri, potremmo saltare questa parte ed inserirli direttamente nella grammatica. Ma nel caso in cui i token siano più complessi allora questa sezione è necessaria.

    ```
    ...
    %token A B
    ...
    ```

  - **Grammatica**: si tratta di riscrivere la grammatica nella forma EBNF (Extended Backus–Naur form):

    ```
    rule: prod_1 | prod_2 | ... | prod_n;
    ```

    Per gestire il problema dei prefissi, aggiungere un regola iniziale che produce l'assioma della nostra grammatica. Nel nostro caso:

    ```
    %%

    prog:
        srule
    ;

    srule
        : A srule B
        | A B
    ;

    %%
    ```

  - **Codice C addizionale e Main**: In questa sezione andremo a definire la funzione per la gestione degli errori e il Main.

```
int yyerror(const char *msg){
    fprintf(stderr, "%s\n", msg);
return 0;
}

int main()
{
    // Esecuzione Parsing
    int parse = yyparse();
    // Controllo del risultato
    if(parse == 0) printf("\nParsing result:  SUCCESS\n\n");
    return 0;
}
```

2. Creare un file `MyGrammar.l` dove andremo a definire le regole per l'analisi lessicale:

- Questo file deve rispettare la seguente struttura:

```
%{
// Qui le dichiarazioni dei prototipi delle funzioni usate
%}

// Definizione tokens

%%
// Regole per il Lexer
%%

// Codice C addizionale
```

  - **Dichiarazioni dei prototipi**: in questa sezione vanno inseriti i prototipi delle funzioni che verranno utilizzate dal lexer e le librerie necessarie.

```
%option noyywrap
%option yylineno

%{
#include <stdio.h>
#include <string.h>
#include "MyGrammar.tab.h"
%}
...
```

  - **Regole per il Lexer**: qui vanno definite le operazioni da effettuare quando viene riconosciuto un token. Un'operazione necessaria è fare il *return* del token. Questo deve ritornare il nominativo del token e dovrà essere lo stesso che verrà definito nel file 'MyGrammar.y'. Nel nostro esempio inseriamo anche una print per ogni token letto:

```
%%

"a"     {printf("[token at line %d: \"%s\"]\n", yylineno, yytext); return (A);}
"b"     {printf("[token at line %d: \"%s\"]\n", yylineno, yytext); return (B);}
.       {return *yytext;}

%%
```

    Come possiamo vedere oltre ai token della grammatica in input è necessario definire anche un token che equivalga a tutto quello che il Lexer deve segnalare come "token sconosciuto": Il ' . ' indica, infatti, "tutto tranne i token già definiti".

  - **NOTA AGGIUNTIVA: Dichiarazione tokens**: In questo esempio, essendo i token singoli caratteri, li abbiamo inseriti direttamente nella regole per il lexer. Ma nel caso in cui i token siano più complessi allora questa sezione è necessaria. Infatti qui possiamo definire i token tramite espressioni regolari. Se ad esempio al posto del carattere 'a' volessimo un numero qualsiasi allora dovremmo definire un token in questo modo: `num ([0-9])+;`. Il nostro esempio diventerebbe:

```
num ([0-9])+;

%%

{num}   {printf("[token at line %d: \"%s\"]\n", yylineno, yytext); return (NUM);}
"b"     {printf("[token at line %d: \"%s\"]\n", yylineno, yytext); return (B);}
.       {return *yytext;}

%%
```

Compilazione ed esecuzione

1. Generare l'analizzatore lessicale (lexer) definito in *MyGrammar.l* con flex:

```
flex -l example.l
```

2. Generare l'analizzatore sintattico (parser) definito in *MyGrammar.y* con bison:

```
bison -d example.y
```

3. Compilare i file appena generati:

```
gcc -o example example.tab.c lex.yy.c -lm
```

4. Eseguire passando come parametro di input il file a cui applicare il parsing:

```
./example < input-file
```