Esercizio 1

Codificare in Lex l'analizzatore lessicale di un linguaggio in cui ogni frase è composta da una o più dichiarazioni, come nel seguente esempio:

```
int x = 3, num = 100;
string A = "alfa", B = "beta";
boolean ok = true, end = false;
```

Esercizio 1

```
%{
#include <stdlib.h>
#include "def.h"
Lexval lexval;  /* typedef union {int ival; char *sval; } Lexval; */
%}
delimiter       [ \t\n]
spacing         {delimiter}+
letter          [A-Za-z]
digit           [0-9]
intconst        {digit}+
strconst        \"([^\"])*\"
boolconst       false|true
id              {letter}({letter}|{digit})*
%%
{spacing}       ;
int             {return(INT);}
string          {return(STRING);}
boolean         {return(BOOLEAN);}
{intconst}      {lexval.ival = atoi(yytext); return(INTCONST);}
{strconst}      {lexval.sval = newstring(yytext); return(STRCONST);}
{boolconst}     {lexval.ival = (yytext[0] == 'f' ? 0 : 1); return(BOOLCONST);}
{id}            {lexval.sval = newstring(yytext); return(ID);}
=               {return(ASSIGN};}
,               {return(COMMA);}
;               {return(SEMICOLON);}
.               {return(ERROR);}
%%
char *newstring(char *s)
{
  char *p = malloc(strlen(s)+1);
  strcpy(p,s);
  return(p);
}
```

```
int x = 3, num = 100;
string A = "alfa", B = "beta";
boolean ok = true, end = false;
```

Esercizio 2

Codificare in Lex l'analizzatore lessicale del linguaggio definito dalla seguente BNF:

program → *stat-list*
stat-list → *stat* **;** *stat-list* | **ε**
stat → *def-stat* | *if-stat* | **display**
def-stat → **id :** *type*
type → **int** | **string**
if-stat → **if** *expr* **then** *stat* **else** *stat*
expr → **boolconst**

assumendo che:

- un identificatore è una sequenza di lettere minuscole, eventualmente separate da un (solo) underscore, come nel seguente esempio: **alfa_beta_gamma**;
- una costante booleana può essere **true** o **false**.

Esercizio 2

```
%{
#include <stdlib.h>
#include "def.h"  /* encoding of lexical symbols */
Lexval lexval;    /* union {ival, sval} */
%}
delimiter    [ \t\n]
spacing      {delimiter}+
lowercase    [a-z]
id           {lowercase}+('_'{lowercase}+)*
boolconst    false | true
sugar        [;:]
%%
{spacing}    ;
{sugar}      {return(yytext[0]);}
display      {return(DISPLAY);}
int          {return(INT);}
string       {return(STRING);}
if           {return(IF);}
then         {return(THEN);}
else         {return(ELSE);}
{boolconst}  {lexval.ival = (yytext[0]=='f' ? 0 : 1); return(BOOLCONST);}
{id}         {lexval.sval = newstring(yytext); return(ID);}
.            {return(ERROR);}
%%
char *newstring(char *s)
{
  char *p = malloc(strlen(s)+1);
  strcpy(p,s);
  return(p);
}
```

program → *stat-list*
stat-list → *stat* **;** *stat-list* | **ε**
stat → *def-stat* | *if-stat* | **display**
def-stat → **id :** *type*
type → **int** | **string**
if-stat → **if** *expr* **then** *stat* **else** *stat*
expr → **boolconst**

Esercizio 3

Codificare in Lex l'analizzatore lessicale del linguaggio definito dalla seguente EBNF:

program → { *stat* **;** }$^{+}$
stat → *def-stat* | *procedure-call*
def-stat → **id** { **,** **id** } **:** *type*
type → **int** | **string** | **bool** | *structured-type*
structured-type → **matrix** **[** **intconst** { **,** **intconst** } **]** **of** *type*
procedure-call → **call** **id** **(** [*parameters*] **)**
parameters → *param* { **,** *param* }
param → **intconst** | **stringconst** | **boolconst** | **id**

assumendo che:

- un identificatore è una lista (non vuota) di lettere maiuscole, seguita da zero o più cifre;
- una costante intera più lunga di una cifra non può iniziare con 0;
- una costante stringa è una sequenza (anche vuota) di caratteri alfanumerici racchiusa tra doppi apici;
- una costante booleana è **true** o **false**.

Esercizio 3

```
%{
#include <stdlib.h>
#include "def.h"  /* encoding of lexical symbols */
Lexval lexval;    /* union {ival, sval} */
%}
delimiter      [ \t\n]
spacing        {delimiter}+
uppercase      [A-Z]
lowercase      [a-z]
letter         {uppercase} | {lowercase}
digit          [0-9]
initial_digit  [1-9]
alphanum       {letter} | {digit}
intconst       {initial_digit}{digit}* | 0
boolconst      false | true
strconst       \"{alphanum}*\"
id             {uppercase}+{digit}*
sugar          [,;:()\[\]]
%%
{spacing}      ;
{sugar}        {return(yytext[0]);}
matrix         {return(MATRIX);}
int            {return(INT);}
string         {return(STRING);}
bool           {return(BOOL);}
of             {return(OF);}
call           {return(CALL);}
{intconst}     {lexval.ival = atoi(yytext); return(INTCONST);}
{strconst}     {lexval.sval = newstring(yytext); return(STRCONST);}
{boolconst}    {lexval.ival = (yytext[0]=='f' ? 0 : 1); return(BOOLCONST);}
{id}           {lexval.ival = assign_id(yytext); return(ID);}
.              {return(ERROR);}
%%
char *newstring(char *s){
  char *p = malloc(strlen(s)+1);
  strcpy(p,s);
  return(p);}

int assign_id(){...}
```

program → { *stat* **;** }$^{+}$
stat → *def-stat* | *procedure-call*
def-stat → **id** { **,** **id** } **:** *type*
type → **int** | **string** | **bool** | *structured-type*
structured-type → **matrix [intconst** { **,** **intconst** } **] of** *type*
procedure-call → **call id (** [*parameters*] **)**
parameters → *param* { **,** *param* }
param → **intconst** | **stringconst** | **boolconst** | **id**

Esercizio 4

Codificare in Lex l'analizzatore lessicale del linguaggio del seguente frammento di codice:

```
int i = 10, j = -7, k = +55;                    -- counters
string n_of_file2 = "test.log";                 -- file must exist
real x = 10.22, y = -35.06, z = +15.000, w = 0.01;
while i > j do
    alpha(i);
end;
```

assumendo che:

- un commento inizia con '--' e finisce con un newline;
- una costante intera lunga più di una cifra non può iniziare con zero;
- una costante reale è espressa da una parte intera e una parte decimale, entrambe obbligatorie;
- in una costante reale, la parte intera lunga più di una cifra non può iniziare con zero;
- sia la costante intera che quella reale è opzionalmente qualificata con un segno (parte integrante della costante);
- una costante stringa è racchiusa tra doppi apici e può contenere qualsiasi carattere eccetto newline;
- un identificatore inizia con una lettera seguita da una sequenza (anche vuota) di caratteri alfanumerici, eventualmente separati da underscore '_';
- un identificatore non può includere due o più underscore consecutivi e non può terminare con un underscore.

Esercizio 4

```
%{
#include <stdlib.h>
#include "def.h"  /* encoding of lexical symbols */
Lexval lexval;   /* ival, rval, sval */
%}
blank          " "
delimiter      {blank}|\t|\n
spacing        {delimiter}+
comment        "--"(.)*\n
letter         [A-Za-z]
digit          [0-9]
alphanum       {letter}|{digit}
initial_digit  [1-9]
sign            + | -
id             {letter}{alphanum}*(_{alphanum}+)*
intconst       {sign}?({initial_digit} {digit}* | 0)
realconst      {intconst}\.{digit}+
strconst       \"[^\n\"]*\"
sugar          [=;,>()]

%%
```

```
{spacing} ;
{comment}    ;
{sugar}     {return(yytext[0]);}
int         {return(INT);}
string      {return(STRING);}
real        {return(REAL);}
while       {return(BEGIN);}
do          {return(DO);}
end         {return(END);}
{intconst}  {lexval.ival = atoi(yytext); return(INTCONST);}
{realconst} {lexval.rval = atof(yytext); return(REALCONST);}
{strconst}  {lexval.sval = copy(yytext); return(STRCONST);}
{id}        {lexval.ival = assign_id(yytext); return(ID);}
.           {return(ERROR);}

%%

int assign_id(char *s)
{ int line;
  if((line = lookup(s)) == 0) line = insert(s);
  return(line);
}

char *copy(char *s)
{
  char *p = malloc(strlen(s)+1);
  strcpy(p, s);
  return(p);
}
```

Esercizio 5

Codificare in *Lex* un programma che riceve in input un file di testo e stampa le linee (terminate da newline) composte esattamente da tre '**w**' separate tra loro da altri caratteri.

Esercizio 5

Codificare in *Lex* un programma che riceve in input un file di testo e stampa le linee (terminate da newline) composte esattamente da tre '**w**' separate tra loro da altri caratteri.

```
%{
#include <stdio.h>
%}
notw        [^w\n]
line        {notw}*w{notw}+w{notw}+w{notw}*\n
%%
{line}  ECHO;
.       ;
%%
main(){yylex();}
```

Esercizio 6

Codificare in *Lex* un programma completo che riceve in input un file di testo e stampa il contenuto del file, in cui ogni numero intero con segno (sia + che -) è sostituito dallo stesso numero senza segno.

Esercizio 6

Codificare in *Lex* un programma completo che riceve in input un file di testo e stampa il contenuto del file, in cui ogni numero intero con segno (sia + che -) è sostituito dallo stesso numero senza segno.

```
%{
#include <stdio.h>
%}
%option noyywrap
snum [\-+][0-9]+
%%
{snum} {printf("%s",yytext+1);
%%
main(){yylex();}
```

Esercizio 7

Codificare in *Lex* un programma che riceve in input un file di testo e stampa il contenuto del file in cui ogni sequenza di cifre è sostituita da una sequenza di cifre che rappresenta il doppio del numero rappresentato da quella sequenza, come nel seguente esempio:

```
alfa 25 beta 128 fine18  →  alfa 50 beta 256 fine36
```

Esercizio 7

Codificare in *Lex* un programma che riceve in input un file di testo e stampa il contenuto del file in cui ogni sequenza di cifre è sostituita da una sequenza di cifre che rappresenta il doppio del numero rappresentato da quella sequenza, come nel seguente esempio:

`alfa 25 beta 128 fine18` $\rightarrow$ `alfa 50 beta 256 fine36`

```
%{
#include <stdio.h>
%}
%option noyywrap
numero [0-9]+
%%
{numero} {printf("%d", atoi(yytext)*2);}
%%
void main(){ yylex(); }
```