BCSE-III Compiler Design Assignment-3

Using POSIX `lex` Lexical Analyzer Generator

Name: Chirantan Nath
Roll: 101910501064 (Section A3)
Session: 2024-25.

Problem 1

Write a `lex` file to check the validity of a binary message that starts with 1 and ends with 101 and contains any number of dots in between the parts of the messages.

`lex` Code:

```
%{
  /*  Write a lex file to check the validity of a binary message that starts with 1
  and ends with 101 and contains any number of dots in between the parts of the
messages. */
%}
%%

1\.*101\n             {puts("String follows pattern.");}

[Ee][Xx][Ii][Tt]\n    {return 0;}

.+\n                  {puts("String does not follow pattern.");}

%%
inline int yywrap() {return 1;}

int main() {
  puts("Enter strings to be checked in each line. Enter EOF or \'exit\' to exit.");
  yylex();
}
```

Output:

```
chirantan@fedora ~/a/Compiler Design> lex -o p3_1_binary.c p3_1_binary.l
chirantan@fedora ~/a/Compiler Design> cc -o p3_1_binary p3_1_binary.c
chirantan@fedora ~/a/Compiler Design> ./p3_1_binary
Enter strings to be checked in each line. Enter EOF or 'exit' to exit.
1......101
String follows pattern.
1.............101
String follows pattern.
1101
String follows pattern.
11011
String does not follow pattern.
1011
String does not follow pattern.
1ansajfwaf
String does not follow pattern.
exit
chirantan@fedora ~/a/Compiler Design>
```

(continued on next page)

Problem 2

Write a `lex` program to recognize a string that starts with a capital letter which is followed by any small letter or decimal digits and ends with a special character of your choice.[1]

`lex` Code:

```
%{
  /*
  Write a lex program to recognize a string that starts with a capital letter
  which is followed by any small letter or decimal digits and ends with a special
  character of your choice.
  */
%}
exit      [Ee][Xx][Ii][Tt]
LETTER    [A-Z]
letter    [a-z]
digit     [0-9]
special   \$
%%
{LETTER}({letter}|{digit})*{special}\n    {puts("String given follows pattern.");}
{exit}\n                                  {return 0;}
.+\n                                      {puts("String given does not follow
pattern.");}
%%
inline int yywrap() {return 1;}
int main() {
  puts("Enter strings to be checked in each line. Enter EOF or \'exit\' to exit.");
  yylex();
}
```

Output:

```
chirantan@fedora ~/a/Compiler Design> lex -o p3_2.c p3_2.l
chirantan@fedora ~/a/Compiler Design> cc -o p3_2 p3_2.c
chirantan@fedora ~/a/Compiler Design> ./p3_2
Enter strings to be checked in each line. Enter EOF or 'exit' to exit.
Chirantan$
String given follows pattern.
Chirantan123$
String given follows pattern.
Chirantan123*
String given does not follow pattern.
C123$
String given follows pattern.
C$
String given follows pattern.
C$$$$$$
String given does not follow pattern.
exit
chirantan@fedora ~/a/Compiler Design>
```

(continued on next page)

1 The special character *we* chose was the dollar sign (‘$’).

Problem 3

Write a `lex` program that will function as a calculator and perform the operations like addition, subtraction, multiplication, division, modulo and power.

`lex` Code:

```
%{
  /*Write a lex program that will function as a calculator and perform the
   operations like addition, subtraction, multiplication, division, modulo and power.*/
  #include <stdbool.h>
  enum Operation {
    PLUS, MINUS, MUL, DIV, MOD, POW
  } op = PLUS;
  int accumulator = 0;
  bool canTerminate = false;
  void reset() {
    op = PLUS; accumulator = 0;
    canTerminate = false;
  }
  int intpow(int x, int n) {
    int y;
    if(!n) return 1;
    y = 1;
    while(n > 1) {
      if(n & 1) {
        y *= x;
        n--;
      }
      x *= x;
      n >>= 1;
    }
    return x*y;
  }
%}
digitstart                  [1-9]
digit                       [0-9]
plus                        \+
minus                       \-
mul                         \*
div                         \/
mod                         \%
pow                         \^
whitespace                  [ \t\f]
%%

{whitespace}+               {/*Ignore.*/}

{digitstart}{digit}*        {
  int result = atoi(yytext);
  switch(op) {
    case PLUS:
      accumulator += result;
      break;
    case MINUS:
      accumulator -= result;
      break;
    case MUL:
      accumulator *= result;
      break;
    case DIV:
      accumulator /= result;
      break;
    case MOD:
      accumulator %= result;
      break;
```

```
    case POW:
      accumulator = intpow(accumulator, result);
      break;
  }
  canTerminate = true;
  //printf("Recognized number %d, accumulator %d\n", result, accumulator);
}

{plus}                      {op = PLUS; canTerminate = false;}
{minus}                     {op = MINUS; canTerminate = false;}
{mul}                       {op = MUL; canTerminate = false;}
{div}                       {op = DIV; canTerminate = false;}
{mod}                       {op = MOD; canTerminate = false;}
{pow}                       {op = POW; canTerminate = false;}

\n                          {
  //puts("Calculating...");
  if(!canTerminate) {
    fputs("Illegal expression.\n", stderr);
  } else {
    printf("%d\n", accumulator);
  }
  reset();
}

.                           {
  int ch;
  fputs("Illegal character entered. Ignoring line.\n", stderr);
  while((ch = input()) != '\n' && ch != EOF);
  reset();
}
%%
inline int yywrap() {return 1;}
int main() {
  puts("Enter expression on each line. Enter EOF to exit.");
  yylex();
}
```

Output:

```
chirantan@fedora ~/a/Compiler Design> lex -o p3_3_calc.c p3_3_calc.l
chirantan@fedora ~/a/Compiler Design> cc -o p3_3_calc p3_3_calc.c
chirantan@fedora ~/a/Compiler Design> ./p3_3_calc
Enter expression on each line. Enter EOF to exit.
1+3*5
20
2+4
6
1+3
4
3*5
15
3$5
Illegal character entered. Ignoring line.
chirantan@fedora ~/a/Compiler Design>
```