Assignment 1: Solutions

- Page 51:
  - 2.2.1 a and b
    
    (a) a is S \[ (S \rightarrow a) \]
    aa+ is S \[ (S \rightarrow SS+) \]
    aa+a* is S \[ (S \rightarrow SS^*) \]

    (b)
    \[
    \begin{array}{c}
    S \\
    S \\
    S \\
    + \\
    a \\
    a \\
    a \\
    \end{array}
    \]

- 2.2.4 a, b, and c (forget about the part that asks you to show that your grammar is correct)

  (a) expr -> term | expr expr op
      op -> + | - | / | *
      term -> 0|1|2|3|4|5|6|7|8|9

  (b) list -> list, id | id

  (c) list -> id, list | id
expr -> \{\text{print}(\text{"+"})\} \text{expr} + \text{term} | \{\text{print}(\text{"-"})\} \text{expr} - \text{term} | \text{term}
expr -> \{\text{print}(\text{"*"})\} \text{term} * \text{element} | \{\text{print}(\text{"/"})\} \text{term} / \text{element} | \text{element}
\text{element} -> (\text{expr}) | \{\text{print}(\text{"0"})\} 0 | \ldots | \{\text{print}(\text{"9"})\} 9

9-5+2

+952

The second one of 9-5*2 will be done similarly and generate -9*52
2.3.2

expr -> expr {print ('+')} term + | expr {print ('-')} term - | term
term -> term {print ('*')} factor * | term {print ('/')} factor / | factor
factor -> {print('0')} 0 | …. | {print('9')} 9 | {print('(')}expr {print(')')}

95-2*

\[
\begin{array}{c}
\text{expr} \\
\text{term} \\
\text{term} * \\
\text{factor} * \\
\text{factor} \\
\text{2} \\
\text{2} \\
\text{9} \\
\text{9} \\
(9-5)*2
\end{array}
\]

From the above example you can see why we needed to add parentheses to the production rule of factor. Otherwise, the translation scheme will yield 9-5*2 and with the higher precedence of the multiplication it will be 9-(5*2) which is the wrong answer (actually it is the answer of the following one of 952*_-).

The second one 952*_- is done similarly to yield: 9-5*2
See the difference between this one and the 95-2*
In this problem we want to have a translation scheme from postfix to prefix notation.

expr -> expr \{\texttt{print (+)}\} term + | expr \{\texttt{print (-)}\} term - | term

term -> term \{\texttt{print (*)}\} factor * | term \{\texttt{print (/)}\} factor / | factor

factor -> \{\texttt{print(0)}\} 0 | …. | \{\texttt{print(9)}\} 9 | expr

Note that here we don’t have the parentheses in factor, because both prefix and postfix notation do not need precedence rules.

Let’s try it with 95-2*

\[
\begin{array}{c}
\text{expr} \\
\text{term} \\
\text{term} \\
\text{factor} \\
\text{factor} \\
\text{expr} \\
\text{term} \\
\text{term} \\
\text{factor} \\
\text{factor} \\
9 \\
9
\end{array}
\]

This yields *-952 which is the prefix notation of 95-2*
2.4.1

(a) \( S \rightarrow +SS \mid -SS \mid a \)

```c
void S()
{
    switch(lookahead):
    case '+': match('+'); S(); S(); break;
    case '-': match('-'); S(); S(); break;
    case 'a': match(a); break;
    default: report("Error!");
}
```

```c
match(terminal t)
{
    if(lookahead == t) lookahead = next terminal;
    else report("Error!");
}
```

(b) \( S \rightarrow S(S)S \mid \varepsilon \)

The above production has two problems. First it is left recursive, so we need to eliminate that in order to avoid infinite recursion. Second, it is ambiguous. But a quick look at what kind of language the above rule generates makes our life easier! The rule generates parentheses, nested and non-nested, for example: ()()(), ((())), ..., This means that if we write the rule as \( S \rightarrow (S)S \mid \varepsilon \) We are still generating the same language and the rule is no longer left-recursive.

```c
void S()
{
    if( lookahead == '(' )
        match('('); S(); match(')'); S();
}
```

```c
match(terminal t)
{
    if(lookahead == t) lookahead = next terminal;
    else report("Error!");
}
```
(c) \( S \rightarrow 0S1 \mid 01 \)

```c
void S( )
{
    match('0');
    if( lookahead = '0' ) S();
    else if( lookahead = '1' )
    {
        match('1');
        return;
    }
    match('1');
}
```

```c
match(terminal t)
{
    if(lookahead == t) lookahead = next terminal;
    else report("Error!");
}
```