# Brief Guide to Completing Assignment 3

## 1 Simple Arithmetic Expression

The general form of m_exp in stdin:

$num_1$ **h_op$_1$** $num_2$ **h_op$_2$** $\cdots$ $num_{n-1}$ **h_op$_{n-1}$** $num_n$ **end_character**

where **h_op** is * or / and **end_character** can be **l_op** (+ or -) or '\n'.

Evaluation of m_exp:

$(\cdots((num_1$ **h_op$_1$** $num_2)$ **h_op$_2$** $num_3)$ $\cdots$ $num_{n-1})$ **h_op$_{n-1}$** $num_n)$

The general form of s_exp in stdin:

$m\_exp$ **l_op$_1$** $m\_exp_2$ **l_op$_2$** $\cdots$ $m\_exp_{n-1}$ **l_op$_{n-1}$** $m\_exp_n$ **end_character**

where **l_op** is + - and **end_character** is '\n'.

Evaluation of s_exp:

$(\cdots((m\_exp_1$ **l_op$_1$** $m\_exp_2)$ **l_op$_2$** $m\_exp_3)$ $\cdots$ $m\_exp_{n-1})$ **l_op$_{n-1}$** $m\_exp_n)$

Notice that the structure of s_exp and m_exp is exactly the same. This suggests that the two functions s_exp() and m_exp() should be very similar.

An s_exp example: 34 * 6 - 7/ 3* 4 +5
The first m_exp: 34*6
The second m_exp: 7/3*4
The third m_exp: 5

## 2 The m_exp() Function

A simple m_exp example in stdin: (□: space, ↩: return, ⇑: indicate location in stdin where next character will be read)

□ □ 3 4 □ * □ □ 3 □ ↩
⇑

How do we implement m_exp() function to evaluate an m_exp?

```
float m_exp(float sub_exp, char op) {
 1. check if it is the end of m_exp
    how? check op to see if op is '+', '-', or '\n'
 2. if yes, return sub_exp (push back op, why?)
 3. if not
      get next num of m_exp from stdin: save in f1
      get next op of m_exp from stdin: save in op1
      find 'sub_exp op f1': save in f1
      return m_exp(f1, op1)
             (f1 has evaluated one more num
                 then sub_exp of the m_exp!)
             (op1 is next operator of the m_exp!)
}
```

# 3 An Example for m_exp() Function

A simple m_exp example in stdin:

□ □ 3 4 □ * □ □ 3 □ ↩
⇑

We can start by getting first num and first operator
f = get_num() (f=34)

□ □ 3 4 □ * □ □ 3 □ ↩
⇑
ch = get_op() (ch='*')

□ □ 3 4 □ * □ □ 3 □ ↩
⇑

Now call m_exp() function
m_exp(f, ch); (f is 34, ch is '*')
ch is not '\n'
f1=get_num() (f1 is 3)

□ □ 3 4 □ * □ □ 3 □ ↩
⇑
ch1=get_op() (ch1 is '\n')

□ □ 3 4 □ * □ □ 3 □ ↩
⇑
f=f*f1
return m_exp(f, ch1) (f is 102, ch1 is '\n')
ch1 is '\n'
push back ch1

□ □ 3 4 □ * □ □ 3 □ ↩
⇑
return 102

We can also start by calling m_exp() with 1 and '*'

```
m_exp(1,'*');
 □  □  3  4  □  *  □  □  3  □  ↩
 ⇑
 '*' is not '+', '-', or '\'
 get_num() ( 34 )

 □  □  3  4  □  *  □  □  3  □  ↩
             ⇑
 get_op() ( '*' )

 □  □  3  4  □  *  □  □  3  □  ↩
                   ⇑
 1*34
 return m_exp(34, '*')
           '*' is not '+', '-', or '\'
           get_num() ( 3 )

 □  □  3  4  □  *  □  □  3  □  ↩
                            ⇑
           get_op() ( '\n' )

 □  □  3  4  □  *  □  □  3  □  ↩
                                  ⇑
           34*3
           return m_exp(102, '\n')
                     '\n' is the end
                     push back '\n'

 □  □  3  4  □  *  □  □  3  □  ↩
                                  ⇑
                     return 102
```

## 4 The s_exp() Function

The structure of s_exp() is exactly the same as the m_exp() function.

The only difference is that in s_exp() we should use m_exp() function to replace get_num() function.

In order for s_exp() to be correct, m_exp() need to push back +, -, or '\n' since those are operators used in s_exp() to perform proper operation or to terminate.

we can run s_exp() function as s_exp(0, '+');

## 5 Implementation

Write get_num() and get_op() first and then test these functions before staring the other parts.

Write m_exp() function and then test it with sample inputs ending with +, - or '\n' before writing s_exp() function.

Write s_exp() function and then test it.