

CMPT125, Fall 2018

Homework Assignment 4

Due date: November 16, 2018

You need to submit your .c files to CourSys.

In this homework you will write a calculator for arithmetic expressions written in infix and prefix notations.

The assignment will be automatically graded. Please make sure that your functions return the required output. The functions don't need to print anything.

Part 1 [40 points]. Parsing a string and converting ints to strings

Write the file `parse.c` with the the following functions. Include `parse.h` as the header

- 1) Write a function `int count_tokens(char* str)` that gets a string and counts the number of tokens in `str`. A token is any substring separated from others by spaces. Examples:
On input `"ABC) 123 !!"` the function will return 4.
On the input `"(8 + (41 - 12))"` the function will return 9.
**You may assume that tokens are separated by a single space, and there are no spaces neither before the first token not after the last token.
- 2) Write a function `char** get_tokens(char* str, int n_tokens)` that gets a string, and the number of tokens obtained in `count_tokens`, and returns an array of strings, where the *i*th element of the array contains the *i*th token. Example:
On input `str = "(8 + (41 - 12))"` and `n_tokens = 9` the function returns the array `["(", "8", "+", "(", "41", "-", "12", ")", ")"]`.
** You may assume that tokens are separated by a single space, and there are no spaces before the first token, and no spaces after the last token.
- 3) Write a function `int is_operator(char* str)` that gets a string and checks if it is an arithmetic operator, i.e., is of the four operators `"+", "-", "*", "/"`.
- 4) Write a function `int is_number(char* str)` that checks if a given string is a non-negative integer, e.g., `"7"`, `"16"`.
- 5) Write a function that gets a non-negative int and return a string containing this int. You may assume that the number has at most 2 digits, i.e., is between 0 and 99.
`char* num_to_str(unsigned int num)`
* Remember to use `malloc` to create the returned string of the heap.
** When allocating a string, remember to set the last char to `'\0'`.
In order to convert a digit into a char you may use:

```
int digit = 3;
char c = '0' + digit;
```

Part 2 [30 points]. Writing a calculator for infix notation:

Write the file `infix.c` with the the following functions. Include `infix.h` as the header

In this part you will write a calculator for arithmetic expressions with binary operations. We will assume that there are only binary operations, there are parentheses for each operation (including the outermost parentheses), and all tokens are separated by single spaces,

- 6) Write a function that gets an array of tokens representing an arithmetic expression and returns the index of the operator for the outermost operation.

```
int find_operator(char** expression, int n_tokens)
```

Examples:

On input ["(", "8", "+", "(", "41", "-", "12", ")", ")", ")]"

the function will return 2. This is because the expression is : (8 + (41 - 12)) and the outermost operation is +.

- 7) Write a (recursive) function that gets an array of tokens representing an arithmetic expression and evaluates the corresponding expression.

```
int infix_eval_tokens(char** str, int n_tokens)
```

* You may assume that all intermediate values in the computation are between 0 and 99

** You may assume that the results of all division operations are integers.

- 8) Use the functions in part 1 and `eval_tokens` to write a function that gets a string representing an arithmetic expression and evaluates the corresponding expression.

```
int infix_eval(char* expression)
```

* You may assume that all intermediate values in the computation are between 0 and 99

** You may assume that the results of all division operations are integers.

*** You may assume that all expression are formatted correctly.

Examples:

```
"( 25 / 5 )"
```

```
"( 5 + ( 7 - 2 ) )"
```

```
"( ( 10 + 6 ) / ( ( 3 + 7 ) - ( 2 + 4 ) ) )"
```

Part 3 [30 points]. Writing a calculator for prefix notation:

Write the file `prefix.c` with the the following functions. Include `prefix.h` as the header

In this part you will write a calculator for arithmetic expressions in prefix notation.

- 9) Write a function that gets an array of tokens representing an arithmetic expression in the prefix notation, and computes the evaluation of the expression. Use the stack evaluation discussed in class.

```
int prefix_eval_tokens(char** prefix_expr, int n_tokens)
```

Examples:

On input `["+ ", "5", "- ", "7", "2"]` the function will return 10.

This is because the expression is : $(5 + (7 - 2)) = 10$.

* You may assume that all intermediate values in the computation are between 0 and 99

** You may assume that the results of all division operations are integers.

*** You may use the stack implementation given in the assignment. The data in stack is of type `void*`. Use casting if necessary.

- 10) Use the functions in part 1 and `prefix_eval` to write a function that gets a string representing an arithmetic expression in prefix notation and evaluates the corresponding expression.

```
int prefix_eval(char* expression)
```

* You may assume that all intermediate values in the computation are between 0 and 99

** You may assume that the results of all division operations are integers.

*** You may assume that all expression are formatted correctly.

Examples:

```
"/ 25 5"
```

```
"+ 5 - 7 2"
```

```
"- / + 15 13 - 5 1 + 2 4"
```

In parts 2 and 3 you may want to use `#include <string.h>` to access functions that implement string manipulations. For example:

```
int atoi(const char *str)
```

The function gets a string representation of an integer, and returns the number as int.

```
int strncmp(const char *str1, const char *str2, int n)
```

The function gets two strings, and check if they are equal in the first n chars or until reaching a terminating null-character in one of them.

More useful functions can be found here:

https://www.tutorialspoint.com/c_standard_library/string_h.htm