Secure and Dependable Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Module: CO-566
Date: 2021-02-19
Due: 2021-02-26

SADS 2021 Problem Sheet #2

This problem sheet asks you to convert infix expressions into postfix expressions. The emphasis of this assignment is on unit testing and on fuzzying your expression parser using the american fuzzy lop (AFL). This assignment naturally builds on the code you have written for the previous assignment.

Problem 2.1: *infix calculator with unit tests* (2+2+1 = 5 points)

Using the stack implementation from the first problem sheet, you are going to implement an infix expression calculator (`expr`) in C. Here are some example invocations in a shell (the quotes prevent the expansion of the * by the shell).

```
$ expr 42
42
$ expr 2 + 3
5
$ expr 2 '*' 3
6
$ expr '(' 2 + 3 ')' '*' 2
10
$ expr 1 + 2 + '(' 3 + 4 + 5 ')'
15
```

The calculator should implement integer arithmetic and the operators `+`, `-`, `*`, `/`, `%`. It should handle error cases:

```
$ expr foo
expr: invalid token 'foo'
$ expr 2 +
expr: missing operand
$ expr 2 4
expr: missing operator
$ expr 2 / 0
expr: arithmetic error
$ expr '(' 2 + 0
expr: missing closing parenthesis
$ expr 2 + 0 ')'
expr: missing opening parenthesis
```

a) Implement unit tests (ideally before writing the implementation of the calculator) using the API defined in the `infix.h` header file shown at the end of this sheet.
b) Implement the API defined in the `infix.h` header file. Use good coding techniques such as defensive programming.
c) Determine the coverage of your unit test collection.

Problem 2.2: *fuzzying your calculator* (1+2+2 = 5 points)

a) Compile your source code using the AFL fuzzer compiler.
b) Run the AFL fuzzer and document test cases found by the fuzzer.
c) Fix the bugs found by the initial run of the AFL fuzzer until it runs for a “longer” time without finding additional crashes or hangs.

Submit your source code together with your american fuzzy lop testing documentation as a zip file (remove all build files before creating the archive).

```c
/*
 * expr/src/infix.h --
 */

/** @file */

#ifndef _INFIX_H
#define _INFIX_H

/**
 * \brief Evaluate an expression in infix notation.
 *
 * This functions takes an array of strings that are interpreted as
 * tokens of an expression in infix notation. The function returns the
 * results as a string via the second argument. The result is
 * allocated using malloc() and must be freed by the caller of this
 * function. Note that the function may return a NULL pointer if no
 * result was calculated. The return value of the function indicates
 * whether the evaluation of the expression in reverse polish notation
 * was successful or there were any errors.
 *
 * \param token Tokens of an expression (NULL-terminated array).
 * \param result Pointer to the string holding the result (malloced).
 * \result One of the error codes defined above.
 */

int expr_infix_eval_token(char *token[], char **result);

/**
 * \brief Evaluate an expression in infix notation.
 *
 * This function splits the expression contained in the string expr
 * into an array of strings and then interpretes the array as tokens
 * of an expression in infix notation. The function returns
 * the results as a string via the second argument. The result is
 * allocated using malloc() and must be freed by the caller of this
 * function. Note that the function may return a NULL pointer if no
 * result was calculated. The return value of the function indicates
 * whether the evaluation of the expression in reverse polish notation
 * was successful or there were any errors.
 *
 * \param expr The expression (whitespace separated numbers and operators)
 * \param result Pointer to the string holding the result (malloced).
 * \result One of the error codes defined above.
 */

int expr_infix_eval(const char *expr, char **result);

#endif
```