Secure and Dependable Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CO21-320203
Date: 2018-02-19
Due: 2018-02-26

SADS 2018 Problem Sheet #1

This problem sheet asks you to implement a simple reverse polish notation calculator in C. The emphasis of this assignment is on the unit testing and the test coverage calculation. The header files provided below give some guidance how your code should be structured. (Do not copy a random implementation from the Internet, you will not receive any points if the implementation does not follow the guidelines given here and there are no proper unit tests.)

You have to submit your source code together with the test coverage report in a zip file (please remove all build files before creating the archive). You are encouraged to use `cmake` and expected to use the `check`[1] unit testing framework for C.

Problem 1.1: *stack implementation with unit tests* (5 points)

Implement a generic stack in C. Write unit test cases for your stack first before writing the stack implementation. Use the `gcc` coverage testing features to record the coverage of your test cases. (You may use the `gcd` C project discussed in class as a template.)

A generic stack is defined by the following `stack.h` header file:

```
/*
 * stack.h --
 */

#ifndef _STACK_H
#define _STACK_H

typedef struct _stack stack_t;

/**
 * \brief Create a new empty stack.
 *
 * \return A pointer acting as a handle for the new stack.
 */

stack_t*
stack_new();

/**
 * \brief Push data (allocated by the caller) on the stack.
 *
 * \param s The stack to push data on.
 * \param data The pointer to data to be pushed on the stack.
 */

void
stack_push(stack_t *s, void *data);

/**
 * \brief Pop data from the top of the stack.
 *
 * \param s The stack to pop data from.
 * \return The data or NULL if the stack is empty.
 */
```

[1] https://libcheck.github.io/check/

```
void*
stack_pop(stack_t *s);

/**
 * \brief Peek on the data at the top of the stack.
 *
 * \param s The stack to peek on.
 * \return The data or NULL if the stack is empty.
 */

void*
stack_peek(stack_t *s);

/**
 * \brief Test whether a stack is empty.
 *
 * \param s The stack to test.
 * \return A non-zero value if the stack is empty, 0 otherwise.
 */

int
stack_empty(stack_t *s);

/**
 * \brief Delete a stack.
 *
 * \param s The stack to delete.
 */

void
stack_del(stack_t *s);

#endif
```

Problem 1.2: *reverse polish notation calculator with unit tests* (5 points)

Using the stack implementation from the first problem, implement a reverse polish calculator (`rpnc`) in C. Here are some example invocation in a shell:

```
$ rpnc 42
42
$ rpnc 2 3 +
5
$ rpnc 2 3 '*'
6
$ rpnc 2 3 + 2 '*'
10
```

The calculator should handle error cases:

```
$ rpnc foo
rpnc: invalid token 'foo'
$ rpnc 2 +
rpnc: missing operand
$ rpnc 2 4
rpnc: missing operator
```

To enable unit testing of the calculator, implement the API defined in the following `rpn.h` header file.

```c
/*
 * rpn.h --
 */

#ifndef _RPN_H
#define _RPN_H

#define RPN_OK                  0
#define RPN_INVALID_TOKEN       -1
#define RPN_MISSING_OPERAND     -2
#define RPN_MISSING_OPERATOR    -3

/**
 * \brief Evaluate an expression in reverse polish notation.
 *
 * \param token The vector of tokens making up the expression.
 * \param result Pointer to the string which will hold the result (malloced).
 * \result One of the error codes defined above.
 */

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

#endif
```

Provide unit tests and determine the coverage of your unit test collection.