Mico's Toy RPN Calculator

User and Developer Manual

Generated by ChatGPT from combined source

April 30, 2025

Contents

1. Overview

This document provides a detailed description of the core functionality implemented in Mico's Toy RPN Calculator. It is intended for both users and developers.

The calculator operates on a fixed-size stack and supports real and complex numbers, strings, real and complex matrices, and various mathematical operations including polynomial evaluation and root-finding.

2. Data Types and Stack Model

2.1. StackElement. The calculator uses a tagged union ‘StackElement‘ to represent elements of different types:

- `TYPE_REAL` – IEEE 754 double
- `TYPE_COMPLEX` – C11 `double complex`
- `TYPE_STRING` – dynamically allocated C string
- `TYPE_MATRIX_REAL` – pointer to `gsl_matrix`
- `TYPE_MATRIX_COMPLEX` – pointer to `gsl_matrix_complex`

The stack is a fixed-size array of these elements, managed with push/pop logic.

3. Register System

Registers are a fixed array of slots, each capable of holding a full `StackElement`. Each register has a boolean flag indicating whether it is occupied.

```c
typedef struct {
  StackElement value;
  bool occupied;
} Register;
```

You can store to a register using two stack elements: the value to store and the register index (real number). You can recall by pushing a register index and calling recall.

4. Polynomial Support

4.1. poly_eval. Evaluates a polynomial at a given point. The point and coefficient matrix must be on the stack.

Stack Before:

```
[coeff_matrix, x]
```

Stack After:

```
[result]
```

Uses Horner's method with support for real or complex x and coefficients.

4.2. poly_roots. Computes the roots of a real-coefficient polynomial.

Stack Before:

```
[coeff_matrix]
```

Stack After:

```
[complex_root_matrix]
```

Uses `gsl_poly_complex_solve`. Complex coefficients are not yet supported here.

5. Matrix Operations

This section can be expanded to include:

- Creation (zeros, ones, random)
- Reshape
- Row/column means, sum, variance

6. Cleanup

The following are essential cleanup helpers:

- `free_element()` to release matrix/string memory
- `free_all_registers()` on exit

7. Stack Operations

The calculator operates on a fixed-size stack implemented as an array of tagged union elements. Each operation follows strict type rules and stack depth checks.

7.1. Data Structures. **StackElement:** tagged union holding real, complex, string, or matrix data.

```c
typedef struct {
  ValueType type;
  union {
    double real;
    double complex complex_val;
    char* string;
    gsl_matrix* matrix_real;
    gsl_matrix_complex* matrix_complex;
  };
} StackElement;
```

Stack: holds a fixed-size array and a top-of-stack index.

```
typedef struct {
  StackElement items[STACK_SIZE];
  int top;
} Stack;
```

7.2. Core Functions.

init_stack(Stack*) Initializes the stack by setting `top = -1`.

push_real / push_complex / push_string / push_matrix_real / push_matrix_complex Pushes an element of the given type onto the stack. These functions allocate memory as needed (e.g., for matrices or strings).

pop(Stack*) Removes the top element of the stack and returns it. Caller is responsible for managing memory if needed.

peek(Stack*) Returns the top element without popping it.

stack_top_type(Stack*) Returns the `ValueType` of the top element.

is_stack_empty(Stack*) Returns true if the stack is empty.

print_stack(Stack*) Utility function to print all stack elements from bottom to top, useful for debugging or interactive use.

7.3. Safety Checks. Each push/pop function includes checks to prevent:

- Stack overflow (`top >= STACK_SIZE - 1`)
- Stack underflow (`top < 0`)
- Invalid memory reuse (e.g., freeing popped strings or matrices)

7.4. Example Usage.

```
push_real(&stack, 3.14);
push_real(&stack, 2.71);
StackElement x = pop(&stack); // 2.71
StackElement y = peek(&stack); // 3.14
```

8. Math and Matrix Functions

This section describes advanced mathematical operations and matrix support in the calculator. Matrix creation, randomization, inversion, and statistical reduction are supported.

8.1. Matrix Creation Functions.

make_matrix_of_zeros Creates an $n \times m$ matrix filled with zeros. **Stack input:** `[rows, cols]` (real numbers) **Stack output:** `[zero_matrix]`

make_matrix_of_ones Similar to zeros but fills all entries with 1.0. **Stack input:** `[rows, cols]` **Stack output:** `[one_matrix]`

make_random_matrix Creates a matrix with uniformly random elements in $[0, 1]$. **Stack input:** `[rows, cols]` **Stack output:** `[random_matrix]`

make_identity_matrix Creates a square identity matrix. **Stack input:** `[n]` **Stack output:** `[identity_matrix]`

8.2. Matrix Dimension Queries.

get_matrix_dimensions Extracts the dimensions of a matrix without popping it. **Stack input:** `[matrix]` **Stack output:** `[rows, cols]`

8.3. Reduction Operations.

matrix_means Reduces a matrix by computing row or column means, sums, or variances. **Parameters:** axis (`"row"` or `"col"`), op (`"mean"`, `"sum"`, `"var"`) **Stack input:** `[matrix]` **Stack output:** `[reduced_matrix]`

8.4. Matrix Reshaping.

reshape_matrix Changes a matrix's shape, ensuring element count remains the same. **Stack input:** `[new_rows, new_cols, matrix]` **Stack output:** `[reshaped_matrix]`

8.5. Register Interactions.

store_to_register / recall_from_register Enable saving and retrieving full stack elements. **store input:** `[value, reg_index]` **recall input:** `[reg_index]`

8.6. Memory Management.

free_all_registers Frees any dynamically allocated content stored in registers. Call this at shutdown to avoid memory leaks.

8.7. Serialization.

save_registers_to_file, load_registers_from_file Save and load full register state to/from disk in a readable tagged format. Supports `REAL`, `COMPLEX`, `STRING`, `MATRIX_REAL`, `MATRIX_COMPLEX`.

9. Statistical Functions

This section documents the statistical and probabilistic capabilities of the calculator, built on top of the GNU Scientific Library (GSL). It includes cumulative distributions, quantiles, and standard probability density functions (PDFs) for normal distributions.

9.1. Normal Distribution Functions.

npdf Computes the normal probability density function (PDF) for a given real input x with mean μ and standard deviation σ.
Stack input: `[x, mean, stddev]` **Stack output:** `[pdf]`

ncdf Computes the cumulative distribution function (CDF) of the standard normal distribution.
Stack input: `[x, mean, stddev]` **Stack output:** `[cdf]`

nquant Computes the quantile (inverse CDF) for a given probability value.
Stack input: `[p, mean, stddev]` **Stack output:** `[x]`

9.2. Vector and Matrix Statistics.

matrix_means Computes the mean along rows or columns.

matrix_reduce Generalizes matrix statistics: `sum`, `mean`, or `var`, with axis control.
Parameters:

- Axis: `"row"` or `"col"`
- Operation: `"sum"`, `"mean"`, `"var"`

Stack input: `[matrix]` **Stack output:** `[reduction_vector]`

9.3. Behavior and Compatibility. All functions handle both real and complex inputs where applicable. Variance is computed with the unbiased estimator $(n-1)$ in the denominator.

Complex matrix functions output real-valued variances (via squared modulus).

9.4. Example.

```
# Push a 3x2 real matrix
[3 2 $ 1 2 3 4 5 6]

# Compute row means
"row" "mean" matrix_reduce

# Compute column variance
"col" "var" matrix_reduce
```

10. String Functions

The calculator supports a variety of string operations. Strings are managed as dynamically allocated `char*` objects and are always null-terminated.

10.1. Supported Operations.

concatenate Concatenates the top two strings on the stack.
Stack input: `[str1, str2]` **Stack output:** `[str1 ++ str2]`

str_to_upper Converts the top string on the stack to uppercase.
Stack input: `[str]` **Stack output:** `[STR]`

str_to_lower Converts the top string on the stack to lowercase.
Stack input: `[str]` **Stack output:** `[str]`

str_reverse Reverses the characters of the string on the top of the stack.
Stack input: `[str]` **Stack output:** `[rts]`

str_length Pushes the length (as a real number) of the top string.
Stack input: `[str]` **Stack output:** `[length]`

10.2. Memory Management. All new strings are allocated via `malloc()` and must be `free()`d when popped off the stack. Existing strings being overwritten or combined are also freed as needed to avoid leaks.

10.3. Example Usage.

```
"hello" "world" concatenate     // "helloworld"
"AbC" str_to_lower              // "abc"
"banana" str_reverse           // "ananab"
"ABCdef" str_to_upper          // "ABCDEF"
"micocalc" str_length          // 8
```

11. Evaluation and Utility Functions

This section covers general-purpose utility functions used internally by the RPN calculator to support parsing, evaluation, and safe numerical operations.

11.1. Parser Helpers.

read_complex Parses a complex number of the form `(re,im)` from a string and stores it as a `double complex`.
Input: C string **Output:** Success flag, complex value through pointer

11.2. Unary Real and Complex Operations.

negate_real / negate_complex Returns the negated version of a real or complex number.

one_over_real / one_over_complex Computes the reciprocal: $\frac{1}{x}$, real or complex.

11.3. Safe and Defensive Math. These functions include checks for division by zero and may be used in expression evaluation.

safe_divide_real Returns a/b, checking for division by zero. Returns 0 and prints error on division by zero.

safe_divide_complex Analogous version for `double complex` division.

11.4. Randomization Support. The file includes initialization of a GSL random number generator, shared across matrix and statistical functions:

```
gsl_rng* rng;
gsl_rng_env_setup();
rng = gsl_rng_alloc(gsl_rng_default);
```

11.5. Evaluation Logic. The main expression evaluation functions dispatch based on token type and stack state. They interact with the lexer, stack, and register subsystems.

eval_expression / eval_line Evaluate a single token or full input line. Handles all types of operations, dispatching to:

- Arithmetic
- Stack manipulation
- Function calls
- Register access

- Matrix logic
- Polynomial handling

init_calculator Main calculator initialization routine. Sets up the stack, registers, RNG, and history system.

cleanup_calculator Frees stack and register memory and shuts down RNG subsystem.

11.6. Example Session Flow.

```
init_calculator();
eval_line("3 4 +");            // evaluates 3 + 4
eval_line("dup inv");          // duplicate and take reciprocal
eval_line("store 0");          // store result in register 0
cleanup_calculator();
```