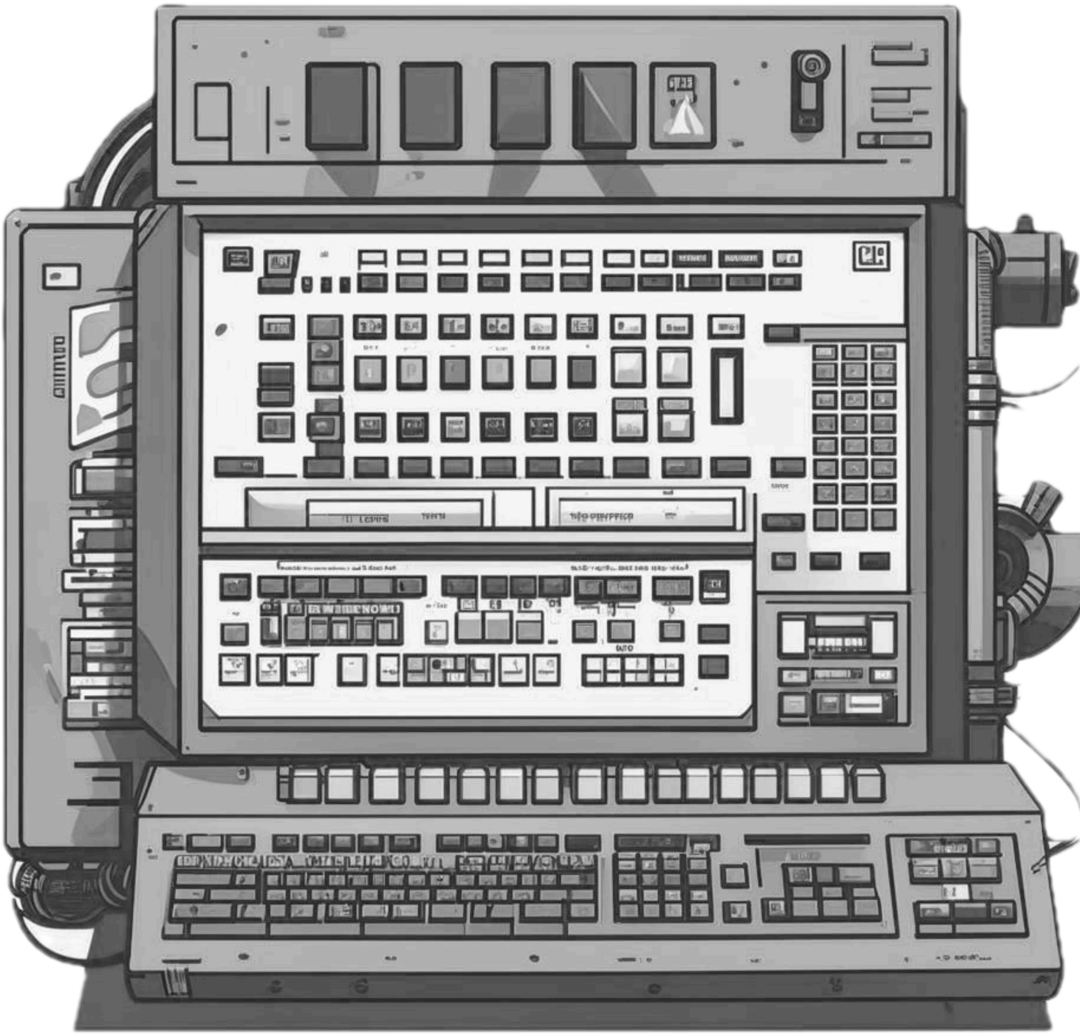

Composed by Vladimir Ulogov

BUND standard library reference

This book serves as a reference guide for the BUND functions (or “words”) defined in standard library.

Referencing 11 functions.

I want to thank my first teacher, who imparted the knowledge and guidance necessary to develop my first programs for the PDP-11 computer.

Introduction

I will introduce a new concatenative programming language called BUND in this work. What is a concatenative language, and how does it differ from the programming languages you're likely familiar with? You're likely acquainted with applicative programming languages like Python, C, or Java. Alternatively, you may have discovered functional programming languages such as Lisp, Haskell, or ML, other examples of applicative programming languages. This category is defined by the way functions are viewed and handled. In applicative languages, a function is treated as a mathematical primitive that computes based on passed arguments and returns a value. In contrast, concatenative programming languages pass a data context from one function to another, external to the function itself. While the stack is the most common method for passing such context, there are concatenative languages that don't utilize a stack. Passing data context enables the concatenation of data processing. Concatenative languages are less known in the software development communities, but you might have heard of languages such as Forth, PostScript, and Factor.

The stack is utilized in many but not all concatenative languages, while applicative languages often use stack structures internally to aid computation. Stacks are indispensable for recursive computation, passing return values computed by functions and storing references to an execution context. What distinguishes concatenative stack-based languages from applicative counterparts is the use of the stack for input data, computational context, and result storage. In essence, everything in concatenative stack-based languages is stored in the stack. In some cases, computational instructions are also stored alongside data on the stack. Since everything, including the context for functions, is stored on the stack, functions in concatenative stack-based languages do not have conventional arguments. Although they function as such, they are often referred to as “words,” as was defined in one of the first concatenative languages to gain popularity - Forth. Another characteristic of concatenative stack-based languages is their reliance on the stack’s Last In, First Out (LIFO) nature. They often employ Reverse Polish Notation (RPN).

So, what will might surprise you in concatenative stack-based language?

- We already mentioned that the functions do not have arguments and no dedicated return value. All input and output data passed to and from the function are passed through the stack.
- You are responsible for ensuring the correct order of the values passed in the data context to the function, as this context is on the stack.
- You are also responsible for interpreting return data placed on the stack. Unlike in the functional language paradigm, there could be more than one return value, depending on your function (or “word”).
- There are no variables. All data are stored on the stack.
- There are no global constants, variables, or values. Everything is on the stack.
- Due to the LIFO nature of the stack, you will deal with RPN, although BUND offers you an ability to creae a stack with FIFO policy.

BUND Standard library reference

Although language design is often simple, elegant, and thoughtfully executed, there is room for greater practicality. A language's core becomes truly functional and valuable to developers only when accompanied by a standard library of useful functions. These functions provide essential tools for performing operations and manipulating data effectively. Moreover, BUND shares several characteristics with other concatenative languages.

> **!** **Memorize**
>
> All run-time functionality of the BUND implemented in standard library.

When I say "all," I mean that every aspect of the functionality extends beyond just implementing the BUND parser and core logic. The BUND standard library is situated across multiple locations. Although this may initially be a design flaw, I had deliberate reasons for structuring the standard library this way.

Info

- [ ] The Rust crate *rust_multistack* encompasses all the logic associated with stack operations. Additionally, it incorporates elements of the standard library that pertain specifically to these operations. Features include data swapping on the stack, data duplication, removal of data, stack rotation, creation of stacks, and other related functionalities.
- [ ] The Rust crate *rust_multistackvm* is a foundational implementation of the BUND virtual machine. Although different tools and interpreters may facilitate access to BUND, the core logic of the BUND language remains intact within this crate. This encompasses data manipulation and conversion, application logic, mathematical operations, lambda function processing, and all other essential features.
- [ ] The Bund runtime serves as the interpreter for the Bund programming language. It encompasses implementing all standard library functions and facilitating user-related features and controls.

Clear current stack - *clear*

⚡ **Danger**

This is destructive operation with stack.

Defined in

- [x] rust_multistack
- [ ] rust_multistackvm
- [ ] bund runtime

This function clears out all data from the current stack

1: **function** CLEAR()
2: ▷ Clearing all values in current stack
3: Name ← VM::current_stack_name()
4: **if** Name = None **then**
5: **return** Error("Error getting current stack name")
6: CLEAR_STACK(Name)

```
//
// Clearing current stack
//
clear
```

Clear named stack - *clear_in*

> ⚡ **Danger**
>
> This is destructive operation with stack.

Defined in

- [x] rust_multistack
- [ ] rust_multistackvm
- [ ] bund runtime

This function clears out all data from the named stack, taking the name of the stack from current stack

function CLEAR_IN()
    ▷ Clearing named stack
    Name ← *current stack*
    if Name = None **then**
        return Error("Stack is too shallow")
    CLEAR_STACK(Name)

```
//
// Remove all data from named stack
//
:StackName clear_in
```

Return name of current stack to current stack - *current*

Defined in

- [x] rust_multistack
- [ ] rust_multistackvm
- [ ] bund runtime

Returns the name of current stack to current stack

```
function CURRENT()
    ▷ Returns the name of current stack to stack
    Name ← VM::current_stack_name()
    if Name = None then
        return Error("Can not detect current stack name")
    current stack ← Name
```

```
//
// Prints the name of current stack
//
current println
```

Drop element from the stack - *drop*

⚡ **Danger**

This is destructive operation with data.

Defined in

- [x] rust_multistack
- [ ] rust_multistackvm
- [ ] bund runtime

This function takes a single value from the top of current stack and discards it.

```
function DROP()
    ▷ Dropping value that is on top of the stack
    Name ← VM::current_stack_name()
    if Value = None then
        return Error("Stack is too shallow")
    DROP(Name)
```

```
//
// Calling this function will remove
// and discard a value
//
42 drop
```

Drop the last value in the named stack - *drop_in*

⚡ **Danger**

This is destructive operation with data.

Defined in

- [x] rust_multistack
- [ ] rust_multistackvm
- [ ] bund runtime

Drop the last value in the named stack

```
function DROP_IN()
    ▷ Drop the value in the named stack
    Name ← VM::current_stack_name()
    if Name = None then
        return Error("Stack is too shallow")
    DROP(Name)
```

```
//
// Drop the last value from stack "A"
//
"A" drop_in
```

Remove stack with all data - *drop_stack*

> ⚡ **Danger**
>
> This is destructive operation with stack.

Defined in

- [x] rust_multistack
- [ ] rust_multistackvm
- [ ] bund runtime

Drop named stack.

function DROP_STACK()
  ▷ Drop the stack
  Name ← *current stack*
  if Value = None **then**
    return Error("Stack is too shallow")
  DROPSTACK(Name)

```
//
// Drop stack "TheStack"
//
:TheStack drop_stack
// Now stack _TheStack_ doesn't exists
```

Duplicate multiple values in current stack - *dup_many*

Defined in

- [x] rust_multistack
- [ ] rust_multistackvm
- [ ] bund runtime

Duplicate multiple values in current stack

```
function DUP_MANY()
    ▷ Duplicate multiple values
    N ← current stack
    Name ← VM::current_stack_name()
    while N >= 0 do
        Value ← current stack
        current stack ← Call("Dup", [N, Name, Value])
        N ← N - 1
```

```
//
// Duplicate data in stack
//
42 41 2 dup_many
// Now we have 42, 42, 41, 41 in stack
```

Duplicate single value in current stack - *dup_one*

Defined in

- [x] rust_multistack
- [ ] rust_multistackvm
- [ ] bund runtime

Duplicate single value in current stack

function DUP_ONE()
  ▷ Duplicate value
  Value ← *current stack*
  Name ← VM::current_stack_name()
  current stack ← Call("Dup", [1, Name, Value])

```
//
// Duplicate data in stack
//
42 dup_one
// Now we have 42, 42 in stack
```

Duplicate single value in named stack - *dup_one_in*

Defined in

- [ ] rust_multistack
- [ ] rust_multistackvm
- [ ] bund runtime

Description of function

Duplicate single value in the named stack, when name of the stack is reading from current stack

function DUP_ONE_IN()
    ▷ Duplicate value
    Name ← *current stack*
    Value ← *current stack*
    stack Name ← Call("Dup", [1, Name, Value])

```
//
// Duplicating single value from stack "A"
//
@A 42
@main
  :A dup_one_in
// Now in stack A we have 42, 42
```

Make existing stack with name - current - *to_current*

Defined in

- [x] rust_multistack
- [ ] rust_multistackvm
- [ ] bund runtime

Taking name of the stack from the stack and makes this stack a current stack. If stack doesn't exists raise an error

1: **function** TO_CURRENT()
2: ▷ Make already existing stack a current stack
3: Value ← *current stack*
4: **if** Value = None **then**
5: **return** Error("Stack is too shallow")
6: TO_CURRENT(Name)

```
//
// Make stack with name "A" - current
// stack must already exists
//
"A" to_current
```

Make stack with name - current - *to_stack*

Defined in

- [x] rust_multistack
- [ ] rust_multistackvm
- [ ] bund runtime

Taking name of the stack from the stack and makes this stack a current stack. If stack doesn't exists VM creates it.

```
function TO_CURRENT()
    ▷ Make stack a current stack. Create if not existing.
    Name ← current stack
    if Name = None then
        return Error("Stack is too shallow")
    if not VM::stack_exists(Name) then
        ENSURE_STACK(Name)
    TO_CURRENT(Name)
```

```
//
// Make stack with name "A" - current
//
"A" to_stack
```

Conclusion

BUND is a very new language. It is currently in its early stages of development, and the language's runtime has many limitations. The standard library requires improvement, and the author or contributor must address several potential bugs. However, the *bundcore* crate and its dependencies have successfully passed all their test cases, which is a promising sign. Although the language is simple and its underlying dependencies are generally stable, there are no guarantees against critical bugs. The license is attached for reference. While concatenative, stack-based programming languages are not widely used in general programming practices, they have stood the test of time and deserve more attention from the software development community. BUND aims to address design gaps in this concept, and the author hopes to spark interest with his ideas and inspirations that brought BUND into existence.

You can get in touch with my via my LinkedIn profile.
The BUND project is hosted on my GitHub page vulogov

License

Apache License Version 2.0, January 2004 http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

“License” shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

“Licensor” shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

“Legal Entity” shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, “control” means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

“You” (or “Your”) shall mean an individual or Legal Entity exercising permissions granted by this License.

“Source” form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

“Object” form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of

this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of

this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the

Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a “NOTICE” text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,

any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade

names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or

agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,

whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing

the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Content