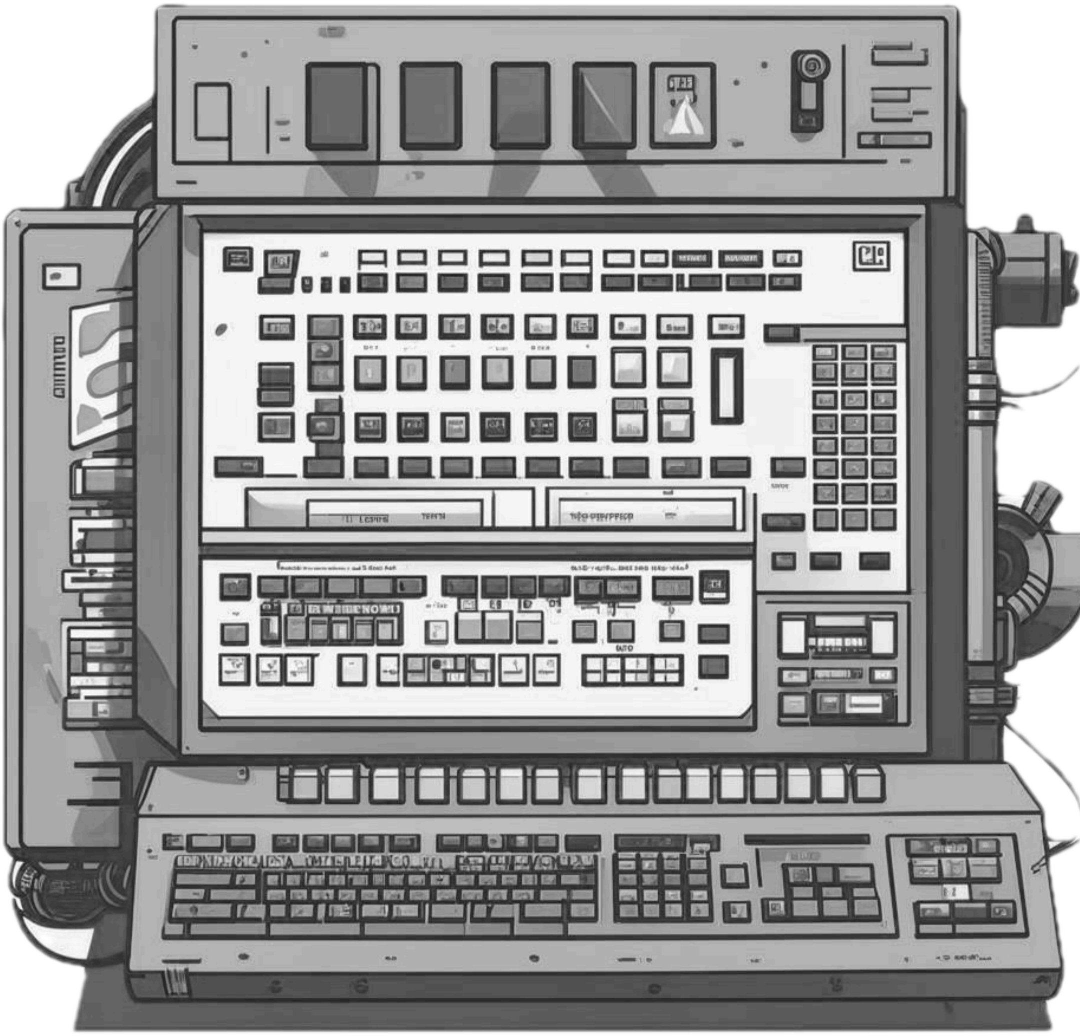


Composed by Vladimir Ulogov

The art of stack operations

This book is a part of the BUND language programming series and introduces the principles of stack operations.

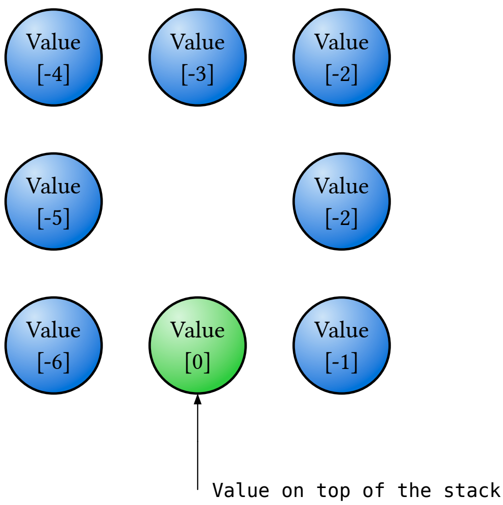
I want to thank my first teacher, who imparted the knowledge and guidance necessary to develop my first programs for the PDP-11 computer.

Introduction

The BUND programming language is a member of the concatenative language family. A notable characteristic of concatenative languages is the presence of a computational context external to the code itself. All computations carried out by the functions, referred to as “words” in concatenative language terminology, are performed over this external context. This differs from the concepts commonly encountered in applicative languages, where function parameters are part of the function context. The computational context is typically structured as a Last In, First Out (LIFO) stack in concatenative languages. However, BUND distinguishes itself from most concatenative languages by having a more sophisticated concept of the computational context.

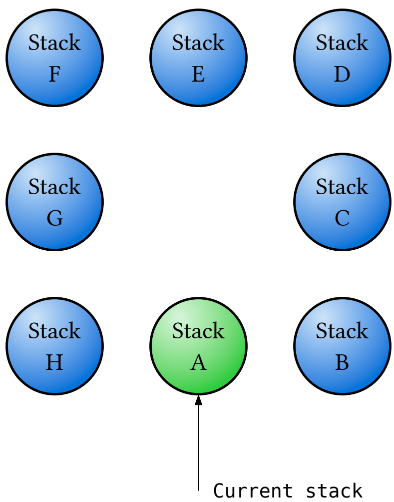
Circular data stack

Instead of using simple LIFO stacks, BUND stores data in multiple named circular buffers, also known as stacks. When you push data to the stack, the circular buffer expands, and when you pull or consume data from the stack, the buffer contracts. While the data buffer is circular, there is always a pointer that refers to the value located on top of the stack. Although you can rotate the buffer in the left or right direction, data is consumed in a single direction only.



Stack-of-stacks references

The next level of abstraction is a circular stack that refers to named data stacks while functioning just like a standard data stack in all other aspects. The stack referred to by the "top of the stack" reference is considered the "current stack," and all operations are by default performed within this data context. When creating a new stack, the reference moves to the top of the stack. When positioning a named stack to become the current stack, the buffer rotates to bring the required stack to the proper position at the "top of the stack."



Workbench

The workbench, an integral component of the BUND virtual machine, is a circular stack that temporarily holds and transfers values between computations conducted in various data contexts. Despite its functional significance, this circular stack does not carry a specific name.

Show me the code !

The "Hello World!" program is often the initial program created in any programming language. It aims to display the "Hello World!" message to the standard output. This example illustrates the stack-based nature of BUND. Initially, a string containing the message is placed on the stack. Subsequently, a function is invoked to retrieve the single element from the stack and print it to the standard output (STDOUT).

Bund

```
//
// This is faimous HelloWorld program written in Bund
//
"Hello world!" println
```

Pushing data to the stack

Like other concatenative languages, BUND does not have specific operations for storing data in the computational context or data stack. Defining a data item in your application's source code instructs the BUND virtual machine to push that value to the top of the current stack. BUND is a dynamically typed language, and this feature provides several properties in the design of the virtual machine that performs actual computations.

- When pushing an item to the stack in BUND, there's no need to specify the data type, as BUND will intelligently determine the actual type of the data.
- It's important to note that data types in BUND are static. Changing the data type is impossible once you define a data item with a specific data type. However, BUND provides conversion functions (words) for converting data between different data types.
- In BUND, data types can be categorized as atomic or container. Atomic data types can only hold a single data type, such as numeric, string, or boolean. On the other hand, container data types can have other atomic or container data items. Examples of container types include lists, dictionaries, lambdas, and pairs.
- BUND is a dynamically typed language whose functions, or "words," can detect data types and perform operations accordingly. For example, the "+" or "add" function can seamlessly handle various numeric and non-numeric data types to produce the most optimal outcome. Nevertheless, this feature does not exempt the BUND language from errors associated with dynamic typing. Therefore, programmers must be mindful of this design decision and exercise caution when dealing with dynamic typing.
- The distinction between data and function in BUND is relatively thin due to its metaprogramming feature. If you decide to utilize metapro-

gramming in your BUND application, please exercise caution and ensure you are familiar with this aspect of the language beforehand.

Numeric data types

There are two types of numeric data - INTEGER and FLOAT. Both of them internally represented by 64-bit signed integers or floats respectfully.

```Bund
//
// Pushing two FLOAT values to the stack
// One is in conventional format another
// is in scientific format
//
3.14 +2e100
```

You can convert values to INTEGER or to FLOAT by using *convert.to_int* or *convert.to_float* respectfully

```Bund
//
// Converting integer value 42 to float
// and pushing it to the stack
//
42 convert.to_float
```

String data type

There are three distinct ways to declare string values, all of which result in the creation of an atomic STRING value:

- Regular string: a set of UNICODE characters enclosed between double quotes.
- Literal string: a set of UNICODE characters enclosed between single quotes.
- Atom string: a set of UNICODE characters, excluding spaces or newlines, preceded by a colon.

```Bund
//
// Example of the regular string
//
"This is a string"
```

Literal string is designed for better handling of the formatting notation

```Bund
//
// Example of the literal string
//
'Это литерал'
```

And Atoms are great for the metaprogramming and definition of the string values where is no spaces.

```Bund
//
// Example of the atom
//
:This_is_atom
```

Library function *convert.to_string* will try to convert any value to it's string representation

Bund

```
//
// Converting float value to string representation
//
42.0 convert.to_string
```

Boolean data type

Boolean data type is an atomic data type internally represented by BOOLEAN value that can take TRUE or FALSE values.

```Bund
//
// Pushing TRUE value to the stack
//
true
```

You can convert non-boolean values to Boolean data types by using *convert.to_bool* function.

```Bund
//
// Converting string value "false" to bool value
// and pushing it to the stack
//
:FALSE convert.to_bool
```

Pointer data type

In the context of metaprogramming in BUND, a function pointer plays a vital role. It is declared by prefixing the function name with a backtick and placing a PTR data object on the stack. This enables you to execute the function the pointer references later after declaration. The PTR object can be handled similarly to any other data value.

```
//
// Example of use PTR object
//
"world!" `println // Here we placing two data objects
                  // On the stack. One is string
                  // Another one is a pointer to
                  // println function
"Hello " print    // Taking single element
                  // from the stack and printing it
                  // and this will be string with
                  // "Hello " string
// Now, we have a PTR object on top of the stack
!                 // And we are executing it
// The outcome is "Hello world!" is printed on terminal
```

You can also create PTR object dynamically by taking STRING value from the stack with help of *ptr* function

```
//
// Example of creating PTR object
//
"Hello world" :println ptr !
```

List data type

A list is an example of a container data type. As previously mentioned, unlike atomic data types, container data types act as containers for holding other container and nuclear data types. A list is a sequential vector that has a collection of data values. You can define a list by declaring the values between square brackets. Due to the dynamically typed nature of the BUND programming language, you do not need to declare the types of data that a list can hold. It can have any data supported by BUND.

```Bund
//
// Here is an example of declaration of LIST value
// in the BUND programming language
//
[
  42              // First element in the list
                  // is an INTEGER
  "Hello world!"  // then a string
  [ 1.0 2.0 3.0 ] // then a LIST
]
```

Lambda data type

A lambda function is anonymous, meaning it does not have a name. The data value containing instructions that comprise the function's body can be stored on the stack and plays a vital role in BUND metaprogramming. Even though lambdas are anonymous and ephemeral by nature, you can register them to turn the lambda function into a named function. Named functions are global and not tied to a particular data context. You can declare a lambda function by specifying data and execution instructions between curly brackets.

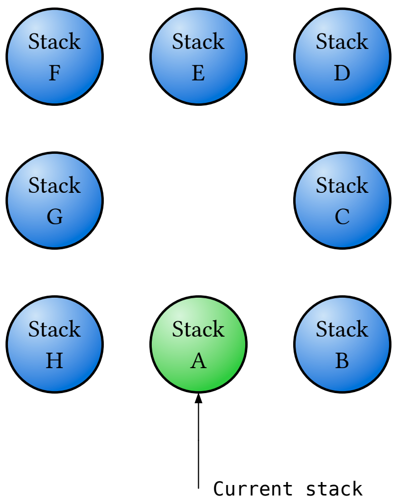
```Bund
//
// Here is an example of declaration of
// anonymous function - lambda
//
{
  "Hello world!"
  println
}
//
// This function will print "Hello world!"
// on terminal
//
```

Stack-related functions

The BUND programming language incorporates a key design feature: the data context located in circular stacks. While BUND offers a wide variety of functions, including internal parts of the standard library, named functions, and anonymous lambda functions, it does not provide a context specific to the function. Instead, it offers a unified data storage context through named and anonymous circular stacks. Functions can retrieve data from the stack and store results in the stacks according to the function's design. Additionally, BUND provides a library of functions for managing data context and contexts, which we will explore further in the following chapters.

Functions for the "stack of stacks"

This chapter explores the functions of the "stack-of-stacks" data structure. This structure consists of a circular buffer containing references to other circular buffers holding the data. The functions are specifically designed to manage the list of stacks, including adding new ones, removing stacks, and positioning the list of stacks.

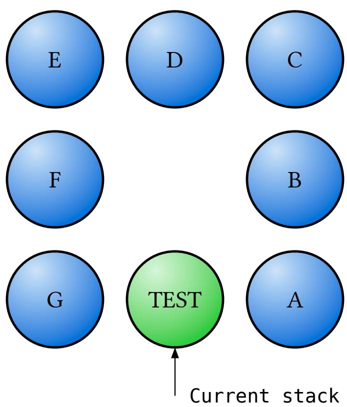


Stacks management word: making named stack current

The function *ensure_stack* will set the named stack as current. If the stack does not exist, it will be created.

```
function ENSURE_STACK()
    ▷ Making named stack current
    X ← current stack
    if X = None then
        return Error("Stack is too shallow")
    if Stack.Not.Exists X then
        CREATESTACK(X)
    MAKESTACKCURRENT(X)
```

```
//
// This snippet will make stack TEST current
//
:TEST ensure_stack
```



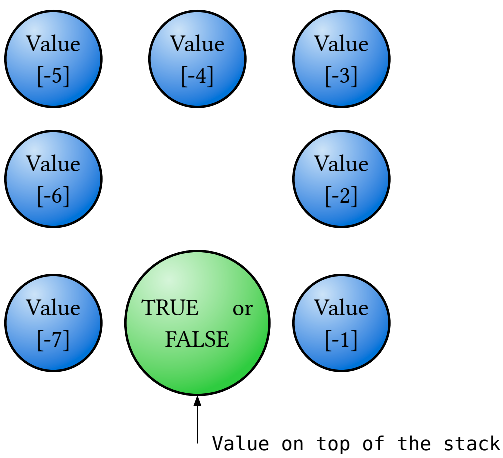
Stacks management word: check if stack exists

The function *stack_exists* will return TRUE value to the current stack if named stack exists. FALSE - otherwise.

function STACK_EXISTS()
        ▷ Check if stack exitst
        X ← *current stack*
        if X = None **then**
                return Error("Stack is too shallow")
        if Stack.Not.Exists X **then**
                current stack ← FALSE
        else
                current stack ← TRUE

Bund

```
//
// This snippet will check if named stack exists
//
:TEST stack_exists
```



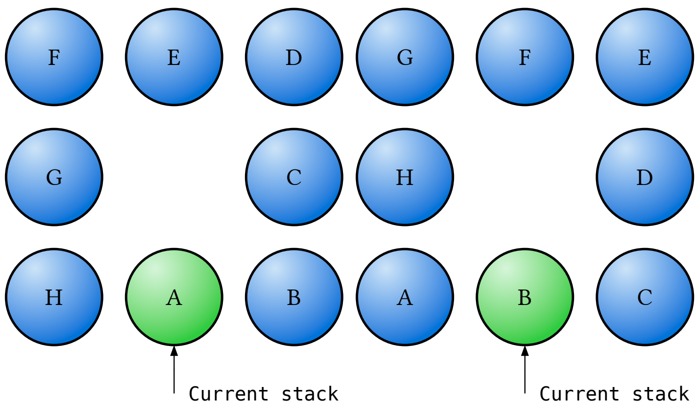
Stacks management word: rotate left

The function `<-` will rotate stacks circular buffer to the left.

```
function STACKS_LEFT()
    ▷ Rotate stacks circular buffer left
    STACKS_LEFT()
```

Bund

```
//
// This snippet will rotate stacks circular buffer left
//
<-
```



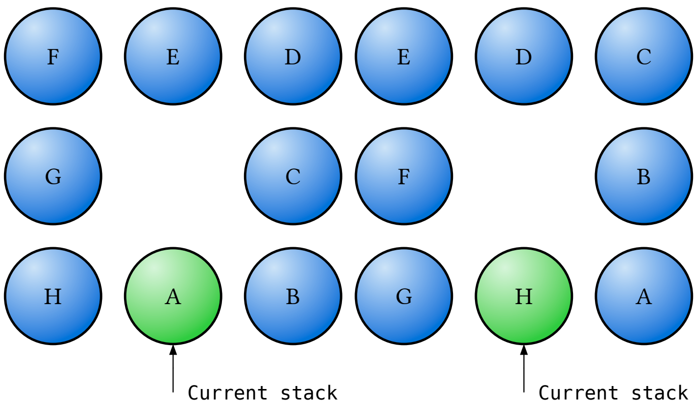
Stacks management word: rotate right

The function `->` will rotate stacks circular buffer to the right.

function STACKS_RIGHT()
    ▷ Rotate stacks circular buffer right
    STACKS_RIGHT()

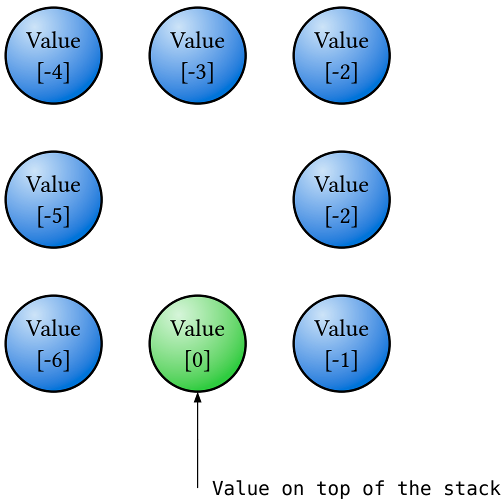
Bund

```
//
// This snippet will rotate stacks
// circular buffer to the right
//
->
```



Managing data on stack

In this chapter, we will explore the fundamental functions, referred to as "words" in concatenative languages, for managing circular stacks containing data. We have previously examined how to add data to the stack, so this chapter will teach you how to manipulate the existing data on the stack and the stack itself.



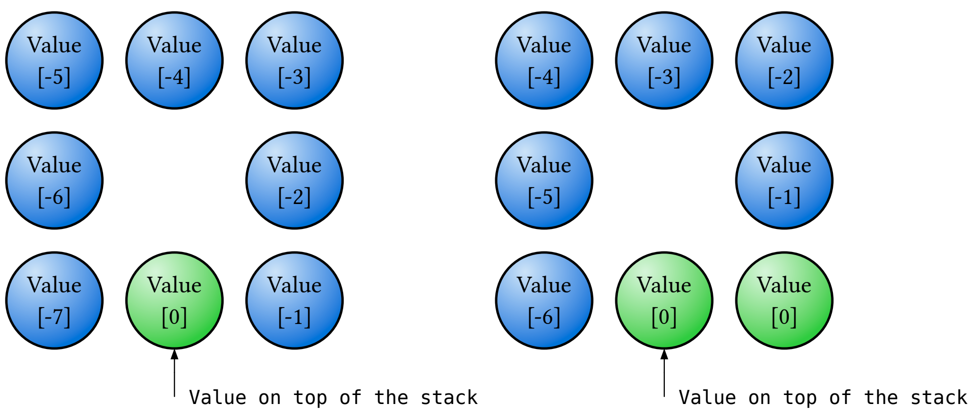
Stack management word: duplicating data on the stack

The function “dup” will duplicate value located on top of the stack. Two values, original one and duplicated are returned to the stack.

function DUP()
    ▷ Duplicating value that is on top of the stack
    Value ← *current stack*
    if Value = None **then**
        return Error(“Stack is too shallow”)
    Value2 ← **Value::dup** (Value)
    current stack ← Value
    current stack ← Value2

Bund

```
//
// This snippet will duplicate value on top of the stack
//
42 dup
```



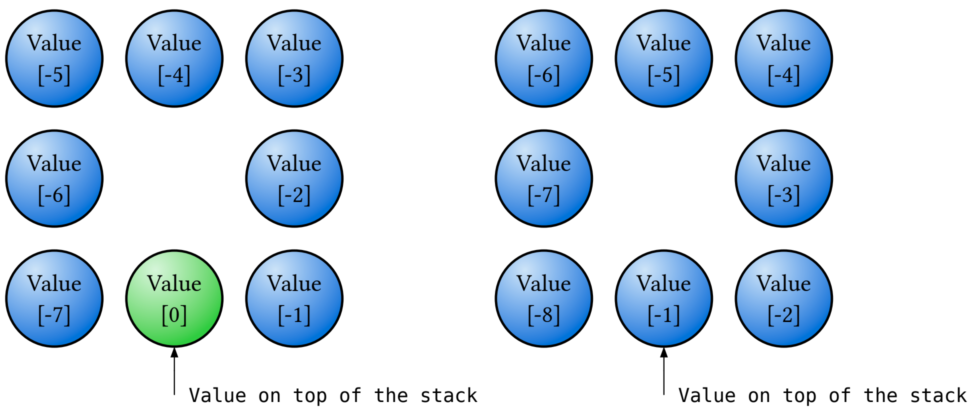
Stack management word: dropping data from stack

The function “drop” will remove current value from the stack

function DROP()
    ▷ Dropping value that is on top of the stack
    Value ← *current stack*
    if Value = None **then**
        return Error(“Stack is too shallow”)

Bund

```
//
// This snippet will remove value 42 from stack
// leaving 41 on top of current stack
//
41 42 drop
```



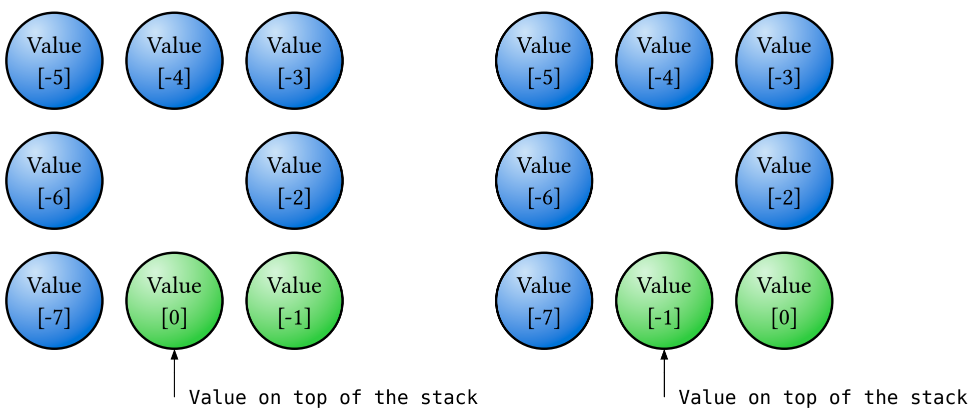
Stack management word: swapping two values.

The function “swap” will swap two values on top of current stack

function SWAP()
    ▷ Swapping values that is on top of the stack
    X ← *current stack*
    if X = None **then**
        return Error(“Stack is too shallow”)
    Y ← *current stack*
    if Y = None **then**
        return Error(“Stack is too shallow”)
    current stack ← Y
    current stack ← X

Bund

```
//
// This snippet will swap values 42 and 41 on stack
// leaving 42 on top of current stack
//
42 41 swap
```



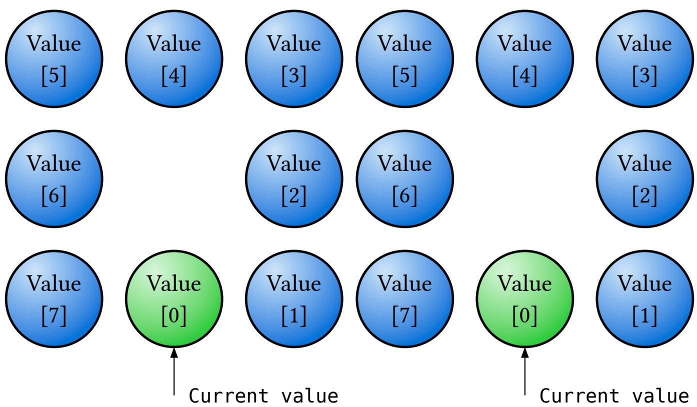
Stack management word: rotate left

The function `<--` will rotate current stack to the right.

function STACK_LEFT()
    ▷ Rotate stack circular buffer right
    STACK_LEFT()

Bund

```
//
// This snippet will rotate current stack
// circular buffer to the left
//
<--
```

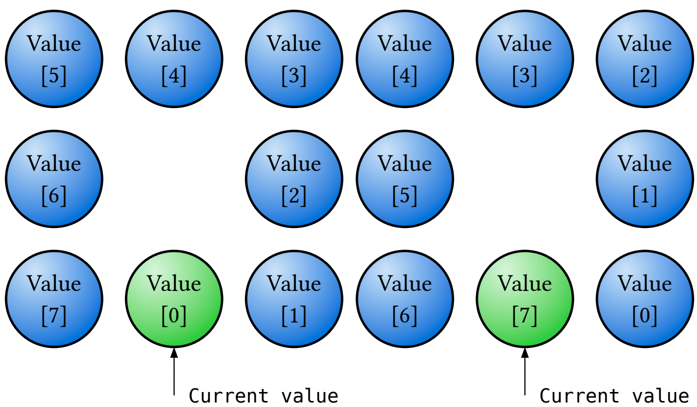


Stack management word: rotate right

The function `-->` will rotate current stack to the right.

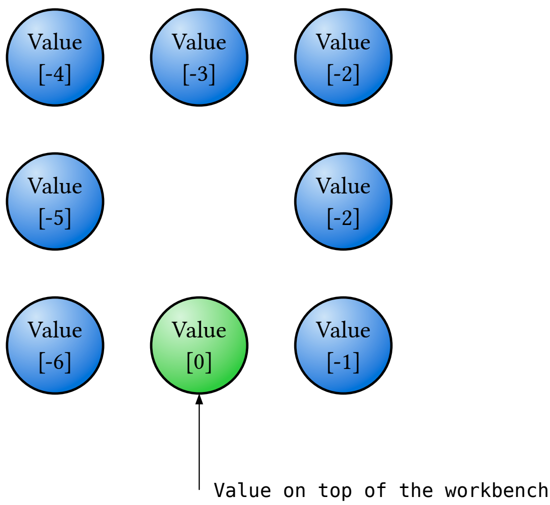
function STACK_RIGHT()
  ▷ Rotate stack circular buffer right
  STACK_RIGHT()

```
//
// This snippet will rotate current stack
// circular buffer to the right
//
-->
```



Managing data on workbench

The workbench is a dedicated circular stack that temporarily holds data between computations in named and anonymous data contexts. Its specific purpose is not defined, allowing developers to use it as they see fit. The workbench can be used as temporary storage for intermediate computations, passing data between data contexts, or holding permanent data useful for computations in different contexts. There are indeed no limitations to its use.



Workbench management word: taking value from stack to workbench.

This function will take value from top of the stack and push to a workbench

function RETURN()
    ▷ Pushing value to a workbench
    X ← *current stack*
    if X = None **then**
        return Error("Stack is too shallow")
    workbench ← X

```
//
// This snippet will send value to a workbench
//
42 .
```

Bund

Workbench management word: taking value from workbench to stack.

This function will take value from workbench and push to a top of current stack

function TAKE()
  ▷ Pushing value to a workbench
  X ← *workbench*
  if X = None **then**
    return Error("Workbench is too shallow")
  current stack ← X

Bund

```
//
// This snippet will send value to stack
//
42 .     // First, let's populate workbench
take     // Then move value back 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]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or im-

Content