UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

Master's Thesis

Martin Holm Cservenka
Department of Computer Science
University of Copenhagen
djp595@alumni.ku.dk

Design and Implementation of Dynamic Memory Management in a Reversible Object-Oriented Programming Language

Supervisors: Robert Glück & Torben Ægidius Mogensen

Submitted: January 25^{th}, 2018

Revision 1.01

Revision History

Revision	Date	Author(s)	Description
1.01	2018-04-11	Martin	Fixed incorrect subfigures (d, e, f) in Figure 4.11. Updated Figure 3.4 to better emphesize output copying. Updated Figure 3.5 to better reflect the harmless garbage flow through programs.

Abstract

The reversible object-oriented programming language (ROOPL) was presented in late 2016 and proved that object-oriented programming paradigms works in the reversible setting. The language featured simple statically scoped objects which made non-trivial programs tedious, if not impossible to write using the limited tools provided. We introduce an extension to ROOPL in form the new language ROOPL++, featuring dynamic memory management and fixed-sized arrays for increased language expressiveness. The language is a superset of ROOPL and has formally been defined by its language semantics, type system and computational universality. Considerations for reversible memory manager layouts are discussed and ultimately lead to the selection of the Buddy Memory layout. Translations of the extensions added in ROOPL++ to the reversible assembly language PISA are presented to provide garbage-free computations. The dynamic memory management extension successfully increases the expressiveness of ROOPL and as a result, shows that non-trivial reversible data structures, such as binary trees and doubly-linked lists, are feasible and do not contradict the reversible computing paradigm.

Preface

This Master's Thesis is submitted as the last part for the degree of Master of Science in Computer Science at the University of Copenhagen, Department of Computer Science, presenting a 30 ECTS workload.

The thesis consists of 231 pages and a ZIP archive containing source code and test programs developed as part of the thesis work.

I would like to thank my two supervisors, Robert Glück and Torben Mogensen, for their invaluable supervision and guidance throughout this project and introduction to the field of reversible computing. A big thanks to my university colleague and friend, Tue Haulund, for allowing me to continue his initial work on Roopl and providing information, sparring and source code material and for being a great ally through our years at the University of Copenhagen. In addition, thanks to my dear aunt Doris, for financially supporting my studies by paying for all my books needed. Finally, a thanks to Jess, for all the love and support throughout the entire span of my thesis process.

Table of Contents

List of Figures

CHAPTER 1

Introduction

In recent years, technologies such as cloud-based services, deep learning, cryptocurrency mining and other services requiring large computational power and availability have been on the rise. Most of these services are hosted on massive server parks, consuming immense amounts of electricity in order to power the machines and the cooling architectures as heat dissipates from the hardware. A recent study showed that the Bitcoin network including its mining processes' currently stands at 0.13% of the total global electricity consumption, rivaling the usage of a small country like Denmark's [6]. With the recent years focus on climate and particularly energy consumption, companies have started to attempt to reduce their power usage in these massive server farms. As an example, Facebook built new server park in the arctic circle in 2013, in an attempt to take advantage of the natural surroundings in the cooling architecture to reduce its power consumption [24].

Reversible computing presents a possible solution the problematic power consumption issues revolving around computations. Traditional, irreversible computers dissipates heat during their computation. Landauer's principle states that deletion of information in a system always results by an increase in energy consumption. In reversible computing, all information is preserved throughout the execution, and as such, the energy consumption theoretically should be smaller [14].

Currently, reversible computing is not commercially appealing, as it is an area which still is being actively researched. However, several steps has been taken in the direction of a fully reversible system, which some day might be applicable in a large setting. Reversible machine architectures have been presented such as the Pendulum architecture and its instruction set Pendulum ISA (Pisa) [26, 3] and the BobISA architecture and instruction set [23] and high level languages Janus [16, 31, 29] and R [7] exists.

While cryptocurrency mining and many other computations are not reversible, the area remains interesting in terms of its applications and gains.

1.1 Reversible Computing

Reversible computing is a two-directional computational model in which all processes are time-invertible. This means, that at any time during execution, the computation can return to a former state. In order to maintain *reversibility*, the reversible computational model cannot compute *many-to-one* functions, as the models requires an exact inverse f^{-1} of a function f in order to

support backwards determinism. Therefore, reversible programs must only consist of *one-to-one* functions, also known as *injective* functions, which result in a garbage-free computation, as garbage-generating functions simply can be unwinded to clean up.

Each step of a reversible program is locally invertible, meaning each step has exactly one inverse step. A reversible program can be inverted simply by computing the inverse of each of its steps, without any knowledge about the overall functionality or requirements of the program. This property immediately yields interesting consequences in terms of software development, as an encryption or compression algorithm implemented in a reversible language immediately yields the decryption or decompression algorithm by running the algorithm backwards.

The reversibility is however not free and comes and the cost of strictness when writing programs. Almost every popular, irreversible programming language features a conditional component in form of **if-else**-statements. In these languages, we only define the *entry*-condition in the conditional, that is, the condition that determines which branch of the component we continue execution in. In reversible languages, we must also specify an *exit*-condition, such that we can determine which branch we should follow, when executing the program in reverse. In theory, this sounds trivial, but in practice it turns to add a new layer of complexity when writing programs.

1.2 Object-Oriented Programming

Object-oriented programming (OOP) has for many years been the most widely used programming paradigm as reflected in the popular usage of object-oriented programming languages, such as the C-family languages, Java, PHP and in recent years Javascript and Python. The OOP core concepts such as *inheritance*, *encapsulation* and *polymorphism* allows complex systems to be modeled by breaking the system into smaller parts in form of abstract objects [17].

1.3 Reversible Object-Oriented Programming

The high-level reversible language Roopl (Reversible Object-Oriented Programming Language) was introduced in late 2016 [11, 12]. The language extends the design of previously existing reversible imperative languages with object-oriented programming language features such as user-defined data types, class inheritance and subtype-polymorphism. As a first, Roopl successfully integrates the object-oriented programming (OOP) paradigms into the reversible computation setting using a static memory manager to maintain garbage-free computation, but at cost of programmer usability as objects only lives within **construct** / **deconstruct** blocks, which needs to be predefined, as the program call stack is required to be reset before program termination.

Conceptualizations and ideas for the Joule language was also published in 2016 [22]. The language, a homonym of Janus Object-Oriented Language, Jool, presented an alternative OOP extension to Janus, differing from Roopl. The language featured heap allocated objects with constructors and multiple object references, as such also addressing the problems with Roopl. The language is still a work in progress, aiming to provide a useful, reversible object oriented-programming language.

1.4 Motivation

The block defined objects of ROOPL and lack of multiple references are problematic when writing complex, reversible programs using OOP methodologies as they pose severe limitations on the expressiveness. It has therefore been proposed to extend and partially redesign the language with dynamic memory management in mind, such that these shortcomings can be addressed, and ultimately increase the usability of reversible OOP. Work within the field of reversible computing related to heap manipulation [2], reference counting [18] and garbage collection [19] suggests that a ROOPL extension is feasible.

1.5 Thesis Statement

An extension of the reversible object-oriented programming language with dynamic memory management is feasible and effective. The resulting expressiveness allows non-trivial reversible programming previously unseen, such as reversible data structures, including linked lists, doubly linked lists and trees.

1.6 Outline

This Master's thesis consists of four chapters, besides the introductory chapter. The following summary describes the following chapters.

- **Chapter 2** formally defines the ROOPL extension exemplified by the new language ROOPL++, a superset of ROOPL.
- **Chapter 3** serves as a brief description of dynamic memory management along with a discussion of various reversible, dynamic memory management layouts.
- **Chapter 4** presents the translation techniques utilized in compiling a ROOPL++ program to PISA instructions.
- **Chapter 5** presents the conclusions of the thesis and future work proposals.

Besides the five chapters, a number of appendices is supplied, containing PISA translations of the reversible heap allocation algorithm, the source code of the ROOPL++ to PISA compiler, the ROOPL++ source code for the example programs and their translated PISA versions.

CHAPTER 2

The ROOPL++ Language

With the design and implementation of the REVERSIBLE OBJECT-ORIENTED PROGRAMMING LANGUAGE (ROOPL) and the work-in-progress report of JOULE, the first steps into the uncharted lands of Object-Oriented Programming (OOP) and reversibility were taken. In this chapter, we present ROOPL++, the natural successor to ROOPL, improving the object instantiation of the language by letting objects live outside **construct**/**deconstruct** blocks, allowing complex, reversible programs to be written using OOP methodologies. As with its predecessor, ROOPL++ is purely reversible and each component of a program written in ROOPL++ is locally invertible. This ensures no computation history is required, nor added program size for backwards program execution.

Inspired by other language successors such as C++ was to C, ROOPL++ is a superset of ROOPL, containing all original functionality of its predecessor, extended with new object instantiation methods for increased programming usability and an array type.

```
1  class Fib
2      int[] xs
3
4      method init()
5          new int[2] xs
6
7      method fib(int n)
8          if n = 0 then
9              xs[0] ^= 1
10             xs[1] ^= 1
11         else
12             n -= 1
13             call fib(n)
14             xs[0] += xs[1]
15             xs[0] <=> xs[1]
16         fi xs[0] = xs[1]
17
18     method get(int out)
19         out ^= xs[1]
20
21 class Program
22     int result
23     int n
24
25     method main()
26         n ^= 4
27
28         new Fib f
29         call f::init()
30         call f::fib(n)
31         call f::get(result)
32         uncall f::fib(n)
33         uncall f::init()
34         delete Fib f
```

Figure 2.1: Example ROOPL++ program implementing the Fibonacci function

2.1 Syntax

A ROOPL++ program consists, analogously to a ROOPL program, of one or more class definitions, each with a varying number of fields and class methods. The entry point of the program is a nullary main method, which is defined exactly once and is instantiated during program start-up. Fields of the main object will serve as output of the program, just as in ROOPL.

ROOPL++ Grammar

$prog$	::=	cl^+	(program)
cl	::=	**class** c (**inherits** c)$^?$ $(t\ x)^*$ m^+	(class definition)
d	::=	$c \mid c[e] \mid$ **int**$[e]$	(class and arrays)
t	::=	**int** $\mid c \mid$ **int**[] $\mid c[]$	(data type)
y	::=	$x \mid x[e]$	(variable identifiers)
m	::=	**method** q **(**$t\ x,\ \ldots,\ t\ x$**)** s	(method)
s	::=	$y\ \odot$ **=** $e \mid y$ **<=>** y	(assignment)
	$\mid$	**if** e **then** s **else** s **fi** e	(conditional)
	$\mid$	**from** e **do** s **loop** s **until** e	(loop)
	$\mid$	**construct** $c\ x$ s **destruct** x	(object block)
	$\mid$	**local** $t\ x = e$ s **delocal** $t\ x = e$	(local variable block)
	$\mid$	**new** $d\ y \mid$ **delete** $d\ y$	(object con- and destruction)
	$\mid$	**copy** $d\ y\ y \mid$ **uncopy** $d\ y\ y$	(reference con- and destruction)
	$\mid$	**call** q **(**$x,\ \ldots,\ x$**)** $\mid$ **uncall** q **(**$x,\ \ldots,\ x$**)**	(local method invocation)
	$\mid$	**call** y**::**q **(**$x,\ \ldots,\ x$**)** $\mid$ **uncall** y**::**q **(**$x,\ \ldots,\ x$**)**	(method invocation)
	$\mid$	**skip** $\mid s\ s$	(statement sequence)
e	::=	$\overline{n} \mid x \mid x[e] \mid$ **nil** $\mid e\ \otimes\ e$	(expression)
$\odot$	::=	**+** $\mid$ **-** $\mid$ **^**	(operator)
$\otimes$	::=	$\odot \mid$ ***** $\mid$ **/** $\mid$ **%** $\mid$ **&** $\mid$ **|** $\mid$ **&&** $\mid$ **||** $\mid$ **<** $\mid$ **>** $\mid$ **=** $\mid$ **!=** $\mid$ **<=** $\mid$ **>=**	(operator)

Syntax Domains

$prog \in$ Programs	$s \in$ Statements	$n \in$ Constants
$cl \in$ Classes	$e \in$ Expressions	$x \in$ VarIDs
$t \in$ Types	$\odot \in$ ModOps	$q \in$ MethodIDs
$m \in$ Methods	$\otimes \in$ Operators	$c \in$ ClassIDs

Figure 2.2: Syntax domains and EBNF grammar for ROOPL++

The ROOPL++ grammar extends the grammar of ROOPL with a new static integer or class array type and a new object lifetime option in form of objects outside of blocks, using the **new** and

delete approach. Furthermore, the local block extension proposed in [11] has become a standard part of the language. Class definitions remains unchanged, and consists of a **class** keyword followed by a class name. Subclasses must be specified using the **inherits** keyword and a following parent class name. Classes can have any number of fields of any of the data types, including the new Array type. A class definition is required to include at least one method, defined by the **method** keyword followed by a method name, a comma-separated list of parameters and a body.

Reversible assignments for integer variables and integer array elements uses similar syntax as JANUS assignments, by updating a variable through any of the addition (+=), subtraction (−=) or bitwise XOR (^=) operators. As with JANUS, when updating a variable x using any of said operators, the right-hand side of the operator argument must be entirely independent of x to maintain reversibility. Usage of these reversible assignment operators for object or array variables is undefined. Variables and array elements of any type can be swapped using the <=> operator as long as the variable is of same type as the array type. If an array is of a base class type, subclass variable values can be swapped in and out of the array, as long as the resulting value in the variable is still of the original subclass type.

ROOPL++ objects can be instantiated in two ways. Either using object blocks known from ROOPL, or by using the **new** statement. The object-blocks have a statically-scoped lifetime, as the object only exists within the **construct** and **destruct** segments. Using **new** allows the object to live until program termination, if the program terminates with a **delete** call. By design, it is the programmers responsibility to deallocate objects instantiated by the **new** statement.

Arrays are also instantiated by usage of **new** and **delete**. Assignment of array cells depend on the type of the arrays, which is further discussed in section 2.4.

The methodologies for argument aliasing and its restrictions on method on invocations from ROOPL carries over in ROOPL++ and object fields are as such disallowed as arguments to local methods to prevent irreversible updates and non-local method calls to a passed objects are prohibited. The parameter passing scheme remains call-by-reference and the object model of ROOPL remains largely unchanged in ROOPL++.

2.2 Object Instantiation

Object instantiation through the **new** statement follows the pattern of the mechanics known from the **construct**/**destruct** blocks from ROOPL, but providing improved scoping and lifetime options objects. The mechanisms of the statement

$$\textbf{construct}\ c\ x \quad s \quad \textbf{destruct}\ x$$

are as follows:

1. Memory for an object of class c is allocated. All fields are automatically zero-initialized by virtue of residing in already zero-cleared memory.
2. The block statement s is executed, with the name x representing a reference to the newly allocated object.

3. The reference x may be modified by swapping its value with that of other references of the same type, but it should be restored to its original value within the statement block s, otherwise the meaning of the object block is undefined.
4. Any state that is accumulated within the object should be cleared or uncomputed before the end of the statement is reached, otherwise the meaning of the object block is undefined.
5. The zero-cleared memory is reclaimed by the system.

The statement pair consisting of

new c x ... **delete** c x

could be considered a *dynamic* block, meaning we can have overlapping blocks. Compared to **construct**/**destruct** block consisting of a single statement, the **new**/**delete** block consist of two separate statements. We can as such initialize an object x of class c and an object y of class d and destroy x before we destroy y, a feature that was not possible in Roopl. The mechanisms of the **new** statement are as follows:

1. Memory for an object of class c is allocated. All fields are automatically zero-initialized by virtue of residing in already zero-cleared memory.
2. The address of the newly allocated block is stored in the previously defined and zero-cleared reference x.

and the mechanisms of the **delete** statement are as follow

1. The reference x may be modified by swapping its internal field values with that of other references of the same type, but should be zero-cleared before a **delete** statement is called on x, otherwise the meaning of the object deletion is undefined.
2. Any state that is accumulated within the object should be cleared or uncomputed before the **delete** statement is executed, otherwise the meaning of the object block is undefined.
3. The zero-cleared memory is reclaimed by the system.

The mechanisms of the **new** and **delete** statements are, essentially, a split of the mechanisms of the **construct**/**destruct** blocks into two separate statements. As with Roopl, fields must be zero-cleared after object deletion, otherwise it is impossible for the system to reclaim the memory reversibly. This is the responsibility of the of the programmer to maintain this, and to ensure that objects are indeed deleted in the first place. A **new** statement without a corresponding **delete** statement targeting the same object further ahead in the program is undefined, as is a delete statement without a preceding **new** statement.

Note that variable scopes are always static, but object scopes can be either static (using **construct**/**destruct**) or dynamic (using **new**/**delete**).

2.3 Array Model

Besides asymmetric object lifetimes, ROOPL++ also introduces reversible, fixed-sized arrays of either integer or object types. While ROOPL only featured integers and custom data types in form of classes, one of its main inspirations, JANUS, implemented static, reversible arrays [31].

While ROOPL by design did not include any data storage language constructs, as they are not especially noteworthy nor interesting from an OOP perspective, they do generally improve the expressiveness of the language. Arrays were decided to be part of the core language for this reason, as one of the main goals of ROOPL++ is increased expressiveness while implementing reversible programs.

In ROOPL++, arrays expand upon the array model from JANUS. Arrays are indexed by integers, starting from 0. In JANUS, only integer arrays were allowed, while in ROOPL++ arrays of any type can be defined, meaning either integer arrays or custom data types in form of class arrays. They are however, still restricted to one dimension.

Array element accessing is accomplished using the bracket notation known from JANUS. Accessing an out-of-bounds index is undefined. Array instantiation and element assignments, aliasing and circularity is described in detail in the following section.

Arrays can contain elements of different classes sharing a base class, that is, say class A and B both inherit from some class C and array x is of type $C[]$. In this case, the array can hold elements of type A, B, and C. When swapping array elements from a base class array with object references, the programmer must be careful not to swap the values of, say, and A object into a B reference.

2.4 Array Instantiation

Array instantiation uses the **new** and **delete** keywords to reversibly construct and destruct array types. The mechanisms of the statement

$$\textbf{new int}[e]\ x$$

in which we reserve memory for an integer array are as follows

1. The expression e is evaluated
2. Memory equal to the integer value that e evaluates to and an additional small amount of memory for of overhead is reserved for the array.
3. The address of the newly allocated memory is stored in the previously defined and zero-cleared reference x.

In ROOPL++, we only allow instantiation of fixed-sized arrays of a length defined in the given expression e. Array elements are assigned dependent on the type of the array. For integer arrays, any of the reversible assignment operators can be used to assign values to cells. For class arrays, we assign cell elements a little differently. We either make use of the **new** and **delete** statements, but instead of specifying which variable should hold the newly created/deleted object or array,

we specify which array cell it should be stored in or we use the **swap** statement to swap values in and out of array cells. Usage of the assignment operators on non-integer arrays is undefined.

```
1    new int[5] intArray          // Init new integer array
2    new Foo[2] fooArray          // init new Foo array
3
4    intArray[1] += 10            // Legal array integer assignment
5    intArray[1] -= 10            // Legal Zero-clearing for integer array cells
6
7    new Foo fooObject
8    fooArray[0] <=> fooObject    // Legal object array cell assignment
9    new Foo fooArray[2]          // Legal object array cell assignment
10
11   ...                          // Clear all array cells
12
13   delete Foo fooArray[0]       // Legal object array cell zero-clearing
14   delete Foo fooArray[1]       // Legal object array cell zero-clearing
```

Listing 2.1: Assignment of array elements

As with ROOPL++ objects instantiated outside of **construct**/**destruct** blocks, arrays must be deleted before program termination to reversibly allow the system to reclaim the memory. Before deletion of an array, all its elements must be zero-cleared such that no garbage data resides in memory after erasure of the array reference.

Consider the statement

$$\textbf{delete int}[e]\ x$$

with the following mechanics

1. The reference x may be modified by swapping, assigning cell element values and zero-clearing cell element values, but must be restored to an array of same type with fully zero-cleared cells before the **delete** statement. Otherwise, the meaning of the statement is undefined.
2. The value of the expression e is evaluated and used to reclaim the allocated memory space.
3. If the reference x is a fully zero-cleared array upon the **delete** statement execution, the zero-cleared memory is reclaimed by the system.

With reversible, fixed-sized arrays of varying types, we must be extremely careful when updating and assigning values, to ensure we maintain reversibility and avoid irreversible statements. Therefore, when assigning or updating integer elements with one of the reversible assignment operators, we prohibit the cell value from being reference on the right hand side, meaning the following statement is prohibited

$$x[5]\ \mathrel{+}=\ x[5] + 1$$

However, we do allow other initialized, non-zero-cleared array elements from the same array or arrays of same type to be referenced in the right hand side of the statement. As with regular assignment, we still prohibit the left side reference to occur in the ride side, meaning the following statements are also prohibited

$$x\ \mathrel{+}=\ y[x]$$
$$y[x]\ \mathrel{+}=\ x$$

2.5 Referencing

Besides the addition of dynamically lifetimed objects and arrays, ROOPL++ also increases program flexibility by allowing multiple references to objects and arrays through the usage of the **copy** statement. Once instantiated through either a **new** or **construct**/**destruct** block, an object or array reference can be copied into another zero-cleared variable. The reference acts as a regular instance and can be modified through methods as per usual. To delete a reference, the logical inverse statement **uncopy** must be used.

The syntax for referencing consists of the statement

$$\textbf{copy}\ c\ x\ x'$$

which copies a reference of variable x, an instance of class or array c, and stores the reference in variable x'.

For deleting copies, the following statement is used

$$\textbf{uncopy}\ c\ x\ x'$$

which simply zero-clears variable x', which is a reference to variable x, an instance of class or array c.

The mechanism of the **copy** statement is simply as follows

1. The memory address stored in variable x is copied into the zero-cleared variable x'. If x' is not zero-cleared or x is not a class instance, then **copy** is undefined.

The mechanism of the **uncopy** statement is simply as follows

1. The memory address stored in variable x' is zero-cleared if it matches the address stored in x. If x' is not a copy of x or x has been zero-cleared before the **uncopy** statement is executed, said statement is undefined.

As references do not require all fields or cells to be zero-cleared (as they are simple pointers to existing objects or arrays), the reversible programmer should carefully ensure that all references are un-copied before deleting said object or array, as copied references to cleared objects or arrays would be pointing to cleared memory, which might be used later by the system. These type of references are also known as *dangling pointers*.

It should be noted, that from a language design perspective, it is the programmer's responsibility to ensure such situations do not occur. From an implementation perspective, such situations are usually checked by the compiler either statically during compilation or during the actual runtime of the program. This is addressed later in sections 3.3 and 4.9.

2.6 Local Blocks

The local block presented in the extended JANUS in [29] consisted of a local variable allocation, a statement and a local variable deallocation. These local variable blocks add immense programmer usability as the introduce a form of reversible temporary variable. The ROOPL compiler features support for local integer blocks, but not object blocks. In ROOPL++, local blocks can be instantiated with all of the languages variable types; integers, arrays and user-defined types in the form of objects.

Local integer blocks works exactly the same as in ROOPL and JANUS, where the local variable initialized will be set to the evaluated result of a given expression.

Local array and object blocks feature a number of different options. If a local array or object block is initialized with a **nil** value, the variable must afterwards be initialized using a new statement before any type-specific functionality is accessible. If the block is initiated with an existing object or array reference, the local variable essentially becomes a reference copy, analogous to a variable initialized from a **copy** statement.

$$\textbf{construct}\ c\ x \quad s \quad \textbf{destruct}\ x \quad \overset{\underline{\textbf{def}}}{=\!=} \quad \begin{array}{l} \textbf{local}\ c\ x\ = \textbf{nil} \\ \textbf{new}\ c\ x \quad s \quad \textbf{delete}\ c\ x \\ \textbf{delocal}\ c\ x = \textbf{nil} \end{array}$$

Figure 2.3: construct/destruct-blocks can be considered syntactic sugar

For objects, the **construct**/**destruct**-blocks can be considered syntactic sugar for a local block defined with a **nil** value, containing a **new** statement in the beginning of its statement block and a **delete** statement in the very end, as shown in figure 2.3.

As local array and object blocks allow freedom in terms of their interaction with other statements in the language, it is the programmer's responsibility that the local variable is deallocated using a correct expression at the end of the block definition. The value of the variable is a pointer to an object or an array. Said object or array must have all fields/cells zero-cleared before the pointer is zero-cleared at the end of the local block. If the pointer is at any point exchanged with the pointer of another object or array using the **swap** statement, the same conditions apply.

2.7 ROOPL++ Expressiveness

By introducing dynamic lifetime objects and by allowing objects to be referenced multiple times, we can express non-trivial reversible programs. To demonstrate the capacities, expressiveness and possibilities of ROOPL++, the following section presents previously unseen reversible data structures, which now are feasible, written in ROOPL++.

2.7.1 Linked List

Haulund presented a linked list implemented in ROOPL in [11]. The implementation featured a *ListBuilder* and a *Sum* class, required to determine and retain the sum of a constructed linked list as the statically scoped object blocks of ROOPL would deallocate automatically after building the full list. In ROOPL++, we do not face the same challenges and the implementation becomes much more straightforward. Figure 2.5 implements a *LinkedList* class, which simply has the head of the list and the list length as its internal fields. For demonstration, the class allows extension of the list by either appending or prepending cell elements to the list. In either case, we first check if the *head* field is initialized. If not, the cell we are either appending or prepending simply becomes the new head of the list. If we are appending a cell the Cell-class *append* method is called on the *head* cell with the new cell as its only argument. When prepending, the existing head is simply appended to the new cell and the new cell is set as head of the linked list.

```
1    class Cell
2        Cell next
3        int data
4
5        method constructor(int value)
6            data ^= value
7
8        method append(Cell cell)
9            if next = nil & cell != nil then
10               next <=> cell           // Store as next cell if current cell is end of list
11           else skip
12           fi next != nil & cell = nil
13
14           if next != nil then
15               call next::append(cell) // Recursively search until we reach end of list
16           else skip
17           fi next != nil
```

Figure 2.4: Linked List cell class

Figure 2.4 shows the *Cell* class of the linked list which has a *next* and a *data* field, a constructor and the *append* method. The append method works by recursively looking through the linked cell nodes until we reach the end of the free list, where the *next* field has not been initialized yet. When we find such a cell, we simply swap the contents of the *next* and *cell* variables, s.t. the cell becomes the new end of the linked list.

An interesting observation is that the *append* method is called an additional time *after* setting the cell as the new end of the linked list. In a non-reversible programming language, we would simply call append in the else-branch of the first conditional. In the reversible setting, this is not an option, as the append call would modify the value of the *next* and *cell* variables and as

```
1   class LinkedList
2       Cell head
3       int listLength
4
5       method insertHead(Cell cell)
6           if head = nil & cell != nil then
7               head <=> cell               // Set cell as head of list if list is empty
8           else skip
9           fi head != nil & cell = nil
10
11      method appendCell(Cell cell)
12          call insertHead(cell)           // Insert as head if empty list
13
14          if head != nil then
15              call head::append(cell)     // Iterate until we hit end of list
16          else skip
17          fi head != nil
18
19          listLength += 1                 // Increment length
20
21      method prependCell(Cell cell)
22          call insertHead(cell)           // Insert as head if empty list
23
24          if cell != nil & head != nil then
25              call cell::append(head)     // Set cell.next = head. head = nil after execution
26          else skip
27          fi cell != nil & head = nil
28
29          if cell != nil & head = nil then
30              cell <=> head               // Set head = cell. Cell is nil after execution
31          else skip
32          fi cell = nil & head != nil
33
34          listLength += 1                 // Increment length
35
36      method length(int result)
37          result ^= listLength
```

Figure 2.5: Linked List class

such, corrupt the control flow as the exit condition would be true after executing both the then- and else-branch of the conditional. To avoid this, we simply call one additional time with a **nil** value *cell*. This "wasted" additional call with a **nil** value is a recurring technique in the following presented reversible data structure implementations.

2.7.2 Binary Tree

Figures 2.6, 2.8 and 2.7 shows the implementation of a binary tree in form of a rooted, unbalanced, min-heap. The *Tree* class shown in figure 2.6 has a single root node field and the three methods *insertNode, sum* and *mirror*. For insertion, the *insertNode* method is called from the *root*, if it is initialized and if not, the passed node parameter is simply set as the new root of the tree. The *insertNode* method implemented in the *Node* class shown in figure 2.8 first determines if we need to insert left or right but checking the passed value against the value of the current node. This is done recursively, until an uninitialized node in the correct subtree has been found. Note that as a consequence of reversibility, the value of node we wish to insert must be passed separately in the method call as we otherwise cannot zero-clear it after swapping the node we are inserting

with either the right or left child of the current cell.

```
1     class Tree
2         Node root
3
4         method insertNode(Node node, int value)
5             if root = nil & node != nil then
6                 root <=> node
7             else skip
8             fi root != nil & node = nil
9
10            if root != nil then
11                call root::insertNode(node, value)
12            else skip
13            fi root != nil
14
15        method sum(int result)
16            if root != nil then
17                call root::getSum(result)
18            else skip
19            fi root != nil
20
21        method mirror()
22            if root != nil then
23                call root::mirror()
24            else skip
25            fi root != nil
```

Figure 2.6: Binary Tree class

Summing and mirroring the tree works in a similar fashion by recursively iterating each node of the tree. For summing we simply add the value of the node to the sum and for mirroring we swap the children of the node and then recursively swap the children of the left and right node, if initialized. The sum and mirror methods are implemented in figure 2.7.

```
1     method getSum(int result)
2         result += value                  // Add the value of this node to the sum
3
4         if left != nil then
5             call left::getSum(result)    // If we have a left child, follow that path
6         else skip                        // Else, skip
7         fi left != nil
8
9         if right != nil then
10            call right::getSum(result)   // If we have a right child, follow that path
11        else skip                        // Else, skip
12         fi right != nil
13
14    method mirror()
15        left <=> right                   // Swap left and right children
16
17        if left = nil then skip
18        else call left::mirror()         // Recursively swap children if left != nil
19        fi left = nil
20
21        if right = nil then skip
22        else call right::mirror()        // Recursively swap children if right != nil
23        fi right = nil
```

Figure 2.7: Binary Tree node class (cont)

```
1   class Node
2       Node left
3       Node right
4       int value
5
6       method setValue(int newValue)
7           value ^= newValue
8
9       method insertNode(Node node, int nodeValue)
10          // Determine if we insert left or right
11          if nodeValue < value then
12              if left = nil & node != nil then
13                  // If open left node, store here
14                  left <=> node
15              else skip
16              fi left != nil & node = nil
17
18              if left != nil then
19                  // If current node has left, continue iterating
20                  call left::insertNode(node, nodeValue)
21              else skip
22              fi left != nil
23          else
24              if right = nil & node != nil then
25                  // If open right node spot, store here
26                  right <=> node
27              else skip
28              fi right != nil & node = nil
29
30              if right != nil then
31                  // If current node has, continue searching
32                  call right::insertNode(node, nodeValue)
33              else skip
34              fi right != nil
35          fi nodeValue < value
```

Figure 2.8: Binary Tree node class

2.7.3 Doubly Linked List

Finally, we present the reversible doubly linked list, shown in figures 2.10-2.12. A *cell* in a doubly linked list contains a reference to itself named *self*, a reference to its left and right neighbours, a data and an index field. As with the linked list and binary tree implementation the *DoubleLinkedList* class has a field referencing the head of the list and its *appendCell* method is identical to the one of the linked list.

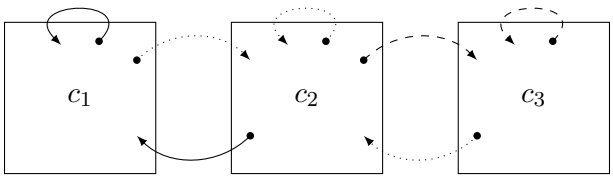


Figure 2.9: Multiple identical reference are needed for a doubly linked list implementation

This data structure is particularly interesting, as it, unlike the former two presented structures, cannot be expressed in ROOPL, as this requires multiple reference to objects, in order for an object to point to itself and to its left and right neighbours. Figure 2.9 shows the multiple

references needed for the doubly linked list implementation denoted by the three different arrow types.

```
1    class DoublyLinkedList
2        Cell head
3        int length
4
5        method appendCell(Cell cell)
6            if head = nil & cell != nil then
7                head <=> cell
8            else skip
9            fi head != nil & cell = nil
10
11           if head != nil then
12               call head::append(cell)
13           else skip
14           fi head != nil
15
16           length += 1
```

Figure 2.10: Doubly Linked List class

When we append a cell to the list, we first search recursively through the list until we are at the end. The new cell is then set as *right* of the current cell. A reference to the current self is created using the **copy** statement, and set as *left* of the new end of the list, thus resulting in the new cell being linked to list and now acting as end of the list.

```
1        class Cell
2        int data
3        int index
4        Cell left
5        Cell right
6        Cell self
7
8        method setData(int value)
9            data ^= value
10
11       method setIndex(int i)
12           index ^= i
13
14       method setLeft(Cell cell)
15           left <=> cell
16
17       method setRight(Cell cell)
18           right <=> cell
19
20       method setSelf(Cell cell)
21           self <=> cell
```

Figure 2.11: Doubly Linked List Cell class

The data structure could relatively easily be extended to work as a dynamic array. Currently each cell contains an *index* field, specifying their position in the list. If, say, we wanted to insert some new data at index n, without updating the existing value, but essentially squeezing in a new cell, we could add a method to the *DoublyLinkedList* class taking a data value and an index. When executing this method, we could iterate the list until we reach the cell with index n, construct a new *cell* instance, update required *left* and *right* pointers to insert the new cell at

```
1     method append(Cell cell)
2         if right = nil & cell != nil then   // If current cell does not have a right neighbour
3             right <=> cell                  // Set new cell as right neighbour of current cell
4
5             local Cell selfCopy = nil
6             copy Cell self selfCopy         // Copy reference to current cell
7             call right::setLeft(selfCopy)   // Set current as left of  right neighbour
8             delocal Cell selfCopy = nil
9
10            local int cellIndex = index + 1
11            call right::setIndex(cellIndex) // Set index in right neighbour of current
12            delocal int cellIndex = index + 1
13        else skip
14        fi right != nil & cell = nil
15
16        if right != nil then
17            call right::append(cell)        // Keep searching for empty right neighbour
18        else skip
19        fi right != nil
```

Figure 2.12: Doubly Linked List Cell class (cont)

the correct position, in such a way that the old cell at index n now is the new right neighbour of the cell and finally recursively iterating the list, incrementing the index of cells to the right of the new cell by one. In reverse, this would remove a cell from the list. If we want to update an existing value at a index, a similar technique could be used, where we iterate through the cells until we find the correct index. If we are given an index that is out of bounds in terms of the current length of the list, we could extend the tail on the list until reach a cell with the wanted index. When we are zero-clearing a value that is the furthest index, the inverse would apply, and a such we would zero-clear the cell, and the deallocate cells until we reach a cell which does not have a zero-cleared *data* field.

This extended doubly linked list would also allow lists of n-dimensional lists, as the type of the *data* field simply could be changed to, say, a *FooDoublyLinkedList*, resulting in an array of Foo arrays.

2.8 Type System

The type system of ROOPL++ expands on the type system of ROOPL presented by Haulund [11] and is analogously described by syntax-directed inference typing rules in the style of Winskel [28]. As ROOPL++ introduces two new types in form of *references* and arrays, a few ROOPL typing rules must be modified to accommodate these added types. For completeness all typing rules, including unmodified rules, are included in the following sections.

2.8.1 Preliminaries

The types in ROOPL++ are given by the following grammar:

$$\tau ::= \textbf{int} \mid c \in \text{ClassIDs} \mid r \in \text{ReferenceIDs} \mid i \in \text{IntegerArrayIDs} \mid o \in \text{ClassArrayIDs}$$

The type environment Π is a finite map pairing variables to types, which can be applied to an identifier x using the $\Pi(x)$ notation. Notation $\Pi' = \Pi[x \mapsto \tau]$ defines updates and creation of a new type environment Π' such that $\Pi'(x) = \tau$ and $\Pi'(y) = \Pi(y)$ if $x \neq y$, for some variable identifier x and y. The empty type environment is denoted as $[\,]$ and the function $vars : Expressions \rightarrow \text{VarIDs}$ is described by the following definition

$$\begin{aligned}
&\text{vars}(\bar{n}) && = \emptyset \\
&\text{vars}(\textbf{nil}) && = \emptyset \\
&\text{vars}(x) && = \{\ x\ \} \\
&\text{vars}(x[e]) && = \{\ x\ \} \cup \text{vars}(e) \\
&\text{vars}(e_1 \otimes e_2) && = \text{vars}(e_1) \cup \text{vars}(e_2).
\end{aligned}$$

The binary subtype relation $c_1 \prec: c_2$ is required for supporting subtype polymorphism and is defined as follows:

$$\begin{aligned}
&c_1 \prec: c_2 && \text{if } c_1 \text{ inherits from } c_2 \\
&c \prec: c && (reflexivity) \\
&c_1 \prec: c_3 && \text{if } c_1 \prec: c_2 \text{ and } c_2 \prec: c_3 \ (transitivity)
\end{aligned}$$

Furthermore, we formally define object models in such a way that inherited fields and methods are included, unless overridden by the derived fields. Therefore, we define Γ to be the class map of a program p, such that Γ is a finite map from class identifiers to tuples of methods and fields for the class p. Application of a class map Γ to some class cl is denoted as $\Gamma(cl)$. Construction of a class map is done through function *gen*, as shown in figure 2.13. Figure 2.14 defines the $fields$ and *methods* functions to determine these given a class. Set operation $\uplus$ defines method overloading by dropping base class methods if a similarly named method exists in the derived class. The definitions shown in Figure 2.13 and 2.14 are originally from [11].

$$\text{gen}\Big(\overbrace{cl_1,\ ...,\ cl_n}^{p}\Big) = \overbrace{\Big[\alpha(cl_1) \mapsto \beta(cl_1),\ ...,\ \alpha(cl_n) \mapsto \beta(cl_n)\Big]}^{\Gamma}$$

$$\alpha\Big(\textbf{class } c\ \cdots\Big) = c \qquad \beta(cl) = \Big(\text{fields}(cl),\ \text{methods}(cl)\Big)$$

Figure 2.13: Definition *gen* for constructing the finite class map Γ of a given program p, originally from [11]

$$\text{fields}(cl) = \begin{cases} \eta(cl) & \text{if } cl \sim\ [\textbf{class } c\ \cdots] \\ \eta(cl) \cup \text{fields}\Big(\alpha^{-1}(c')\Big) & \text{if } cl \sim [\textbf{class } c \textbf{ inherits } c'\ \cdots] \end{cases}$$

$$\text{methods}(cl) = \begin{cases} \delta(cl) & \text{if } cl \sim [\textbf{class } c\ \cdots] \\ \delta(cl) \uplus \text{methods}\Big(\alpha^{-1}(c')\Big) & \text{if } cl \sim [\textbf{class } c \textbf{ inherits } c'\ \cdots] \end{cases}$$

$$A\ \uplus B\ \stackrel{def}{=}\ A \cup \left\{ m \in B \;\middle|\; \nexists\ m'\Big(\zeta(m') = \zeta(m) \wedge m' \in A\Big) \right\}$$

$$\zeta\Big(\textbf{method } q\ (\cdots)\ s\Big) = q \qquad \eta\Big(\textbf{class } c\ \cdots\ \overbrace{t_1 f_1\ \cdots\ t_n f_n}^{fs}\ \cdots\Big) = fs$$

$$\delta\Big(\textbf{class } c\ \cdots \overbrace{\textbf{method } q_1\ (\cdots)\ s_1\ \cdots\ \textbf{method } q_n\ (\cdots)\ s_n}^{ms}\ \cdots\Big) = ms$$

Figure 2.14: Definition of fields and methods, originally from [11]

Finally, we formally define a link between arrays of a given type and other types. The function *arrayType*, defined in figure 2.15, is c if the passed array a is an array of class c instances.

$$\text{arrayType}(a) = \begin{cases} c & \text{if } a \in ClassArrayIDs \text{ and } a \text{ is a } c \text{ array} \\ \textbf{int} & \text{if } a \in i \end{cases}$$

Figure 2.15: Definition *arrayType* for mapping types of arrays to either class types or the integer type

2.8.2 Expressions

The type judgment

$$\frac{}{\Pi \vdash_{expr} \ e : \tau}$$

defines the type of expressions. The judgment reads as: under type environment Π, expression e has type τ.

$$\frac{}{\Pi \vdash_{expr} \ n : \textbf{int}} \text{ T-Con} \qquad \frac{\Pi(x) = \tau}{\Pi \vdash_{expr} \ x : \tau} \text{ T-Var} \qquad \frac{\tau \neq \textbf{int}}{\Pi \vdash_{expr} \ \textbf{nil} \ : \tau} \text{ T-Nil}$$

$$\frac{\Pi \vdash_{expr} \ e_1 : \textbf{int} \qquad \Pi \vdash_{expr} \ e_2 : \textbf{int}}{\Pi \vdash_{expr} \ e_1 \otimes e_2 : \textbf{int}} \text{ T-BinOpInt}$$

$$\frac{\Pi \vdash_{expr} \ e_1 : c \qquad \Pi \vdash_{expr} \ e_2 : c \qquad \otimes \in \{\textbf{=}, \textbf{!=}\}}{\Pi \vdash_{expr} \ e_1 \otimes e_2 : \textbf{int}} \text{ T-BinOpObj}$$

Figure 2.16: Typing rules for expressions in Roopl, originally from [11]

The original expression typing rules from Roopl are shown in figure 2.16. The type rules T-Con, T-Var and T-Nil defines typing of the simplest expressions. Numeric literals are of type **int**, typing of variable expressions depends on the type of the variable in the type environment and the **nil** literal is a non-integer type. All binary operations are defined for integers, while only equality-operators are defined for objects.

With the addition of the Roopl++ array type, we extend the expression typing rules with rule T-ArrElemVar which defines typing for array element variables, shown in figure 2.17.

$$\frac{\Pi(x) = \tau[\,] \qquad \Pi_{expr} \vdash \ e \ : \ \textbf{int}}{\Pi \vdash_{expr} \ x[e] : \tau} \text{ T-ArrElemVar}$$

Figure 2.17: Typing rule extension for the Roopl typing rules

2.8.3 Statements

The type judgment

$$\frac{}{\langle \Pi, \ c \rangle \ \vdash^{\Gamma}_{stmt} \ s}$$

defines well-typed statements. The judgment reads as under type environment Π within class c, statement s is well-typed with class map Γ.

$$\frac{x \notin \text{vars}(e) \qquad \Pi \vdash_{expr} \ e : \textbf{int} \qquad \Pi(x) = \textbf{int}}{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ x \odot= e} \text{ T-AssVar}$$

$$\frac{\Pi \vdash_{expr} \ e_1 : \textbf{int} \qquad \langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ s_1 \qquad \langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ s_2 \qquad \Pi \vdash_{expr} \ e_2 : \textbf{int}}{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ \textbf{if } e_1 \textbf{ then } s_1 \textbf{ else } s_2 \textbf{ fi } e_2} \text{ T-If}$$

$$\frac{\Pi \vdash_{expr} \ e_1 : \textbf{int} \qquad \langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ s_1 \qquad \langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ s_2 \qquad \Pi \vdash_{expr} \ e_2 : \textbf{int}}{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ \textbf{from } e_1 \textbf{ do } s_1 \textbf{ loop } s_2 \textbf{ until } e_2} \text{ T-Loop}$$

$$\frac{\langle \Pi[x \mapsto c'], c \rangle \vdash^{\Gamma}_{stmt} \ s}{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ \textbf{construct } c' \ x \quad s \quad \textbf{destruct } x} \text{ T-ObjBlock} \qquad \frac{}{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ \textbf{skip}} \text{ T-Skip}$$

$$\frac{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ s_1 \qquad \langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ s_2}{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ s_1 \ s_2} \text{ T-Seq} \qquad \frac{\Pi(x_1) = \Pi(x_2)}{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ x_1 \textbf{ <=> } x_2} \text{ T-SwpVar}$$

$$\frac{\begin{array}{c}\Gamma(\Pi(c)) = \Big(fields, \ methods\Big) \quad \Big(\textbf{method } q(t_1 \ y_1, \ ..., \ t_n \ y_n) \ s\Big) \in methods \\ \{x_1, \ ..., \ x_n\} \cap fields = \emptyset \qquad i \neq j \implies x_i \neq x_j \qquad \Pi(x_1) \prec: t_1 \ \cdots \ \Pi(x_n) \prec: t_n\end{array}}{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ \textbf{call } q(x_1, \ ..., \ x_n)} \text{ T-Call}$$

$$\frac{\begin{array}{c}\Gamma(\Pi(x_0)) = \Big(fields, \ methods\Big) \quad \Big(\textbf{method } q(t_1 \ y_1, \ ..., \ t_n \ y_n) \ s\Big) \in methods \\ i \neq j \implies x_i \neq x_j \qquad \Pi(x_1) \prec: t_1 \ \cdots \ \Pi(x_n) \prec: t_n\end{array}}{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ \textbf{call } x_0 :: q(x_1, \ ..., \ x_n)} \text{ T-CallO}$$

$$\frac{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ \textbf{call } q(x_1, \ ..., \ x_n)}{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ \textbf{uncall } q(x_1, \ ..., \ x_n)} \text{ T-Uc} \qquad \frac{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ \textbf{call } x_0 :: q(x_1, \ ..., \ x_n)}{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \ \textbf{uncall } x_0 :: q(x_1, \ ..., \ x_n)} \text{ T-Uco}$$

Figure 2.18: Typing rules for statements in Roopl, originally from [11]

Typing rule T-AssVar defines variable assignments for an integer variable and an integer expression result, given that the variable x does not occur in the expression e.

The type rules T-If and T-Loop defines reversible conditionals and loops as known from Janus, where entry and exit conditions are integers and branch and loop statements are well-typed statements.

The object block, introduced in Roopl, is only well-typed if its body statement is well-typed.

The **skip** statement is always well-typed, while a sequence of statements are well-typed if each of the provided statements are. Variable **swap** statements are well-typed if both operands are of the same type under type environment Π.

As with Roopl, type correctness of local method invocation is defined in rule T-Call iff:

- The number of arguments matches the method arity
- No class fields are present in the arguments passed to the method (To prevent irreversible updates)
- The argument list contains unique elements
- Each argument is a subtype of the type of the equivalent formal parameter.

For foreign method invocations, typing rule T-CallO. A foreign method invocation is well-typed using the same rules as for T-Call besides having no restrictions on class fields parameters in the arguments, but an added rule stating that the callee object x_0 must not be passed as an argument.

The typing rules T-UC and T-UCO defines uncalling of methods in terms of their respective inverse counterparts.

$$\Pi(x) = \textbf{int}[\,] \quad \Pi \vdash_{expr} e_1 \ : \ \textbf{int}$$

$$\frac{\Big(\ x\ \cup \text{vars}(e_1)\Big) \cap \text{vars}(e_2) = \emptyset \qquad \Pi \vdash_{expr}\ e_2 : \textbf{int}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt}\ \ x[e_1]\ \odot\!\!= e_2}\ \text{T-ArrElemAss}$$

$$\frac{\Pi(x) = c'}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt}\ \ \textbf{new}\ c'\ x}\ \text{T-ObjNew} \qquad \frac{\Pi(x) = c'}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt}\ \ \textbf{delete}\ c'\ x}\ \text{T-ObjDlt}$$

$$\frac{\text{arrayType}(a) \in \Big\{\text{classIDs}, \textbf{int}\Big\} \qquad \Pi \vdash_{expr}\ e = \textbf{int} \qquad \Pi(x) = a[\,]}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt}\ \ \textbf{new}\ a[e]\ x}\ \text{T-ArrNew}$$

$$\frac{\text{arrayType}(a) \in \Big\{\text{classIDs}, \textbf{int}\Big\} \qquad \Pi \vdash_{expr}\ e = \textbf{int} \qquad \Pi(x) = a[\,]}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt}\ \ \textbf{delete}\ a[e]\ x}\ \text{T-ArrDlt}$$

$$\frac{\Pi(x) = c' \qquad \Pi(x') = c'}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt}\ \ \textbf{copy}\ c'\ x\ x'}\ \text{T-Cp} \qquad \frac{\Pi(x) = c' \qquad \Pi(x') = c'}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt}\ \ \textbf{uncopy}\ c'\ x\ x'}\ \text{T-Ucp}$$

$$\frac{\langle \Pi, c\rangle \vdash_{expr}\ e_1\ :\ c' \qquad \langle \Pi[x \mapsto c'], c\rangle \vdash^{\Gamma}_{stmt}\ \ s \qquad \langle \Pi, c\rangle \vdash_{expr}\ e_2\ :\ c'}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt}\ \ \textbf{local}\ c'\ x = e_1 \quad s \quad \textbf{delocal}\ c'\ x = e_2}\ \text{T-LocalBlock}$$

Figure 2.19: Typing rules extensions for statements in Roopl++

Figure 2.19 shows the typing rules for the extensions made to Roopl in Roopl++, covering the **new**/**delete** and **copy**/**uncopy** statements for objects and arrays and local blocks.

The typing rule T-ArrElemAss defines assignment to integer array element variables, and is well-typed when the type of array x is **int**, the variable $x[e_1]$ is not present in the right-hand

side of the statement, no variables in e_1 exist in e_2 and both expressions e_1 and e_2 evaluates to integers.

The T-ObjNew and T-ObjDlt rules define well-typed **new** and **delete** statements for dynamically lifetimed objects. The **new** statement is well-typed, as long as $c' \in$ classIDs and the variable x is of type c' under type environment Π and **delete** is also well-typed if the type of x under type environment Π is equal to c'.

The T-ArrNew and T-ArrDlt rules define well-type **new** and **delete** statement for Roopl++ arrays. The **new** statement is well-typed, if the type of the array either is a classID or **int**, the length expression evaluates to an integer and x is of of type $a[\,]$ under the type environment Π, and **delete** is well-typed if the type of the array is either a classID or **int**, the length expression evaluates to an integer and x is equal to the array type a.

Typing rules T-Cp and T-Ucp define well-typed reference copy and un-copying statements. A well-typed **copy** or **uncopy** statement requires that the types of x and x' both are c' under type environment Π

The rule T-LocalBlock defines well-typed local blocks. A local block is well-typed if its two expression e_1 and e_2 are well-typed and its body statement s is well-typed.

2.8.4 Programs

As with Roopl, a Roopl++ program is well-typed if all of its classes and their respective methods are well-typed and if there exists a nullary main method. Figure 2.20 shows the typing rules for class methods, classes and programs.

$$\frac{\langle \Pi[x_1 \mapsto t_1,\ ...,\ x_n \mapsto t_n],\ c\rangle \vdash^{\Gamma}_{stmt}\ \ s}{\langle \Pi, c\rangle \vdash^{\Gamma}_{meth}\ \ \textbf{method}\ q(t_1 x_1,\ ...,\ t_n x_n)\ s}\ \text{T-Method}$$

$$\frac{\Gamma(c) = \Big(\overbrace{\{\langle t_1, f_1\rangle,\ ...,\ \langle t_i, f_i\rangle\}}^{fields},\ \overbrace{\{m_1,\ ...,\ m_n\}}^{methods}\Big) \qquad \Pi = [f_1 \mapsto t_1,\ ...,\ f_n \mapsto t_n] \qquad \langle \Pi, c\rangle \vdash^{\Gamma}_{meth}\ m_1 \quad \cdots \quad \langle \Pi, c\rangle \vdash^{\Gamma}_{meth}\ m_n}{\vdash^{\Gamma}_{class}\ c}\ \text{T-Class}$$

$$\frac{\Big(\textbf{method main}\ ()\ s\Big) \in \bigcup_{i=1}^{n} \text{methods}(c_i) \qquad \Gamma = \text{gen}(c_1,\ ...,\ c_n) \qquad \vdash^{\Gamma}_{class}\ c_1 \quad \cdots \quad \vdash^{\Gamma}_{class}\ c_n}{\vdash_{prog}\ c_1\ \cdots\ c_n}\ \text{T-Prog}$$

Figure 2.20: Typing rules for class methods, classes and programs, originally from [11]

2.9 Language Semantics

The following sections contain the operational semantics of ROOPL++, as specified by syntax-directed inference rules.

2.9.1 Preliminaries

We define l to be a location. We define a location for integer variables to bind to a single location in program memory and a vector of memory locations for object and array variables, where the vector is the size of the object or array. A memory location is in the set of non-negative integers, $\mathbb{N}_0$. An environment γ is a partial function mapping variables to memory locations. A store μ is a partial function mapping memory locations to values. An object is a tuple of a class name and an environment mapping fields to memory locations. A value is either an integer, an object, a location or a vector of locations.

Applications of environments γ and stores μ are analogous to the type environment Γ, defined in section 2.8.1.

$$
\begin{aligned}
l \in \text{Locs} &= \mathbb{N}_0 \\
\gamma \in \text{Envs} &= \text{VarIDs} \rightharpoonup \text{Locs} \\
\mu \in \text{Stores} &= \text{Locs} \rightharpoonup \text{Values} \\
\text{Objects} &= \Big\{ \langle c_f,\ \gamma_f \rangle \mid c_f \in \text{ClassIDs} \ \wedge \ \gamma_f \in \text{Envs} \Big\} \\
v \in \text{Values} &= \mathbb{Z} \cup \text{Objects} \ \cup \text{Locs} \cup [\text{Locs}]
\end{aligned}
$$

Figure 2.21: Semantic values, originally from [11]

2.9.2 Expressions

The judgment:

$$\langle \gamma, \mu \rangle \vdash_{expr} \ e \Rightarrow v$$

defines the meaning of expressions. We say that under environment γ and store μ, expression e evaluates to value v.

$$\frac{}{\langle \gamma, \mu \rangle \vdash_{expr} \ n \Rightarrow \bar{n}} \textsc{Con} \qquad \frac{}{\langle \gamma, \mu \rangle \vdash_{expr} \ x \Rightarrow \mu\Big(\gamma(x)\Big)} \textsc{Var} \qquad \frac{}{\langle \gamma, \mu \rangle \vdash_{expr} \ \texttt{nil} \Rightarrow 0} \textsc{Nil}$$

$$\frac{\langle \gamma, \mu \rangle \vdash_{expr} \ e_1 \Rightarrow v_1 \qquad \langle \gamma, \mu \rangle \vdash_{expr} \ e_2 \Rightarrow v_2 \qquad [\![\otimes]\!](v_1, v_2) = v}{\langle \gamma, \mu \rangle \vdash_{expr} \ e_1 \otimes e_2 \Rightarrow v} \textsc{BinOp}$$

Figure 2.22: Semantic inference rules for expressions, originally from [11]

As shown in figure 2.22, expression evaluation has no effects on the store. Logical values are represented by *truthy* and *falsy* values of any non-zero value and zero respectively. The evaluation of binary operators is presented in figure 2.24.

$$\frac{\langle \gamma, \mu \rangle \vdash_{expr} \; e \Rightarrow v \qquad \gamma(x) = l \qquad \mu(l)[v] = l' \qquad \mu(l') = w}{\langle \gamma, \mu \rangle \vdash_{expr} \; x[e] \Rightarrow w} \textsc{ArrElem}$$

Figure 2.23: Extension to the semantic inference rules for expression in RoopL++

For RoopL++, we extend the expression ruleset with a single rule for array element variables shown in figure 2.23. As with the expressions inference rules in RoopL, this extension has no effect on the store.

$$\begin{aligned}
[\![\texttt{+}]\!](v_1, v_2) &= v_1 + v_2 & [\![\texttt{\%}]\!](v_1, v_2) &= v_1 \bmod v_2 \\
[\![\texttt{-}]\!](v_1, v_2) &= v_1 - v_2 & [\![\texttt{\&}]\!](v_1, v_2) &= v_1 \wedge v_2 \quad , \textit{bitwise} \\
[\![\texttt{*}]\!](v_1, v_2) &= v_1 \times v_2 & [\![\texttt{|}]\!](v_1, v_2) &= v_1 \vee v_2 \quad , \textit{bitwise} \\
[\![\texttt{/}]\!](v_1, v_2) &= v_1 \div v_2 & [\![\texttt{\^{}}]\!](v_1, v_2) &= v_1 \oplus v_2 \\
[\![\texttt{\&\&}]\!](v_1, v_2) &= \begin{cases} 0 & \text{if } v_1 = 0 \vee v_2 = 0 \\ 1 & \text{otherwise} \end{cases} & [\![\texttt{<=}]\!](v_1, v_2) &= \begin{cases} 0 & \text{if } v_1 \leq v_2 \\ 1 & \text{otherwise} \end{cases} \\
[\![\texttt{||}]\!](v_1, v_2) &= \begin{cases} 0 & \text{if } v_1 = v_2 = 0 \\ 1 & \text{otherwise} \end{cases} & [\![\texttt{>=}]\!](v_1, v_2) &= \begin{cases} 0 & \text{if } v_1 \geq v_2 \\ 1 & \text{otherwise} \end{cases} \\
[\![\texttt{<}]\!](v_1, v_2) &= \begin{cases} 1 & \text{if } v_1 < v_2 \\ 0 & \text{otherwise} \end{cases} & [\![\texttt{=}]\!](v_1, v_2) &= \begin{cases} 0 & \text{if } v_1 = v_2 \\ 1 & \text{otherwise} \end{cases} \\
[\![\texttt{>}]\!](v_1, v_2) &= \begin{cases} 1 & \text{if } v_1 > v_2 \\ 0 & \text{otherwise} \end{cases} & [\![\texttt{!=}]\!](v_1, v_2) &= \begin{cases} 0 & \text{if } v_1 \neq v_2 \\ 1 & \text{otherwise} \end{cases}
\end{aligned}$$

Figure 2.24: Definition of binary expression operator evaluation, originally from [11]

2.9.3 Statements

The judgment

$$\gamma \vdash^{\Gamma}_{stmt} \; s : \mu \rightleftharpoons \mu'$$

defines the meaning of statements. We say that under environment γ, statement s with class map Γ reversibly transforms store μ to store μ'. Figure 2.25a, 2.25b and 2.25c defines the operational semantics of RoopL++.

The following semantic rules have been simplified from the original RoopL semantics [11] to better accommodate the extended language.

The inference rule Skip defines the operational semantics of **skip** statements and has no effects on the store μ.

$$\frac{}{\gamma \vdash^{\Gamma}_{stmt} \ \textbf{skip} : \mu \rightleftharpoons \mu} \textsc{Skip}$$

$$\frac{\gamma \vdash^{\Gamma}_{stmt} \ s_1 : \mu \rightleftharpoons \mu' \qquad \gamma \vdash^{\Gamma}_{stmt} \ s_2 : \mu' \rightleftharpoons \mu''}{\gamma \vdash^{\Gamma}_{stmt} \ s_1 \ s_2 : \mu \rightleftharpoons \mu''} \textsc{Seq}$$

$$\frac{\langle \gamma, \mu \rangle \vdash^{\Gamma}_{stmt} \ e \Rightarrow v \qquad [\![\odot]\!]\Big(\mu\Big(\gamma(x)\Big), v\Big) = v'}{\gamma \vdash^{\Gamma}_{stmt} \ x \odot = e \ : \ \mu \rightleftharpoons \mu[\gamma(x) \mapsto v']} \textsc{AssVar}$$

$$\frac{\mu\Big(\gamma(x_1)\Big) = v_1 \qquad \mu\Big(\gamma(x_2)\Big) = v_2}{\gamma \vdash^{\Gamma}_{stmt} \ x_1 \ \textbf{<=>} \ x_2 \ : \ \mu \rightleftharpoons \mu[\gamma(x_1) \mapsto v_2, \ \gamma(x_2) \mapsto v_1]} \textsc{SwpVar}$$

$$\frac{\langle \gamma, \mu \rangle \vdash^{\Gamma}_{expr} \ e_1 \not\Rightarrow 0 \qquad \gamma \vdash^{\Gamma}_{stmt} \ s_1 : \mu \rightleftharpoons \mu' \qquad \gamma \vdash^{\Gamma}_{loop} \ (e_1, s_1, s_2, e_2) : \mu' \rightleftharpoons \mu''}{\gamma \vdash^{\Gamma}_{stmt} \ \textbf{from} \ e_1 \ \textbf{do} \ s_1 \ \textbf{loop} \ s_2 \ \textbf{until} \ e_2 \ : \ \mu \rightleftharpoons \mu''} \textsc{LoopMain}$$

$$\frac{\langle \gamma, \mu \rangle \vdash^{\Gamma}_{expr} \ e_2 \not\Rightarrow 0}{\gamma \vdash^{\Gamma}_{loop} \ (e_1, s_1, s_2, e_2) \ : \ \mu \rightleftharpoons \mu} \textsc{LoopBase}$$

$$\frac{\begin{array}{cc} \langle \gamma, \mu \rangle \vdash^{\Gamma}_{expr} e_2 \Rightarrow 0 & \gamma \vdash^{\Gamma}_{stmt} s_2 : \mu \rightleftharpoons \mu' \\ \langle \gamma, \mu' \rangle \vdash^{\Gamma}_{expr} e_1 \Rightarrow 0 \qquad \gamma \vdash^{\Gamma}_{stmt} s_1 : \mu' \rightleftharpoons \mu'' & \gamma \vdash^{\Gamma}_{loop} (e_1, s_1, s_2, e_2) : \mu'' \rightleftharpoons \mu''' \end{array}}{\gamma \vdash^{\Gamma}_{loop} \ (e_1, s_1, s_2, e_2) \ : \ \mu \rightleftharpoons \mu'''} \textsc{LoopRec}$$

$$\frac{\langle \gamma, \mu \rangle \vdash^{\Gamma}_{expr} e_1 \not\Rightarrow 0 \qquad \gamma \vdash^{\Gamma}_{stmt} s_1 : \mu \rightleftharpoons \mu' \qquad \langle \gamma, \mu' \rangle \vdash^{\Gamma}_{expr} e_2 \not\Rightarrow 0}{\gamma \vdash^{\Gamma}_{stmt} \ \textbf{if} \ e_1 \ \textbf{then} \ s_1 \ \textbf{else} \ s_2 \ \textbf{fi} \ e_2 \ : \ \mu \rightleftharpoons \mu'} \textsc{IfTrue}$$

$$\frac{\langle \gamma, \mu \rangle \vdash^{\Gamma}_{expr} e_1 \Rightarrow 0 \qquad \gamma \vdash^{\Gamma}_{stmt} s_1 : \mu \rightleftharpoons \mu' \qquad \langle \gamma, \mu' \rangle \vdash^{\Gamma}_{expr} e_2 \Rightarrow 0}{\gamma \vdash^{\Gamma}_{stmt} \ \textbf{if} \ e_1 \ \textbf{then} \ s_1 \ \textbf{else} \ s_2 \ \textbf{fi} \ e_2 \ : \ \mu \rightleftharpoons \mu'} \textsc{IfFalse}$$

Figure 2.25a: Semantic inference rules for statements, modified from [11]

Rule Seq defines statement sequences where the store potentially is updated between each statement execution.

Rule AssVar defines reversible assignment in which variable identifier x under environment γ is mapped to the value v' resulting in an updated store μ'. For variable swapping SwpVar defines how value mappings between two variables are exchanged in the updated store.

For loops and conditionals, Rules LoopMain, LoopBase and LoopRec define the meaning of loop statements and IfTrue and IfFalse, similarly to the operational semantics of Janus, as

$$\frac{\gamma(this) = l \quad \mu(l) = l' \quad \mu(l') = \Big\langle c, (l_1,\ \ldots, l_m)\Big\rangle \quad \gamma(y_i) = l'_i \qquad \Gamma(c) = \Big\langle \overbrace{(x_1,\ \ldots, x_m)}^{fields}, \overbrace{(\ldots, \textbf{method}\ q(t_1 z_1,\ \ldots, t_l z_k)\ s,\ \ldots)}^{methods} \Big\rangle \qquad \gamma' = [this \mapsto l, x_1 \mapsto l_1,\ \ldots, l_m \mapsto v_m, z_1 \mapsto l'_1, \ldots, z_k \mapsto l'_k] \qquad \gamma' \vdash^{\Gamma}_{stmt}\ s\ : \mu \rightleftharpoons \mu'}{\gamma \vdash^{\Gamma}_{stmt}\ \textbf{call}\ q(y_1,\ ...,\ y_n)\ :\ \mu \rightleftharpoons \mu'}\ \textsc{Call}$$

$$\frac{\gamma \vdash^{\Gamma}_{stmt}\ \textbf{call}\ q(y_1,\ ...,\ y_n)\ :\ \mu' \rightleftharpoons \mu}{\gamma \vdash^{\Gamma}_{stmt}\ \textbf{uncall}\ q(y_1,\ ...,\ y_n)\ :\ \mu \rightleftharpoons \mu'}\ \textsc{Uncall}$$

$$\frac{\gamma(x_0) = l \quad \mu(l) = l' \quad \mu(l') = \Big\langle c, (l_1,\ \ldots, l_m)\Big\rangle \quad \gamma(y_i) = l'_i \qquad \Gamma(c) = \Big\langle \overbrace{(x_1,\ \ldots, x_m)}^{fields}, \overbrace{(\ldots, \textbf{method}\ q(t_1 z_1,\ \ldots, t_l z_k)\ s,\ \ldots)}^{methods} \Big\rangle \qquad \gamma' = [this \mapsto l, x_1 \mapsto l_1,\ \ldots, l_m \mapsto v_m, z_1 \mapsto l'_1, \ldots, z_k \mapsto l'_k] \qquad \gamma' \vdash^{\Gamma}_{stmt}\ s\ : \mu \rightleftharpoons \mu'}{\gamma \vdash^{\Gamma}_{stmt}\ \textbf{call}\ x_0 :: q(y_1,\ ...,\ y_n)\ :\ \mu \rightleftharpoons \mu'}\ \textsc{CallObj}$$

$$\frac{\gamma \vdash^{\Gamma}_{stmt}\ \textbf{call}\ x_0 :: q(y_1,\ ...,\ y_n)\ :\ \mu' \rightleftharpoons \mu}{\gamma \vdash^{\Gamma}_{stmt}\ \textbf{uncall}\ x_0 :: q(y_1,\ ...,\ y_n)\ :\ \mu \rightleftharpoons \mu'}\ \textsc{ObjUncall}$$

$$\frac{\Gamma(c) = \Big\langle \overbrace{(x_1,\ \ldots, x_m)}^{fields}, methods \Big\rangle \quad \gamma' = \gamma[x \mapsto l_0] \quad l_0 \notin \text{dom}(\mu)\ \ldots\ l_m \notin \text{dom}(\mu) \qquad \mu' = \mu\Big[\gamma'(x) \mapsto l_0, l_0 \mapsto \Big\langle c, (l_1,\ \ldots, 1_m)\Big\rangle, l_1 \mapsto 0,\ \ldots, l_m \mapsto 0\Big] \quad \gamma' \vdash^{\Gamma}_{stmt}\ s\ :\ \mu' \rightleftharpoons \mu'' \qquad \mu'' = \mu'''\Big[\gamma'(x) \mapsto l_0, l_0 \mapsto \Big\langle c, (l_1,\ \ldots, 1_m)\Big\rangle, l_1 \mapsto 0,\ \ldots, l_m \mapsto 0\Big]}{\gamma \vdash^{\Gamma}_{stmt}\ \textbf{construct}\ c\ x \quad s \quad \textbf{destruct}\ x\ :\ \mu \rightleftharpoons \mu'''}\ \textsc{ObjBlock}$$

Figure 2.25b: Semantic inference rules for statements, modified from [11] (cont)

presented in [29]. LoopMain is entered if e_1 is true and each iteration enters LoopRec until e_2 is false, in which case LoopBase is executed. Similarly, if e_1 and e_2 are true, rule IfTrue is entered, executing the then-branch of the conditional. If e_1 and e_2 are false, the IfFalse rule is executed and the else-branch is executed.

Rules Call, Uncall, CallObj and UncallObj respectively define local and non-local method invocations. For local methods, method q in current class c should be of arity n matching the number of arguments. The updated store μ' is obtained after statement body execution in the object environment. As local uncalling is the inverse of local calling, the direction of execution is simply reversed, and as such the input store a **call** statement serves as the output store of the

$$\frac{\langle \gamma, \mu \rangle \vdash^{\Gamma}_{stmt} \ e_1 \Rightarrow v_1 \quad \langle \gamma, \mu \rangle \vdash^{\Gamma}_{stmt} \ e_2 \Rightarrow v_2 \quad \gamma(x) = l \quad \mu(l)[v_1] = l' \quad \mu(l') = w \quad [\![\odot]\!](w, v_2) = w'}{\gamma \vdash^{\Gamma}_{stmt} \ x[e_1] \odot\!= e_2 \ : \ \mu \rightleftharpoons \mu[l' \mapsto w']} \textsc{AssArrElemVar}$$

$$\frac{\Gamma(c) = \left\langle \overbrace{(x_1, \ \ldots, x_m)}^{fields}, methods \right\rangle \quad \gamma(x) = l \quad l_0 \notin \ \mathrm{dom}(\mu) \ \ldots \ l_m \notin \ \mathrm{dom}(\mu)}{\gamma \vdash^{\Gamma}_{stmt} \ \textbf{new} \ c \ x \ : \ \mu[l \mapsto 0] \rightleftharpoons \mu\left[l \mapsto l_0, l_0 \mapsto \left\langle c, (l_1, \ \ldots, 1_m) \right\rangle, l_1 \mapsto 0, \ \ldots, l_m \mapsto 0\right]} \textsc{ObjNew}$$

$$\frac{\langle l, \gamma \rangle \vdash^{\Gamma}_{stmt} \ \textbf{new} \ c \ x \ : \ \mu' \rightleftharpoons \mu}{\gamma \vdash^{\Gamma}_{stmt} \ \textbf{delete} \ c \ x \ : \ \mu \rightleftharpoons \mu'} \textsc{ObjDelete}$$

$$\frac{\langle \gamma, \mu \rangle \vdash^{\Gamma}_{stmt} \ e \Rightarrow n \quad \gamma(x) = l \quad \mu(l) = 0 \quad l' \notin \ \mathrm{dom}(\mu)}{\gamma \vdash^{\Gamma}_{stmt} \ \textbf{new} \ a[e] \ x \ : \ \mu \rightleftharpoons \mu[l \mapsto l', l' \mapsto 0^n]} \textsc{ArrNew}$$

$$\frac{\dfrac{\langle l, \gamma \rangle \vdash^{\Gamma}_{stmt} \ \textbf{new} \ a[e] \ x \ : \ \mu' \rightleftharpoons \mu}{\gamma \vdash^{\Gamma}_{stmt} \ \textbf{delete} \ a[e] \ x \ : \ \mu \rightleftharpoons \mu'} \textsc{ArrDelete} \quad \gamma(x) = l \quad \gamma(x') = l' \quad \mu(l) = v}{\gamma \vdash^{\Gamma}_{stmt} \ \textbf{copy} \ c \ x \ x' \ : \ \mu[l' \mapsto 0] \leftrightharpoons \mu[l' \mapsto v]} \textsc{Copy}$$

$$\frac{\langle l, \gamma \rangle \vdash^{\Gamma}_{stmt} \ \textbf{copy} \ c \ x \ x' \ : \ \mu' \leftrightharpoons \mu}{\gamma \vdash^{\Gamma}_{stmt} \ \textbf{uncopy} \ c \ x \ x' \ : \ \mu \rightleftharpoons \mu'} \textsc{Uncopy}$$

$$\frac{\langle \gamma, \mu \rangle \vdash^{\Gamma}_{stmt} \ e_1 \Rightarrow v_1 \qquad \langle \gamma, \mu' \rangle \vdash^{\Gamma}_{stmt} \ e_2 \Rightarrow v_2 \quad r \notin \ \mathrm{dom}(\mu) \qquad \gamma[x \mapsto r] \vdash^{\Gamma}_{stmt} \ s : \mu[r \mapsto v_1] \rightleftharpoons \mu'[r \mapsto v_2]}{\gamma \vdash^{\Gamma}_{stmt} \ \textbf{local} \ c \ x = e_1 \quad s \quad \textbf{delocal} \ x = e_2 \ : \ \mu \rightleftharpoons \mu'} \textsc{LocalBlock}$$

Figure 2.25c: Extension to the semantic inference rules for statements in Roopl++

uncall statement, similarly to techniques presented in [31, 29].

The statically scoped object blocks are defined in rule ObjBlock. The operation semantics of these blocks are similar to **local**-blocks from Janus. We add the reference x to a new environment and afterwards map the location of x to the object tuple at location l_0, containing the locations of all object fields, all of which, along with l_0, must be unused in μ. The result store μ'' is obtained after executing the body statement s in store μ' mapping x to object reference at l_0, as long as all object fields are zero-cleared in μ''' afterwards. If any of these conditions fail, the object block statement is undefined.

Figure 2.25c shows the extensions to the semantics of Roopl with rules for **new**/**delete** and

copy/**uncopy** statements, array element assignment and local blocks.

Rule AssArrElemVar defines reversible assignment to array elements. After evaluating expressions e_1 to v_1 and e_2 to v_2, the value at the location of variable $x[v_1]$ under environment γ is mapped to the value v_3 resulting in an updated store μ'.

Dynamic object construction and destruction is defined by rules ObjNew and ObjDelete. For construction, x must be bound to a location l. We then make location l point to a new pair consisting of the class name and a vector of m new locations mapping object fields to locations. For destruction, x is still bound to l return l to a null pointer. As with object blocks, it is the program itself responsible for zero-clearing object fields before destruction. If the object fields are not zero-cleared, the ObjDelete statement is undefined.

Array construction and destruction is very similar to object construction and destruction. The major difference is we bind the location to a vector of size equal to the evaluated expression result. For deletion, we return the location of x to a null pointer and remove the binding to the vector from the store.

Object and array referencing is defined by rules Copy and Uncopy. A reference is created and a new store μ' obtained by mapping x' to the reference r which x current maps to, if c matches the tuple mapped to the location l. A reference is removed and a new store μ' obtained if x and x' maps to the same reference r and x' then is removed from the store.

Local blocks are as previously mentioned, semantically similar to object blocks, where the memory location of variable x is mapped to an unused reference r in the store μ. Before body statement execution, we let r bind to the evaluated value of e_1, v_1. The result store after body statement execution, μ' must have r mapped to the expression value of e_2, v_2. r is then zero-cleared using the value of expression evaluation and becomes unused again.

2.9.4 Programs

The judgment

$$\vdash_{prog} p \Rightarrow \sigma$$

defines the meaning of programs. The class p containing the main method is instantiated and the main function is executed with the partial function σ as the result, mapping variable identifiers to values, correlating to the class fields of the main class.

$$\frac{\begin{array}{c}\Gamma = \text{gen}(c_1,\ ...,\ c_n) \qquad \Gamma(c_1) = \Big(\overbrace{\{\langle t_1, f_1\rangle,\ ...,\ \langle t_n, f_n\rangle\}}^{fields},\ methods\Big) \\ \Big(\textbf{method main ()}\ s\Big) \in methods \quad \gamma = [f_1 \mapsto 1,\ ...,\ f_i \mapsto i] \\ \mu = [1 \mapsto 0,\ ...,\ i \mapsto 0,\ this \mapsto i+1, i+1 \mapsto \langle c_1, \gamma\rangle] \qquad \gamma \vdash^{\Gamma}_{stmt}\ s\ :\ \mu \rightleftharpoons \mu'\end{array}}{\vdash_{prog}\ c_1\ \cdots\ c_n \Rightarrow (\gamma, \mu')}\ \textsc{Main}$$

Figure 2.26: Semantic inference rules for programs, originally from [11]

As with Roopl programs, the fields of the main method in the main class c are bound in a new environment, starting at memory address 1, as 0 is reserved for **nil**. The fields are zero-initialized

in the new store μ and address $i + 1$ which maps to the new instance of c. After body execution, store μ' is obtained.

2.10 Program Inversion

In order to truly show that ROOPL++ in fact is a reversible language, we must demonstrate and prove local inversion of statements is possible, such that any program written in ROOPL++, regardless of context, can be executed in reverse. Haulund presented a statement inverter for ROOPL in [11], which maps statements to their inverse counterparts. Figure 2.27 shows the statement inverter, extended with the new ROOPL++ statements for construction/destruction and referencing copying/copy removal.

$$
\begin{aligned}
&\mathcal{I}[\![\textbf{skip}]\!] = \textbf{skip} && \mathcal{I}[\![s_1\ s_2]\!] = \mathcal{I}[\![s_2]\!]\ \mathcal{I}[\![s_1]\!] \\
&\mathcal{I}[\![x\ \texttt{+=}\ e]\!] = x\ \texttt{-=}\ e && \mathcal{I}[\![x\ \texttt{-=}\ e]\!] = x\ \texttt{+=}\ e \\
&\mathcal{I}[\![x\ \texttt{\^{}=}\ e]\!] = x\ \texttt{\^{}=}\ e && \mathcal{I}[\![x\ \texttt{<=>}\ e]\!] = x\ \texttt{<=>}\ e \\
&\mathcal{I}[\![x[e_1]\ \texttt{+=}\ e_2]\!] = x[e_1]\ \texttt{-=}\ e_2 && \mathcal{I}[\![x[e_1]\ \texttt{-=}\ e_2]\!] = x[e_1]\ \texttt{+=}\ e_2 \\
&\mathcal{I}[\![x[e_1]\ \texttt{\^{}=}\ e_2]\!] = x[e_1]\ \texttt{\^{}=}\ e_2 && \mathcal{I}[\![x[e_1]\ \texttt{<=>}\ e_2]\!] = x[e_1]\ \texttt{<=>}\ e_2 \\
&\mathcal{I}[\![\textbf{new}\ c\ x]\!] = \textbf{delete}\ c\ x && \mathcal{I}[\![\textbf{copy}\ c\ x\ x']\!] = \textbf{uncopy}\ c\ x\ x' \\
&\mathcal{I}[\![\textbf{delete}\ c\ x]\!] = \textbf{new}\ c\ x && \mathcal{I}[\![\textbf{uncopy}\ c\ x\ x']\!] = \textbf{copy}\ c\ x\ x' \\
&\mathcal{I}[\![\textbf{call}\ q(\ldots)]\!] = \textbf{uncall}\ q(\ldots) && \mathcal{I}[\![\textbf{call}\ x :: q(\ldots)]\!] = \textbf{uncall}\ x :: q(\ldots) \\
&\mathcal{I}[\![\textbf{uncall}\ q(\ldots)]\!] = \textbf{call}\ q(\ldots) && \mathcal{I}[\![\textbf{uncall}\ x :: q(\ldots)]\!] = \textbf{call}\ x :: q(\ldots) \\
&\mathcal{I}[\![\textbf{if}\ e_1\ \textbf{then}\ s_1\ \textbf{else}\ s_2\ \textbf{fi}\ e_2]\!] && = \textbf{if}\ e_1\ \textbf{then}\ \mathcal{I}[\![s_1]\!]\ \textbf{else}\ \mathcal{I}[\![s_2]\!]\ \textbf{fi}\ e_2 \\
&\mathcal{I}[\![\textbf{from}\ e_1\ \textbf{do}\ s_1\ \textbf{loop}\ s_2\ \textbf{until}\ e_2]\!] && = \textbf{from}\ e_1\ \textbf{do}\ \mathcal{I}[\![s_1]\!]\ \textbf{loop}\ \mathcal{I}[\![s_2]\!]\ \textbf{until}\ e_2 \\
&\mathcal{I}[\![\textbf{construct}\ c\ x\quad s\quad \textbf{destruct}\ x]\!] && = \textbf{construct}\ c\ x\quad \mathcal{I}[\![s]\!]\quad \textbf{destruct}\ x \\
&\mathcal{I}[\![\textbf{local}\ t\ x\ = e\quad s\quad \textbf{delocal}\ t\ x\ = e]\!] && = \textbf{local}\ t\ x = e\quad \mathcal{I}[\![s]\!]\quad \textbf{delocal}\ t\ x\ = e
\end{aligned}
$$

Figure 2.27: ROOPL++ statement inverter, extended from [11]

Program inversion is conducted by recursive descent over components and statements. A proposed extension to the statement inverter for whole-program inversion is retained in the ROOPL++ statement inverter. The extension covers a case that reveals itself during method calling. As a method call is equivalent to an uncall with the inverse method we simply change calls to uncalls during inversion, the inversion of the method body cancels out. The proposed extension, presented in [31, 11], simply avoids inversion of calls and uncalls, as shown in figure 2.28.

2.10.1 Invertibility of Statements

While the invertibility of statements remains untouched by the extensions made in ROOPL++, the following proof, originally presented in [11], has been included for completeness.

$$\mathcal{I}'[\![\textbf{call } q(\ldots)]\!] = \textbf{call } q(\ldots) \qquad \mathcal{I}'[\![\textbf{call } x :: q(\ldots)]\!] = \textbf{call } x :: q(\ldots)$$
$$\mathcal{I}'[\![\textbf{uncall } q(\ldots)]\!] = \textbf{uncall } q(\ldots) \qquad \mathcal{I}'[\![\textbf{uncall } x :: q(\ldots)]\!] = \textbf{uncall } x :: q(\ldots)$$
$$\mathcal{I}'[\![s]\!] = \mathcal{I}[\![s]\!]$$

Figure 2.28: Modified statement inverter for statements, originally from [11]

If execution of a statement s in store μ yields μ', then execution of the inverse statement, $\mathcal{I}[\![s]\!]$ in store μ' should yield μ. Theorem 2.1 shows that $\mathcal{I}$ is a statement inverter.

Theorem 2.1. *(Invertibility of statements, originally from [11])*

$$\overbrace{\langle l, \gamma\rangle \ \vdash^{\Gamma}_{stmt} \ s \ : \ \mu \rightleftharpoons \mu'}^{\mathcal{S}} \iff \overbrace{\langle l, \gamma\rangle \ \vdash^{\Gamma}_{stmt} \ \mathcal{I}[\![s]\!] \ : \ \mu' \rightleftharpoons \mu}^{\mathcal{S}'}$$

Proof. By structural induction on the semantic derivation of $\mathcal{S}$ (omitted). It suffices to show that $\mathcal{S} \implies \mathcal{S}'$, as this can serve as proof of $\mathcal{S}' \implies \mathcal{S}$, as $\mathcal{I}$ is an involution.

2.10.2 Type-Safe Statement Inversion

Given a well-typed statement, the statement inverter $\mathcal{I}$ should always produce a well-typed, inverse statement in order to correctly support backwards determinism of injective functions. Theorem 2.2 describes this.

Theorem 2.2. *(Inversion of well-typed statements, originally from [11])*

$$\overbrace{\langle \Pi, \ c\rangle \ \vdash^{\Gamma}_{stmt} \ s}^{\mathcal{T}} \implies \overbrace{\langle \Pi, \ c\rangle \ \vdash^{\Gamma}_{stmt} \ \mathcal{I}[\![s]\!]}^{\mathcal{T}'}$$

Proof. By structural induction on $\mathcal{T}$. Unmodified ROOPL statements retained in ROOPL++ has been omitted.

- Case $\mathcal{T} =$

$$\frac{\overbrace{\Pi \vdash_{expr} e_1 \ : \ \textbf{int}}^{\mathcal{E}_1} \qquad \overbrace{\Big(x \ \cup \text{vars}(e_1)\Big) \cap \text{vars}(e_2) = \emptyset}^{\mathcal{C}_2} \qquad \overbrace{\Pi \vdash_{expr} \ e_2 : \textbf{int}}^{\mathcal{E}_2}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \ x[e_1] \odot\!= e_2} \ \overbrace{\Pi(x) = \textbf{int}[\,]}^{\mathcal{C}_1} \ \text{T-ARRELEMASS}$$

  In this case, we have $\mathcal{I}[\![x \odot\!= e]\!] = x \odot'\!= e$, for some $\odot'$. Therefore, $\mathcal{T}'$ will also be a derivation of rule T-ARRELEMASS, and as such, we can simply reuse the conditions $\mathcal{C}_1, \mathcal{C}_2$

and the expressions $\mathcal{E}_1.\mathcal{E}_2$ in construction of $\mathcal{T}'$

$$\mathcal{T}' = \frac{\overbrace{\Pi \vdash_{expr} e_1 \ : \ \mathbf{int}}^{\mathcal{E}_1} \qquad \overbrace{\Big(x \ \cup \mathrm{vars}(e_1)\Big) \cap \mathrm{vars}(e_2) = \emptyset}^{\mathcal{C}_2} \qquad \overbrace{\Pi \vdash_{expr} \ e_2 : \mathbf{int}}^{\mathcal{E}_2}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \ x[e_1] \odot' = e_2} \quad \overbrace{\Pi(x) = \mathbf{int}[\,]}^{\mathcal{C}_1}$$

- Case $\mathcal{T} = \dfrac{\overbrace{\Pi(x) = c'}^{\mathcal{C}_1}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \ \mathbf{new} \ c' \ x}$ T-ObjNew

  In this case we have $\mathcal{I}[\![\mathbf{new} \ c \ x]\!] \ = \mathbf{delete} \ c \ x$, meaning $\mathcal{T}'$ must be of the form:

$$\mathcal{T}' = \frac{\overbrace{\Pi(x) = c'}^{\mathcal{C}_2}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \ \mathbf{delete} \ c' \ x}$$

- Case $\mathcal{T} = \dfrac{\overbrace{\Pi(x) = c'}^{\mathcal{C}_1}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \ \mathbf{delete} \ c' \ x}$ T-ObjDlt

  Inverse of the previous case, we now have $\mathcal{I}[\![\mathbf{delete} \ c \ x]\!] \ = \mathbf{new} \ c \ x$, meaning $\mathcal{T}'$ must be of the form:

$$\mathcal{T}' = \frac{\overbrace{\Pi(x) = c'}^{\mathcal{C}_2}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \ \mathbf{new} \ c' \ x}$$

- Case $\mathcal{T} = \dfrac{\overbrace{\mathrm{arrayType}(a) \in \Big\{\mathrm{classIDs}, \mathbf{int}\Big\}}^{\mathcal{C}_1} \qquad \overbrace{\Pi \vdash_{expr} \ e = \mathbf{int}}^{\mathcal{E}} \qquad \overbrace{\Pi(x) = a[\,]}^{\mathcal{C}_2}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \ \mathbf{new} \ a[e] \ x}$ T-ArrNew

  In this case we still have $\mathcal{I}[\![\mathbf{new} \ c \ x]\!] \ = \mathbf{delete} \ c \ x$. Using $\mathcal{C}_1$ and $\mathcal{E}$, $\mathcal{T}'$ must be of the form:

$$\mathcal{T}' = \frac{\overbrace{\mathrm{arrayType}(a) \in \Big\{\mathrm{classIDs}, \mathbf{int}\Big\}}^{\mathcal{C}_1} \qquad \overbrace{\Pi \vdash_{expr} \ e = \mathbf{int}}^{\mathcal{E}} \qquad \overbrace{\Pi(x) = a[\,]}^{\mathcal{C}_3}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \ \mathbf{delete} \ a[e] \ x}$$

- Case $\mathcal{T} = \dfrac{\overbrace{\mathrm{arrayType}(a) \in \Big\{\mathrm{classIDs}, \mathbf{int}\Big\}}^{\mathcal{C}_1} \qquad \overbrace{\Pi \vdash_{expr} \ e = \mathbf{int}}^{\mathcal{E}} \qquad \overbrace{\Pi(x) = a[\,]}^{\mathcal{C}_2}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \ \mathbf{delete} \ a[e] \ x}$ T-ArrDlt

Similar to the object deletion case, we still have $\mathcal{I}[\![\textbf{delete } c\ x]\!] = \textbf{new } c\ x$. Using $\mathcal{C}_1$ and $\mathcal{E}$, $\mathcal{T}'$ must be of the form:

$$\mathcal{T}' = \frac{\overbrace{\text{arrayType}(a) \in \left\{\text{classIDs}, \textbf{int}\right\}}^{\mathcal{C}_1} \qquad \overbrace{\Pi \vdash_{expr} e = \textbf{int}}^{\mathcal{E}} \qquad \overbrace{\Pi(x) = a[\,]}^{\mathcal{C}_3}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \textbf{new } a[e]\ x}$$

- Case $\mathcal{T} = \dfrac{\overbrace{\Pi(x) = c'}^{\mathcal{C}_1} \qquad \overbrace{\Pi(x') = c'}^{\mathcal{C}_2}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \textbf{copy } c'\ x\ x'}$ T-Cp

We have $\mathcal{I}[\![\textbf{copy } c\ x\ x']\!] = \textbf{uncopy } c\ x\ x'$. Using $\mathcal{C}_1$, $\mathcal{T}'$ must as such be of the form

$$\mathcal{T}' = \frac{\overbrace{\Pi(x) = c'}^{\mathcal{C}_1} \qquad \overbrace{\Pi(x') = c'}^{\mathcal{C}_3}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \textbf{uncopy } c'\ x\ x'}$$

- Case $\mathcal{T} = \dfrac{\overbrace{\Pi(x) = c'}^{\mathcal{C}_1} \qquad \overbrace{\Pi(x') = c'}^{\mathcal{C}_2}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \textbf{uncopy } c'\ x\ x'}$ T-Ucp

We have $\mathcal{I}[\![\textbf{uncopy } c\ x\ x']\!] = \textbf{copy } c\ x\ x'$. Using $\mathcal{C}_1$, $\mathcal{T}'$ must as such be of the form

$$\mathcal{T}' = \frac{\overbrace{\Pi(x) = c'}^{\mathcal{C}_1} \qquad \overbrace{\Pi(x') = c'}^{\mathcal{C}_3}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \textbf{copy } c'\ x\ x'}$$

- Case $\mathcal{T} = \dfrac{\overbrace{\langle \Pi, c\rangle \vdash^{\Gamma}_{expr} e_1}^{\mathcal{E}_1} \qquad \overbrace{\langle \Pi[x \mapsto c'], c\rangle \vdash^{\Gamma}_{stmt} s}^{\mathcal{S}} \qquad \overbrace{\langle \Pi, c\rangle \vdash^{\Gamma}_{expr} e_2}^{\mathcal{E}_2}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \textbf{local } c'\ x = e_1 \quad s \quad \textbf{delocal } c'\ x = e_2}$ T-LocalBlock

We have $\mathcal{I}[\![\textbf{local } t\ x = e \quad s \quad \textbf{delocal } t\ x = e]\!] = \textbf{local } t\ x = e \quad \mathcal{I}[\![s]\!] \quad \textbf{delocal } t\ x = e$. By the induction hypothesis on $\mathcal{S}$, we obtain $\mathcal{S}'$ of $\langle \Pi[x \mapsto c'], c\rangle \vdash^{\Gamma}_{stmt} \mathcal{I}[\![s]\!]$. Using $\mathcal{E}_1$, $\mathcal{S}'$ and $\mathcal{E}_2$ we construct $\mathcal{T}'$

$$\mathcal{T}' = \frac{\overbrace{\langle \Pi, c\rangle \vdash^{\Gamma}_{expr} e_1}^{\mathcal{E}_1} \qquad \overbrace{\langle \Pi[x \mapsto c'], c\rangle \vdash^{\Gamma}_{stmt} \mathcal{I}[\![s]\!]}^{\mathcal{S}'} \qquad \overbrace{\langle \Pi, c\rangle \vdash^{\Gamma}_{expr} e_2}^{\mathcal{E}_2}}{\langle \Pi, c\rangle \vdash^{\Gamma}_{stmt} \textbf{local } c'\ x = e_1 \quad \mathcal{I}[\![s]\!] \quad \textbf{delocal } c'\ x = e_2}$$

Using these added cases to the original proof provided in [11], Theorem 2.2 shows that well-typedness is preserved over inversion of Roopl++ methods. As methods are well-typed if their body statement is well-typed, inversion of classes and programs also preserve well-typedness, as classes consists of methods and programs of classes, by using the class inverter presented in figure 2.28.

2.11 Computational Strength

Traditional, non-reversible programming languages have their computational strength measured in terms of their abilities to simulate the Turing machine (TM). If any arbitrary Turing machine can be implemented in some programming language, the language is said to be computationally universal or Turing-complete. In essence, Turing-completeness marks when a language can compute all computable functions. Reversible programming languages, like JANUS, ROOPL and ROOPL++, are not Turing-complete as they only are capable of computing injective, computable functions.

For determining computing strength of reversible programming languages, Yokoyama et al. suggests that the reversible Turing machine (RTM) could serve as the baseline criterion [29]. As such, a reversible programming language is reversibly universal or r-Turing complete if it is able to simulate a reversible Turing machine cleanly, i.e. without generating garbage data. If garbage was on the tape, the function simulated by the machine would not be an injective function and as such, no garbage should be left after termination of the simulation.

2.11.1 Reversible Turing Machines

Before we show that ROOPL++ in fact is r-Turing complete, we present the formalized reversible Turing machine definition, as defined in [29].

Definition 2.1. *(Quadruple Turing Machine)*

A TM T *is a tuple* $(Q,\ \Gamma,\ b,\ \delta,\ q_s,\ q_f)$ *where*

Q *is the finite non-empty set of states*

Γ *is the finite non-empty set of tape alphabet symbols*

$b\ \in\ \Gamma$ *is the blank symbol*

$\delta\ :\ (Q\ \times\ \Gamma\ \times\ \Gamma\ \times\ Q)\ \cup\ (Q\ \times\ \{/\}\ \times\ \{L,\ R\}\ \times\ Q)$ *is the partial function representing the transitions*

$q_s\ \in\ Q$ *is the starting state*

$q_f\ \in\ Q$ *is the final state*

The symbols L *and* R *represent the tape head shift-directions left and right. A quadruple is either a symbol rule of the form* $(q_1,\ s_1,\ s_2,\ q_2)$ *or a shift rule of the form* $(q_1,\ /,\ d,\ q_2)$ *where* $q_1 \in Q$, $q_2 \in Q$, $s_1 \in \Gamma$, $s_2 \in \Gamma$ *and* d *being either* L *or* R.

A symbol rule $(q_1,\ s_1,\ s_2,\ q_2)$ *means that in state* q_1*, when reading* s_1 *from the tape, write* s_2 *to the tape and change to state* q_2*. A shift rule* $(q_1,\ /,\ d,\ q_2)$ *means that in state* q_1*, move the tape head in direction* d *and change to state* q_2.

Definition 2.2. *(Reversible Turing Machine)*

A TM T is a reversible TM iff, for any distinct pair of quadruples $(q_1,\ s_1,\ s_2,\ q_2)\ \in\ \delta_T$ and $(q_1',\ s_1',\ s_2',\ q_2')\ \in\ \delta_T$, we have

$$q_1 = q_1' \implies (t_1 \neq / \ \wedge \ t_1' \neq / \ \wedge \ t_1 \neq t_1') \textit{ (forward determinism)}$$

$$q_2 = q_2' \implies (t_1 \neq / \ \wedge \ t_1' \neq / \ \wedge \ t_2 \neq t_2') \textit{ (backward determinism)}$$

A RTM simulation implemented in ROOPL by representing the set of states $\{q_1,\ \ldots,\ q_n\}$ and the tape alphabet Γ as integers and the rule / and direction symbols L and R as the uppercase integer literals **`SLASH`**, **`LEFT`** and **`RIGHT`** was presented in [11]. As ROOPL contains no array or stack primitives, the transition table δ was suggested to be represented as a linked list of objects containing four integers **q1**, **s1**, **s2** and **q2** each, where **s1** equals **`SLASH`** for shift rules. In ROOPL++, we do, however, have an array primitive and as such, we can simply simulate transitions by having rules **q1**, **s1**, **s2** and **q2** represented as arrays, where the number of cells in each array is **`PC_MAX`**, in a similar fashion as shown in [29].

2.11.2 Tape Representation

As with regular Turing machines, the Reversible Turing machines also have tapes of infinite length. Therefore, we must simulate tape growth in either direction. Yokoyama et al. represented the tape using two stack primitives in the Janus RTM interpreter and Haulund used list of objects. In ROOPL++, we could implement a stack, as objects are not statically scoped as in ROOPL. However, in terms of easy of use, a doubly linked list implementation similar to the one presented in section 2.7.3, of simple cell objects containing *value*, *left*, *right* and *self* fields, is more intuitive.

As such, the tape head finds a tape cell by inspecting a specific element of the doubly linked list tape representation. When we move in either direction, we simply set the neighbour element as the new tape head and allocate a new neighbour for the new tape head cell, if we are at the end of the list, to simulate the infinitely-length tape. Reversibly, this means that when we move in the opposite direction, blank cells are deallocated if we are moving the tape head away from the cell currently neighbouring either end of the tape.

Figure 2.29 shows the *moveRight* method for moving the tape head right. If the current tape head has no instantiated right neighbour we construct one using the **new** statement. Uncalling this method will move the tape head left. If the tape head is empty after moving left, we simply allocate a new cell, thus allowing tape growth in both directions.

```
1  method moveRight(int symbol, Cell tapeHead)
2      local Cell right = nil
3      local Cell tmp = nil
4      uncall tapeHead::getSymbol(symbol)   // Put symbol back in current cell
5      call tapeHead::getRight(right)       // Get right neighbour
6
7      if right = nil && symbol = BLANK then
8          symbol ^= BLANK                  // Zero clear symbol
9          new Cell right                   // Init new neighbour
10         copy Cell right tmp              // Copy reference to self
11         uncall right::getSelf(tmp)       // Store self reference
12         uncall right::getLeft(tapeHead)  // Set tape head as left of new cell
13         right <=> tapeHead
14     else
15         call right::getLeft(tmp)         // Get copy of tape head reference
16         uncopy Cell tmp tapeHead         // Clear reference to tape head
17
18         if tapeHead = nil && symbol = BLANK
19             call tmp::getSelf(tapeHead) // rev: set self pointer
20             uncopy Cell tmp tapeHead    // rev: new self pointer
21             delete Cell tmp             // rev: new left neighbour
22             symbol ^= BLANK
23         else skip                       // In reverse:
24         fi tmp = nil                    // Allocate new left if current is nil
25
26         uncall right::getLeft(tmp)       // Put tape head reference back
27         tapeHead <=> right
28         call tapeHead::getRight(right)   // Get right of new tape head
29         call tapeHead::getSymbol(symbol) // Get symbol of new tape head
30     fi right = nil
31     uncall tapeHead::getRight(right)    // Set right neighbour
32     delocal Cell right = nil
33     delocal Cell tmp = nil
```

Figure 2.29: Method for moving the tape head in the RTM simulation

2.11.3 Reversible Turing Machine Simulation

Figure 2.30 shows the modified method *inst* from [29], which executes a single instruction given the tape head, the current state, symbol, program counter and the four arrays representing the transition rules. As described above, we **call** *moveRight* to move the tape head right and **uncall** to move the tape head left.

Figure 2.31 shows the simulate method which is the main method responsible for running the RTM simulation. The tape is extended in either direction when needed, and the program counter is incremented.

Unlike the ROOPL simulation, ROOPL++ is not limited by stack allocated, statically-scoped objects. Due to this limitation, the ROOPL RTM simulator cannot finish with the TM tape as its program output when the RTM halts, as the call stack of the simulation must unwind before termination. As objects in ROOPL++ are not bound by this limitation, the TM tape will exist as the program output when the RTM halts.[1]

Instantiating a RTM simulation consists of initializing an initial tape head cell, as well as the

[1] We are here breaking the rule that a **new** statement must eventually be followed by a **delete** statement to free the data.

```
method inst(int state, int symbol, int[] q1, int[] s1,
            int[] s2, int[] q2, int pc, Cell tapeHead)
    if state = q1[pc] && symbol = s1[pc] then   // Symbol rule:
        state += q2[pc]-q1[pc]                  // set state to q2[pc]
        symbol += s2[pc]-s1[pc]                 // set symbol to s2[pc]
    fi state = q2[pc] && symbol = s2[pc]
    if state = q1[pc] && s1[pc] = SLASH then    // Move rule:
        state += q2[pc]-q1[pc]                  // set state to q2[pc]
        if s2[pc] = RIGHT then
            call moveRight(symbol, tapeHead)    // Move tape head right
        fi s2[pc] = RIGHT
        if s2[pc] = LEFT then
            uncall moveRight(symbol, tapeHead)  // Move tape head left
        fi s2[pc] = LEFT
    fi state = q2[pc] && s1[pc] = SLASH
```

Figure 2.30: Method for executing a single TM transition

```
method simulate(Cell tapeHead, int state, int[] q1, int[] s1, int[] s2, int[] q2, int pc)
    from state = Qs do
        pc += 1                                 // Increment pc local int symbol = 0
        call tapeHead::getSymbol(symbol)        // Fetch current symbol
        call inst(state, symbol, q1, s1, s2, q2, pc, tapeHead)
        uncall tapeHead::getSymbol(symbol)      // Zero-clear symbol delocal symbol = 0
        if pc = PC_MAX then                     // Reset pc
            pc ^= PC_MAX
        else skip
        fi pc = 0
    loop skip
    until state = Qf
```

Figure 2.31: Main RTM simulation method

transition rule arrays. After initialization, the *simulate* method is simply called and the simulation begins.

CHAPTER 3

Dynamic Memory Management

In order to allow objects to live outside of static scopes, we need to utilize a different memory management technique, such that objects are not allocated on the stack. Dynamic memory management presents a method of storing objects in different memory structures, most commonly, a memory heap. Most irreversible, modern programming languages uses dynamic memory management in some form for allocating space for objects in memory.

However, reversible, native support for complex data structures is a non-trivial matter to implement. Variable-sized records and frames need to be stored efficiently in a structured heap, while avoiding garbage build-up to maintain reversibility. A reversible heap manager layout has been proposed for a simplified version of the reversible functional language RFun and later expanded to allow references to avoid deep copying values [2, 30, 18].

This chapter presents a brief introduction to fragmentation, garbage and linearity and how these respectively are handled reversibly, and a discussion of various heap manager layouts considered for Roopl++, along with their advantages and disadvantages in terms of implementation difficulty, garbage build-up and the OOP paradigm.

3.1 Fragmentation

Efficient memory usage is an important matter to consider when designing a heap layout for a dynamic memory manager. In a stack allocating memory layout, the stack discipline is in effect, meaning only the most recently allocated data can be freed. This is not the case with heap allocation, where data can be freed regardless of allocation order. A potential side effect of this freedom, comes as a consequence of memory fragmentation. We distinguish different types of fragmentation as internal or external fragmentation.

Internal fragmentation refers to unused space inside a memory block used to store an object, if, say, the object is smaller than the block it has been allocated to. External fragmentation occurs as blocks freed throughout execution are spread across the memory heap, resulting in *fragmented* free space [20].

3.1.1 Internal Fragmentation

Internal fragmentation occurs in the memory heap when part of an allocated memory block is unused. This type of fragmentation can arise from a number of different scenarios, but mostly it originates from cases of *over-allocation*, which occurs when the memory manager delegates memory larger than required to fit an object, due to e.g. fixed-block sizing.

For an example, consider a scenario, in which we allocate memory for an object of size m onto a simple, fixed-sized block heap. The fixed block size is n and $m \neq n$. If $n > m$, internal fragmentation would occur of size $n - m$ for every object of size m allocated in said heap. If $n < m$, numerous blocks would be required for allocation to fit our object. In this case the internal fragmentation would be of size $n - m \ mod \ n$ per allocated object of size m.

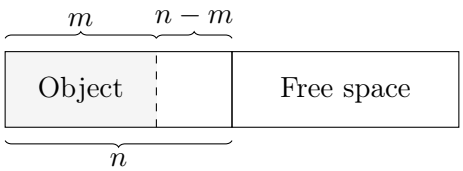


Figure 3.1a: Creation of internal fragmentation of size $n - m$ due to *over-allocation*

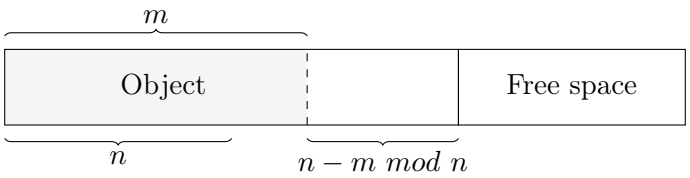


Figure 3.1b: Creation of internal fragmentation of size $n - m \ mod \ n$ due to *over-allocation*

Figure 3.1a and 3.1b visualize the examples of internal fragmentation build-up from *over-allocating* memory.

It is difficult for the memory manager to reclaim wasted memory caused by internal fragmentation, as it usually originates from a design choice. Intuitively, internal fragmentation can best be prevented by ensuring that the size of block(s) being used for allocating space for an object of size m either match or sums to this exact size, when designing the layout.

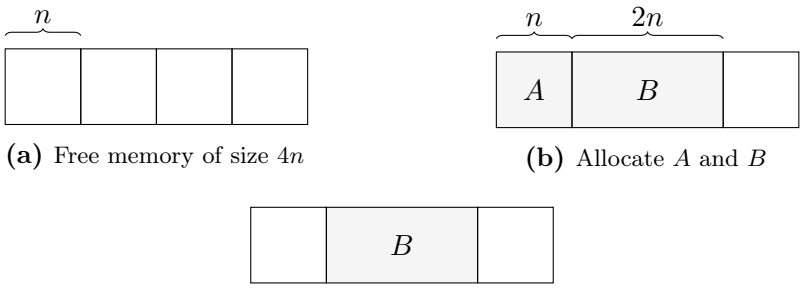
3.1.2 External Fragmentation

External fragmentation materializes in the memory heap when a freed block becomes partly or completely unusable for future allocation if, say, it is surrounded by allocated blocks but the size of the freed block is too small to contain objects on its own.

This type of fragmentation is generally a more substantial cause of problems than internal fragmentation, as the amount of wasted memory typically is larger and less predictable in external fragmentation blocks than in internal fragmentation blocks. Depending on the heap implementation, i.e. a layout using variable-sized blocks of, say, size 2^n, the internal fragment size becomes considerable for large values of n.

Non-allocatable external fragments become a problem when it is impossible to allocate space for a large object as a result of too many non-consecutive blocks scattered around the heap, caused by the external fragmentation. Physically, there is enough space to store the object, but not in the current heap state. In this scenario we would need to relocate blocks in such a manner that the fragmentation disperses, which is not possible to do reversibly.

Allocation and deallocation order is important in order to combat external fragmentation. For example, if we have a class A, which fit on one memory block of size n, and we have a class B, which fit on two memory blocks of size n and limited memory space, we can easily reach a situation, where we cannot fit more B objects due to external fragmentation.



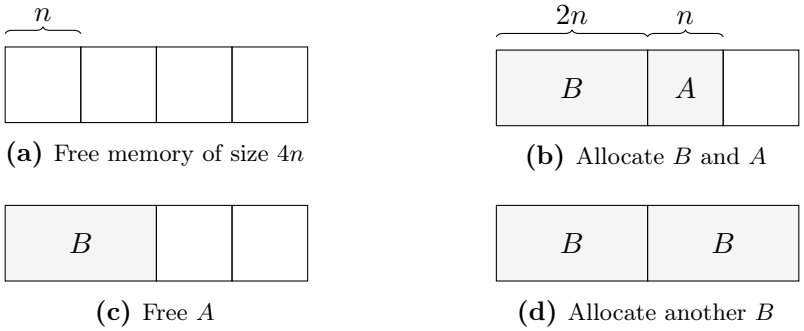
(a) Free memory of size $4n$

(b) Allocate A and B

(c) Free A. Cannot fit another B due to external fragmentation

Figure 3.2: Example of external fragmentation caused for allocation and deallocation order

Figure 3.2 shows this example, where the allocation and deallocation order causes a situation, in which we cannot allocate any more B objects, even though we physically have the required amount of free space in memory.



(a) Free memory of size $4n$

(b) Allocate B and A

(c) Free A

(d) Allocate another B

Figure 3.3: Example of avoiding external fragmentation using allocation and deallocation order

Figure 3.3 shows how changing allocation and deallocation order can combat external fragmentation.

3.2 Memory Garbage

A reversible computation should be garbage-free and as such it should be our goal to return the memory to its original state after program termination.

Traditionally, in non-reversible programming languages, freed memory blocks are simply re-added to the free list during deallocation and no modification of the actual data stored in the block is performed, as it simply is overwritten when the block is used later on. In the reversible setting we must return the memory block to its original state after the block has been freed (e.g. zero-cleared), to uphold the time-invertible and two-directional computational model. Figure 3.4 illustrates how the output data (or garbage) of an injective function f is the input to its inverse function f^{-1}.

In heap allocation layouts, we maintain one or more free lists to keep track of free blocks during program execution, which are stored in memory, besides the heap representation itself. These free lists can essentially be considered garbage and as such, they must also be returned to their original state after execution. Furthermore, the heap itself can also be considered garbage and if it grows during execution, it should also be returned to its original size.

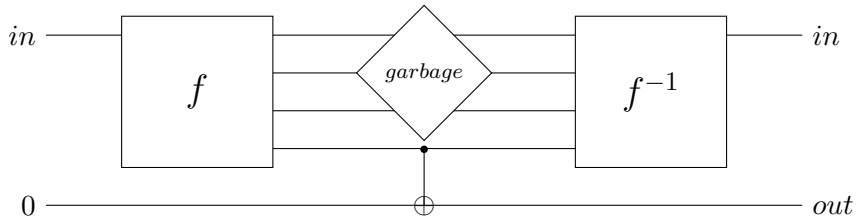


Figure 3.4: The "garbage" output of an injective function f is the input to its inverse function f^{-1}

Returning the free list(s) to their original states is a non-trivial matter, which is highly dependent on the heap layout and free list design. Axelsen and Glück introduced a dynamic memory manager which allowed heap allocation and deallocation, but without restoring the free list to its original state in [2]. Axelsen and Glück argue that an unrestored free list can be considered harmless garbage in the sense that the free list residing in memory after termination is equivalent to a restored free list, as it contains the same blocks, but linked in a different order, depending on the order of allocation and deallocation operations performed during program execution. Figure 3.5 illustrates how an inverse, injective function f^{-1}, whose non-inverse function f computes something which modifies a given free lists, does not require the *exact* output free list of f, but *any* free list of same layout as input for the inverse function f^{-1}. The output free list of f^{-1} will naturally be a further modified free list.

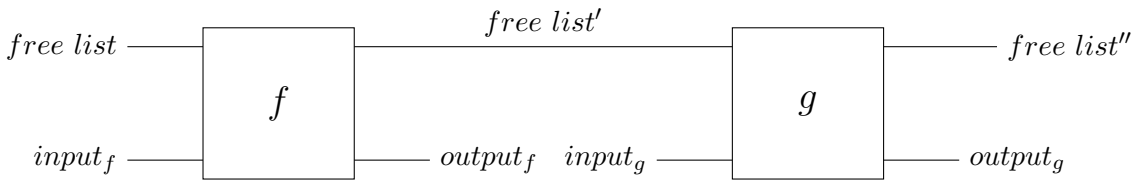


Figure 3.5: All free lists are considered equivalent "garbage" in terms of injective functions

This intuitively leads to the question of garbage classification. In the reversible setting all functions are injective. Thus, given some $input_f$, in a reversible computation using heap allocation, the injective function f produces some $output_f$ and some modified free list $free\ list'$, obtained after storing or freeing data in the heap during the execution of f with an input $free\ list$. A future injective function in the program, function g, must thus take any modification of the original

free list in addition to its input to produce its output $output_g$ and a potentially further modified free list, $free\ list''$. However, in the context of reversible heaps, we must consider all free lists as of equivalent, harmless garbage class and thus freely substitutable with each other, as injective functions still can drastically change the block layout, free list order, etc. during its execution in either direction. Figure 3.5 shows how any free list can be passed between a function f and a function g further in the program.

3.3 Linearity and Reference Counting

Programming languages use different approaches for storing and synchronizing variables and objects in memory. Typing *linearity* is a distinction, which can reduce storage management and synchronization costs [4].

Reversible programming languages such as Janus and Roopl are linear in the sense that object and variable pointers cannot be copied and are only deleted during deallocation. Pointer copying greatly increases the flexibility of programming, especially in a reversible setting where zero-clearing is critical, at the cost of increased management in form of reference counting for e.g. objects. For variables, pointer copying is not particular interesting, nor would it add much flexibility as the values of a variable simply can be copied into statically-scoped local blocks. For objects however, tedious amounts of boilerplate work must be done if object A and B need to work on the same object C and only one reference to each object is allowed. Copying is not an option as field modification in one copy does not affect the other copies.

Mogensen presented the reversible functional language Rcfun which use reference counting to allow multiple pointers to the same memory nodes as well as a translation from Rcfun into Janus in [18]. In Rcfun, reference counting is used to manage and trace the number of pointer copies made by respectively incrementing and decrementing a *reference count* stored in the memory node, whenever the original node pointer is copied or a copy pointer is deleted. For the presented heap manage, deletion of object nodes was only allowed when no references to a node remained.

In non-reversible languages, reference counting is also used in garbage collection by automatically deallocating unreachable objects and variables which contains no referencing.

3.4 Heap Manager Layouts

Heap managers can be implemented in numerous ways. Different layouts yield different advantages when allocating memory, finding a free block or when collecting garbage. As our goal is to construct a garbage-free heap manager, our finalized design should emphasize and reflect this objective in particular. Furthermore, we should attempt to allocate and deallocate memory as efficiently as possible, as merging and splitting of blocks is a non-trivial problem in a reversible setting and to avoid problematic fragmentation.

For the sake of simplicity, we will not consider the issue of retrieving memory pages reversibly. A reversible operating system is a long-term dream of the reversible researcher and as reversible programming language designers, we assume that Roopl++ will be running in an environment,

in which an operating system will be supplying memory pages and their mappings. As such, the following heap memory designs reflect this preliminary assumption, that we can always query the operating system for more memory.

Historically, most object-oriented programming languages utilize a dynamic memory manager during program execution. In lower-level languages such as C or C++, memory management is manual and allocation has to be stated explicitly and with the requested size through the **malloc** statement and deallocated using the **free** statement. Modern languages, such as JAVA and PYTHON, *automagically* allocates and frees space for objects and variable-sized arrays by utilizing their dynamic memory manager and garbage collector to dispatch **malloc**- and **free**-like operations to the operating system and managing the obtained memory blocks in private heap(s) [15, 25, 21]. The heap layout of these managers vary from language to language and compiler to compiler.

Previous work on reversible heap manipulation has been done for reversible functional languages in [2, 10, 19].

Axelsen and Glück presented a static heap structure consisting of LISP-inspired constructor cells of fixed size and a single free list for the reversible function language RFUN in [2]. Mogensen presented an implementation in JANUS of reversible reference counting under the assumption of Axelsen and Glück's heap manager in [18]. Building on the previous work, Mogensen later presented a reversible intermediate language RIL and an implementation in RIL of a reversible heap manager, which uses reference counting and hash-consing to achieve garbage collection in [19].

We do not consider reference counting or garbage collection in the layouts presented in the following sections, but we later show how the selected layout for ROOPL++ is extended with reference counting in section 4.7.

3.4.1 Memory Pools

The simplest heap layout we can design uses fixed-sized blocks. This design is also known as memory pools, as memory is allocated from "pools" of fixed-sized blocks regardless of the record size. To model these pools of fixed-sized blocks, we simply use a linked list of identically sized free block cells, which we maintain over execution. While the fixed-block layout is simple and relatively easy in terms of implementation it is also largely uninteresting as it provides little to no options, besides sizing of the fixed-blocks, to combat fragmentation.

This layout comes with a few options in terms of the actual heap layout. If we only allow allocation of consecutive, adjacent free blocks, we should keep the free list sorted. If the free list is not sorted, and we have to allocate an object which requires n blocks, we have to iterate the free list n^2 times in the worst case to find a chain of consecutive blocks large enough to fit the object. The sorting part itself is non-trivial matter. Furthermore, we need some overhead storage inside the object to contains the references of the blocks occupied by the object, or some other structure which can be used when deallocating the object and returning all the blocks to the free list. If we allow allocation of non-consecutive blocks, larger amounts of bookkeeping is required as we need to store knowledge of when and where the object is split.

Figures 3.2 and 3.3 from earlier in this chapter, in section 3.1.2 on page 47 illustrates examples with consecutive, fixed-sized block allocation.

3.4.2 One Heap Per Record Size

Instead of allocating space for objects from a single free list and heap, we could design an approach which uses one heap per record size, known as a multi-heap layout. The respective classes and their sizes are easily identified during compile time from which the amount of heaps and free list will be initialized. This means the layout is very dynamic and potentially can change drastically in terms of the amount of heaps utilized depending on the input program.

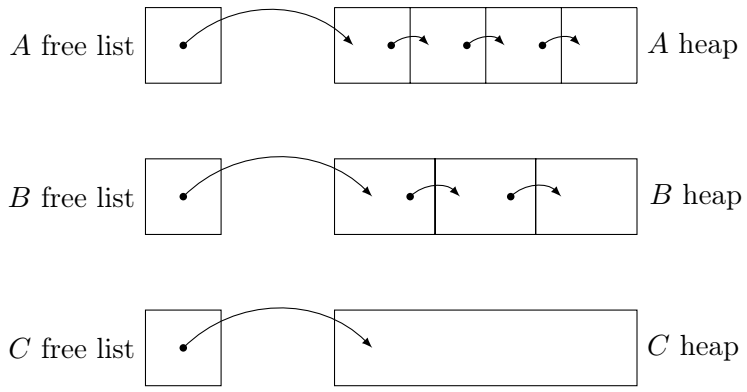


Figure 3.6: Memory layout using one heap per record size

Figure 3.6 illustrates three heaps with respective free lists for three classes A, B and C of size n, $2n$ and $4n$. Each heap is represented as a simple linked list with the free list simply being a pointer to the first free block in the heap.

The advantage of this approach would be effective elimination of internal and external fragmentation, as each heap fits their targeted record perfectly, making each allocation and deallocation tailored to the size of the record obtained from a static analysis during compilation, resulting in no over-allocation and no unusable chunks of freed memory appearing during varying deallocation order. Implementation-wise, allocation of an object of a given class simply becomes the task of popping the head of the respective free list, which can easily be determined at compile time. The deallocation is simply adding a new head to the free list.

Listing 3.1 outlines the allocation algorithm for this layout written in extended Janus from [29]. We assume that the heads of the free lists are stored in a single array primitive, such that the free list for records of size n are indexed at $n-2$ and $n > 2$ (as every record needs some overhead) and that we have heaps for continuous size range with no gaps.

The algorithm consists of an entry point named **malloc** and a recursion body named **malloc1**. Given a zero-cleared pointer p, the size of the object we are allocating o_{size} and the array of free lists primitive, the recursion body is called after initializing a *counter*, which is an index into the free lists array and a counter size, c_{size}, which is the block size of the current free list the *counter* is indexed in. The recursion body first updates the free list index until we find a free list with a size greater or equal to the size of the object we are allocating. Once such a free list

has been found, the head of the free list is simply popped and the next block is set as the new head.

```
1  procedure malloc(int p, int osize, int freelists[])
2    local int counter = 0
3    local int csize = 2
4    call malloc1(p, osize, freelists, counter, csize)
5    delocal int csize = 2
6    delocal int counter = 0
7
8  procedure malloc1(int p, int osize, int freelists[], int counter, int csize)
9    if (csize < osize) then
10     counter += 1
11     csize += 1
12     call malloc1(p, osize, freelists, counter, csize)
13     csize -= 1
14     counter -= 1
15   else
16     p += freelists[counter]
17     freelists[counter] -= p
18
19     // Swap head of free list with next block of p
20     freelists[counter] ^= M(p)
21     M(p) ^= freelists[counter]
22     freelists[counter] ^= M(p)
23   fi csize < osize
```

Listing 3.1: Allocation algorithm for one heap per record size implemented in extended Janus

The obvious disadvantage to this layout is the amount of bookkeeping and workload associated with growing and shrinking a heap and its neighbours, in case the program requests additional memory from the operating system. In real world object-oriented programming, most classes feature a small number of fields, very rarely more than 16.

Additionally, helper classes of other sizes would spawn additional heaps and bookkeeping work, making the encapsulation concept of OOP rather unattractive, for the optimization-oriented reversible programmer.

Finally, while internal and external fragmentation is effectively eliminated, we are left with additional and considerable amounts of garbage in forms of all the heaps and free lists initialized in memory. If two record types only differ one word in size, two heaps would be initialized. Each heap intuitively need to be initialized with a chunk of memory from the underlying operating system such that objects can be allocated on their respective heaps, regardless of the number of times the heap is used during program execution. This is an obvious space requirement increase over the previously presented layout, and on average, the amount of required memory for a program compiled using this approach would probably be larger, than some of the following layouts, due to unoptimized heap utilization and sharing. Heap of some sizes may be mostly empty when another is full, resulting in wasted memory.

3.4.3 One Heap Per Power-Of-Two

To address the issues of the previous heap manager layout, we can optimize the amounts of heaps required by introducing a relatively small amount of internal fragmentation. Instead of having a heap per record size, we could have a heap per power-of-two. Records would be stored in the heap closest to their respective size and, as such, we reduce the number of heaps needed, as

many different records can be stored in the same heap. Records of size 5, 6, 7 and 8 would in the former layout be stored in four different heaps, where they would be stored in a single heap using this layout. Figure 3.7 illustrates the free lists and heaps up to 2^n.

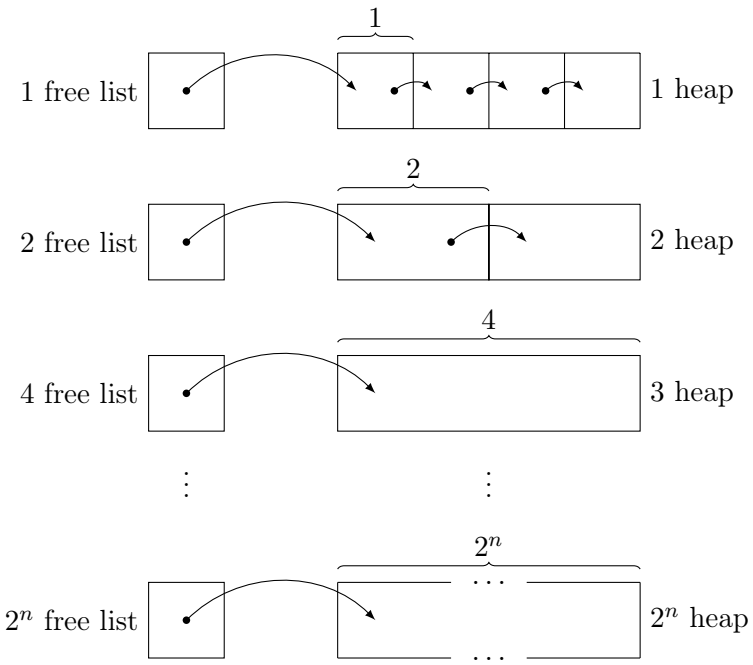


Figure 3.7: Memory layout using one heap per power-of-two

Internal fragmentation does become a problem for very large records, as blocks are only of size 2^n. An object of size 65 would fit in a 128 sized block, resulting in considerable amounts of wasted memory space in form of internal fragmentation. However, in the real world, most records are small and allocation of records causing this much amount of fragmentation is an unlikely scenario. To avoid large amounts of internal fragmentation building up when allocating large records, we could allocate space for large objects using smaller blocks. If a record exceeds some limit, which has been determined the cutoff point, one kilobyte for an example, we could split it into $\sqrt{n}$ sized chunks and use blocks of that size instead. This would reduce the amount of internal fragmentation at the cost of increased bookkeeping. For smaller records, very minimal amounts of internal fragmentation occur.

The number of heaps needed for a computation can be determined at compile time by finding the smallest and largest record sizes and ensuring we have heaps to fit these effectively. The allocation process consists of determining the closest 2^n to the size of the record we are allocating and then simply popping the head of the respective free list.

Listing 3.2 shows a modified **malloc1** recursion body for the power-of-two approach. Once again, we assume our array of free lists contains the head of each free list, such that index n is the head of the free list of size 2^{n+1}. Instead of incrementing the counter size by one, as in the former layout algorithm, we double it, using the shown **double** procedure. Besides this change, the algorithm remains unchanged and still assumes each heap has been initialized along with the free lists.

```
1  procedure double(int target)
2    local int current = target
3    target += current
4    delocal int current = target / 2
5
6  procedure malloc1(int p, int osize, int freelists[], int counter, int csize)
7    if (csize < osize) then
8        counter += 1
9        call double(csize)
10       call malloc1(p, osize, freelists, counter, csize)
11       uncall double(csize)
12       counter -= 1
13   else
14     if freelists[counter] != 0 then
15       p += freelists[counter]
16       freelists[counter] -= p
17
18       // Swap head of free list with next block of p
19       freelists[counter] ^= M(p)
20       M(p) ^= freelists[counter]
21       freelists[counter] ^= M(p)
22     else
23       counter += 1
24       call double(csize)
25       call malloc1(p, osize, freelists, counter, csize)
26       uncall double(csize)
27       counter -= 1
28     fi freelists[counter] = 0 || p != freelists[counter]
29   fi csize < osize
```

Listing 3.2: Allocation algorithm for one heap per power-of-two implemented in extended Janus

3.4.4 Shared Heap, Record Size-Specific Free Lists

A natural proposal, considering the disadvantages of the previously presented designs, would be using a shared heap instead of record-specific heaps. This way, we ensure minimal fragmentation when allocating and freeing as the different free lists ensure that allocation of an object wastes as little memory as possible. By only keeping one heap, we eliminate the growth/shrinking issues of the multiple heap layout.

There is, however, still a considerable amount of bookkeeping involved in maintaining multiple free lists. Having mixed-size blocks in a single heap is also a task which might prove difficult to accomplish reversibly. How initialization and destruction of said heap should work is not clear. As with the multiple heap version of this layout, we are still left with the issues surrounding two records which only differs one word in size. In the former layout, two heaps were required to store records of these types. In this layout, we need to store two block sizes in our heap to allocate these records, with no internal fragmentation. We could allow these objects to be allocated on similarly-sized blocks, if we round the calculated class sizes up to, say, a power-of-two. We would essentially have a shared heap, power-of-two-specific free lists layout.

As the only change in this design are the heaps themselves, the allocation process remains unchanged from the one presented in listing 3.1 or listing 3.2 if we use the power-of-two approach. Figure 3.8 visualizes the shared heap and the free lists of this layout.

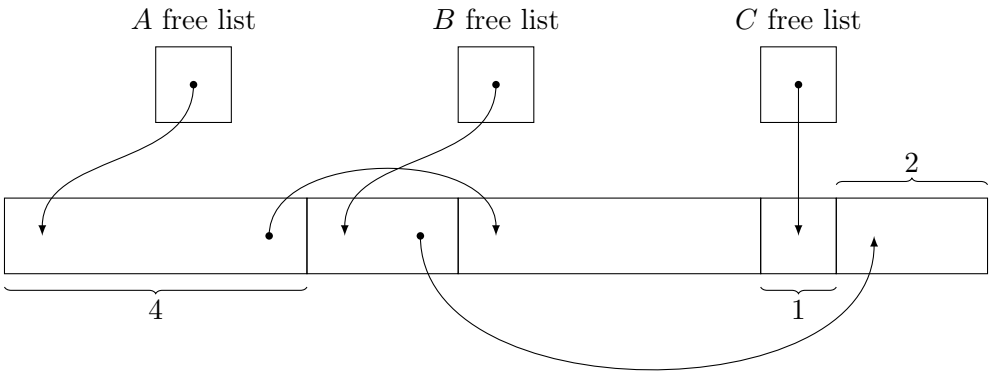
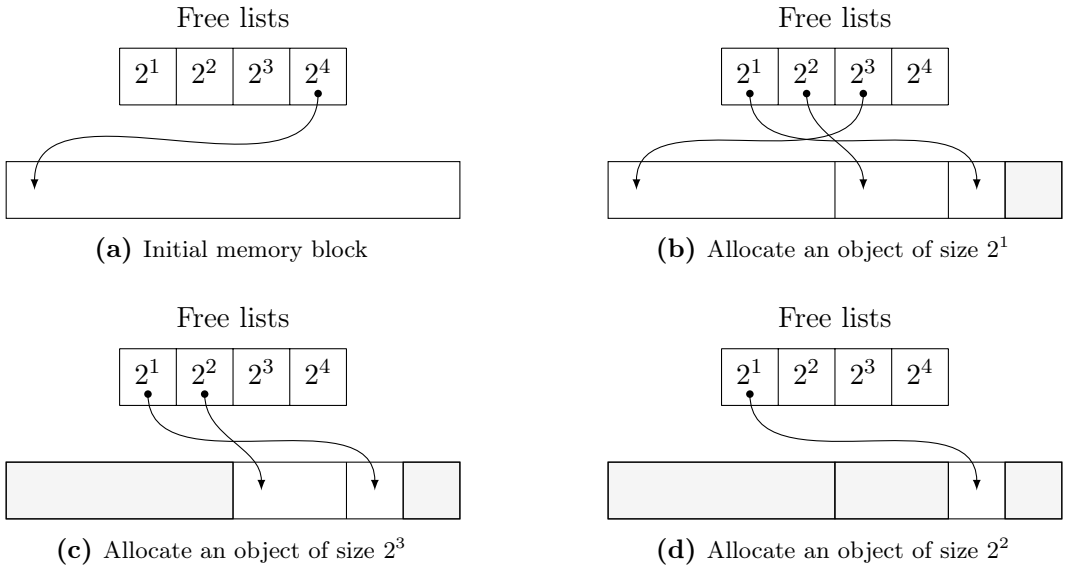


Figure 3.8: Record size-specific free lists on a shared heap (powers of two)

3.4.5 Buddy Memory

The Buddy Memory layout utilizes blocks of variable-sizes of the power-of-two, typically with one free list per power-of-two using a shared heap. When allocating an object of size m, we simply check the free lists for a free block of size n, where $n \geq m$. Is such a block found and if $n > m$, we split the block into two halves recursively, until we obtain the smallest block capable of storing m. When deallocating a block of size m, we do the action described above in reverse, thus merging the blocks again, where possible [13].



(a) Initial memory block

(b) Allocate an object of size 2^1

(c) Allocate an object of size 2^3

(d) Allocate an object of size 2^2

Figure 3.9: Buddy Memory block allocation example

Figure 3.9 illustrates an example of block splitting during allocation in the buddy system. Originally, one block of free memory is available. When allocating a record three factors smaller than the original block, three splits occurs.

This layout is somewhat of a middle ground between the previous three designs, addressing a number of problems found in these. The Buddy Memory layout uses a single heap for all

record-types, thus eliminating the problems related to moving adjacent heaps reversibly in a multi-heap layout. To optimize the problems around initializing a usable amount of variable-sized blocks in a shared heap, we simply initialize one large block in the buddy system, which we will split into smaller parts during execution and merge where possible when freed.

The main drawback from this layout is the amount of internal fragmentation. As we only allocate blocks of a power-of-two size, substantial internal fragmentation follows when allocating large records, i.e. allocating a block of size 128 for a record of size 65. However, as most real world programs uses much smaller sized records, we do not consider this a very frequent scenario. As discussed in section 3.4.3, we would split large records into chunks of $\sqrt{n}$ at the cost of additional bookkeeping.

Implementation-wise, this design would require doubling and halving of numbers related to the power-of-two. This action translates well into the reversible setting, as a simply bit-shifting directly gives us the desired result.

```
1  procedure malloc1(int p, int osize, int freelists[], int counter, int csize)
2    if (csize < osize) then
3        counter += 1
4        call double(csize)
5        call malloc1(p, osize, freelists, counter, csize)
6        uncall double(csize)
7        counter -= 1
8    else
9        if freelists[counter] != 0 then
10           p += freelists[counter]
11           freelists[counter] -= p
12
13           // Swap head of free list with next block of p
14           freelists[counter] ^= M(p)
15           M(p) ^= freelists[counter]
16           freelists[counter] ^= M(p)
17       else
18           counter += 1
19           call double(csize)
20           call malloc1(p, osize, freelists, counter, csize)
21           uncall double(csize)
22           counter -= 1
23           freelists[counter] += p
24           p += csize
25       fi freelists[counter] = 0 || p - csize != freelists[counter]
26   fi csize < osize
```

Listing 3.3: The Buddy Memory algorithm implemented in extended Janus

Listing 3.3 shows the Buddy Memory algorithm implemented in the extended Janus variant with local blocks from [29]. For simplification, object sizes are rounded to the nearest power-of-two during compile-time. The algorithm extends on the one heap per power-of-two algorithm presented in listing 3.2, page 53. The body of the allocation function is still executed recursively until a free list for a 2^n larger than the size of the object has been found. Once found, we continue searching until we have found a non-empty free list. If the non-empty free list for a 2^n larger than the object is found, the head of the list is popped and the popped block is split recursively, until a block the desired size is obtained. Throughout the splitting process, empty free lists are updated when a larger free block is split into a block which fits into those lists.

Since a split block is always added as two blocks to an empty free list, we can only merge adjacent

blocks if they are the only two blocks in a free list.

CHAPTER 4

Compilation

The following chapter presents the considerations and translation schemas used in the process of translating RoopL++ to the reversible low-level machine language Pisa. As RoopL++ is an extension of RoopL, many techniques are carried directly over, and have as such been left out.

Before presenting the RoopL++ compiler, a brief overview of the memory layout and modeling of the RoopL compiler, which the RoopL++ compiler is a continuation of, is provided.

4.1 The RoopL to Pisa Compiler

Haulund presented a proof-of-concept compiler along with the design for RoopL. The compiler translates well-typed RoopL programs into the reversible machine language Pisa in [11]. The RoopL compiler (RoopLC) is written in Haskell and hosted at `https://github.com/TueHaulund/ROOPLC`.

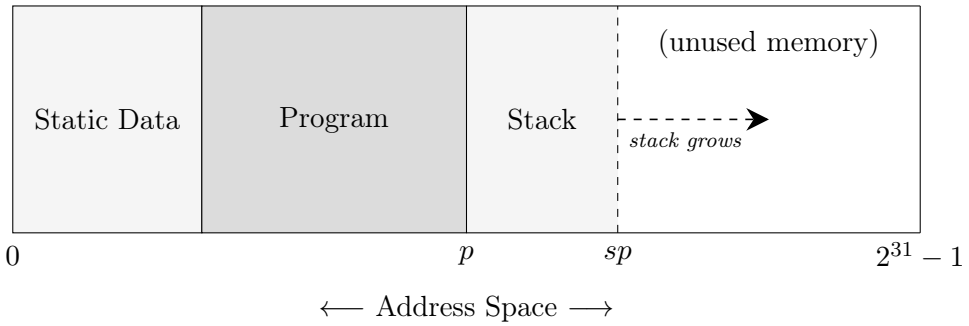


Figure 4.1: Memory layout of a RoopL program, originally from [11]

Figure 4.1 shows the memory layout of a compiled RoopL program. The layout consists of a static storage segment, the program segment and the stack.

The object model is simple and only features one additional word for storing the address of the virtual table for the object class. Figure 4.2 shows the prefixing for three simple classes modeling geometric shapes.

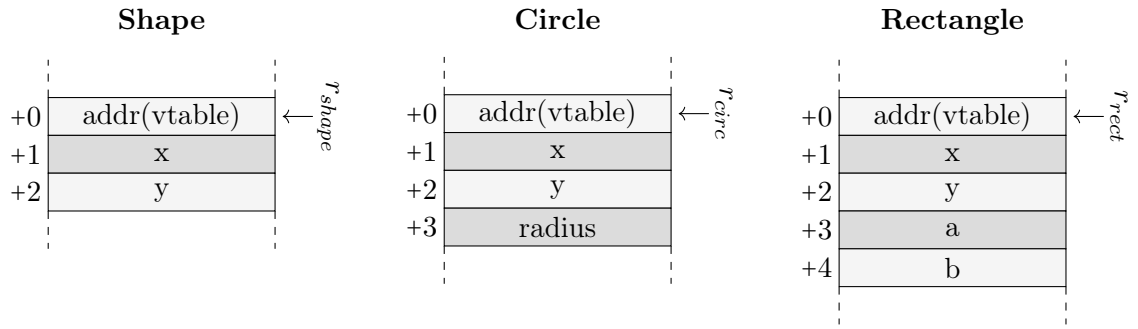


Figure 4.2: Illustration of prefixing in the memory layout of 3 Roopl objects, originally from [11]

4.2 Roopl++ Memory Layout

Roopl++ builds upon the memory layout of its predecessor with dynamic memory management. The reversible Buddy Memory heap layout presented in section 3.4.5 is utilized in Roopl++ as it is an interesting layout, addressing a number of disadvantages found in other considered layouts, naturally translates into a reversible setting with one simple restriction (i.e only blocks which are heads of their respectable free lists are allocatable) and since its only drawback is dismissible in most real world scenarios.

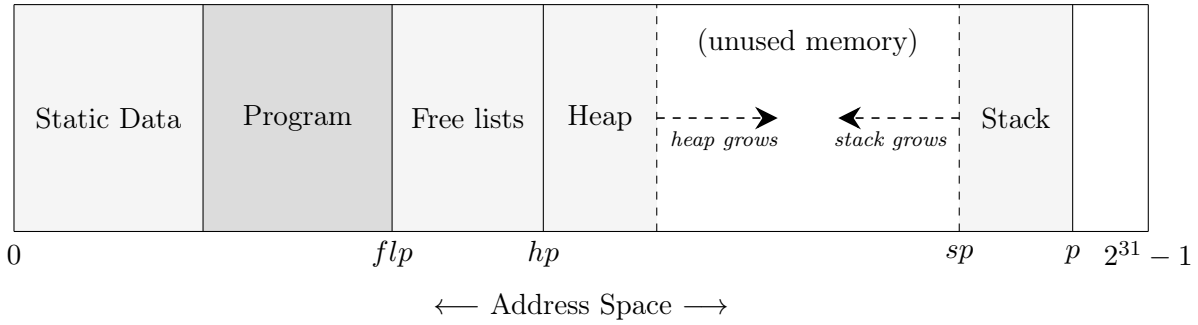


Figure 4.3: Memory layout of a Roopl++ program

Figure 4.3 shows the full layout of a Roopl++ program stored in memory.

- As with Roopl, the static storage segment contains load-time labelled **`DATA`** pseudo-instructions, initialized with virtual function tables and other static data needed by the translated program.
- The program segment is stored right after the static storage and contains the translated Roopl++ program instructions.
- The free lists maintained by the Buddy Memory heap layout is placed right after the program segment, with the *free list pointer* flp pointing at the first free list. The free lists are simply the address pointing to the first block of its respective size. The free lists are stored such that the free list at address $flp + i$ corresponds to the free list of size 2^{i+1}.
- The heap begins directly following the free lists. Its beginning is marked by the *heap pointer* (hp).

- Unlike in ROOPL, where the stack grows upwards, the ROOPL++ stack grows downwards and begins at address p. The stack remains a LIFO structure, analogously to ROOPL.

As mentioned in the previous chapter, we assume an underlying reversible operating system providing us with additional memory when needed. With no real way of simulating this, the ROOPL++ compiler places the stack at a fixed address p and sets one free block in the largest 2^n free list initially. The number of free lists and the address p is configurable in the source code, but defaults to 10 free lists, meaning initially one block of size 1024 is available and the stack is placed at address 1024 words after the heap.

In traditional compilers, the heap pointer usually points to the end of the heap. For reasons stated above, we never grow the heap as we start with a heap of fixed size. As such, the heap pointer simply points to the beginning of the heap.

The heap can simply be expanded by adding another block of the largest possible size and storing the address of the respective free list.

In the following sections of this chapter, we will present various translation techniques. In these translations, we will make use of a number of PISA pseudo-instructions to subtract integer values from registers and pushing/popping to the program stack. The pseudo-instructions are shown in figure 4.4 and are modified from [11], as the direction of the program stack is flipped in ROOPL++.

$$\texttt{SUBI}\ \ r\ \ i \quad \overset{\text{def}}{=\!=} \quad \texttt{ADDI}\ \ r\ \ -i$$

$$\texttt{PUSH}\ \ r \quad \overset{\text{def}}{=\!=} \quad \Big[\texttt{EXCH}\ \ r\ \ r_{sp}\ ,\ \ \texttt{SUBI}\ \ r_{sp}\ \ 1\Big]$$

$$\texttt{POP}\ \ r \quad \overset{\text{def}}{=\!=} \quad \Big[\texttt{ADDI}\ \ r_{sp}\ \ 1\ ,\ \ \texttt{EXCH}\ \ r\ \ r_{sp}\Big]$$

Figure 4.4: Definition of pseudoinstructions **SUBI**, **PUSH** and **POP**, modified from [11]

4.3 Inherited ROOPL features

As mentioned, a number of features from ROOPL carries over to ROOPL++.

The dynamic dispatching mechanism presented in [11] is inherited. As such, the invocation of a method implementation is based on the type of the object at run time. Virtual function tables are still the implementation strategy used in the dynamic dispatching implementation.

Evaluation of expressions and control flow remains unchanged.

For completeness, object blocks are included and still stack allocated as their life time is limited to the scope of their block and the dynamic allocation process is quite expensive in terms of register pressure and number of instructions compared to the stack allocated method implemented in the ROOPL compiler.

4.4 Program Structure

The program structure of a translated ROOPL++ is analogous to the program structure of a ROOPL program with the addition of free lists and heap initialization. The full structure is shown in figure 4.5.

```
(1)                ·····                          ; Static data declarations
(2)                ·····                          ; Code for program class methods
(3)  start :       START                          ; Program starting point
(4)                ADDI    r_flps   p             ; Initialize free lists pointer
(5)                XOR     r_hp     r_flps        ; Initialize heap pointer
(6)                ADDI    r_hp     size_fls      ; Initialize heap pointer
(7)                XOR     r_b      r_hp          ; Store address of initial free memory block in r_b
(8)                ADDI    r_flps   size_fls      ; Index to end of free lists
(9)                SUBI    r_flps   1             ; Index to last element of free lists
(10)               EXCH    rb       r_flps        ; Store address of first block in last element of free lists
(11)               ADDI    r_flps   1             ; Index to end of free lists
(12)               SUBI    r_flps   s             ; Index to beginning of free lists
(13)               XOR     r_sp     r_hp          ; Initialize stack pointer
(14)               ADDI    r_sp     offset_stack  ; Initialize stack pointer
(15)               SUBI    r_sp     size_m        ; Allocate space for main object
(16)               XOR     r_m      r_sp          ; Store address of main object in r_m
(17)               XORI    r_v      label_vt      ; Store address of vtable in r_v
(18)               PUSH    r_v                    ; Push address of vtable onto stack
(19)               PUSH    r_m                    ; Push 'this' onto stack
(20)               BRA     label_m                ; Call main procedure
(21)               POP     r_m                    ; Pop 'this' from stack
(22)               POP     r_v                    ; Pop vtable address into r_v
(23)               XORI    r_v      label_vt      ; Clear r_v
(24)               XOR     r_m      r_sp          ; Clear r_m
(25)               ADDI    r_sp     size_m        ; Deallocate space of main object
(26)               SUBI    r_sp     offset_stack  ; Clear stack pointer
(27)               XOR     r_sp     r_hp          ; Clear stack pointer
(28)  finish :     FINISH                         ; Program exit point
```

Figure 4.5: Overall layout of a translated ROOPL++ program

The following PISA code block initializes the free lists pointer, the heap pointer, the stack pointer, allocates the main object on the stack, calls the main method, deallocates the main object and finally clears the free lists, heap and stack pointers.

The free lists pointer is initialized by adding the base address, which varies with the size of the translated program, to the register r_{flps}. In figure 4.5 the base address is denoted by p.

The heap pointer is initialized directly after the free lists pointer by adding the size of the free lists. One free list is the size of one word and the full size of the free lists is configured in the source code (defaulted to 10, as described earlier).

Once the heap pointer and free lists pointer is initialized, the initial block of free memory is placed in the largest free lists by indexing to said list, by adding the length of the list of free lists, subtracting 1, writing the address of the first block (which is the same address as the heap

pointer, which points to the beginning of the heap) to the last free list and then resetting the free lists pointer to point to the first list again, afterwards.

The stack pointer is initialized simply by adding the stack offset to the heap pointer register r_{hp}. The stack offset is configured in the source code and defaults to 1024, as described earlier in this chapter. As such, the heap and the stack each have 1024 words of space to utilize. Once the stack pointer has been initialized, the main object is allocated on the stack and the main method called, analogously to the ROOPL program structure.

When the program terminates and the main method returns, the main object is popped from the stack and deallocated and the stack pointer is cleared. The heap and free list pointer not intentionally not cleared to simulate future program simulation using these pointers. The contents of the free lists and whatever is left on the heap is untouched at this point. It is the programmers responsibility to free dynamically allocated objects in their ROOPL++ program. Furthermore, depending on the deallocation order, we might not end up with exactly one fully merged block in the end and as such, we do not invert the steps taken to initialize this initial free memory block. Analogously to ROOPL, the values of the main object are left in the stack section of memory.

4.5 Buddy Memory Translation

As briefly mentioned in section 4.2, the Buddy Memory layout was selected as the memory manager layout as it addressed a number of problems related to fragmentation and initialization. The Buddy Memory layout could be converted to a reversible section with only a few restrictions and side effects, which will be described in this section. Firstly, we present the algorithm translated to PISA. As the algorithm is quite lengthy, it will be broken down into smaller chunks. The full translation is shown in appendix A.

The Buddy Memory algorithm consists of three JANUS procedures; the entry point **malloc**, the recursion body **malloc1** and a helper function **double**. The entry point is omitted for now, as it differs depending on which type of memory object we are allocating and will be presented in sections 4.6 and 4.8.1. The helper function can be implemented using a single instruction in PISA for our specific case of doubling number in the power-of-two, which we will show later.

(1)	$malloc1_{top}$ **:**	**BRA**	$malloc1_{bot}$	; Receive jump
(2)		**POP**	r_{ro}	; Pop return offset from the stack
(3)		·····		; Inverse of **(7)**
(4)	$malloc1_{entry}$ **:**	**SWAPBR**	r_{ro}	; Malloc1 entry and exit point
(5)		**NEG**	r_{ro}	; Negate return offset
(6)		**PUSH**	r_{ro}	; Store return offset on stack
(7-63)		·····		; Allocation code
(64)	$malloc1_{bot}$ **:**	**BRA**	$malloc1_{top}$	; Jump

Figure 4.6: Dynamic dispatch approach for entering the allocation subroutine

Before we go into depth with the translation of the algorithm, we consider the mechanism for triggering the allocation subroutine. Naively, we could generate the entire block of code required

for allocation for every **new** or **delete** statement in the target program. This approach would severely limit the amount of objects we could allocate as the register pressure of the Buddy Memory implementation is quite high, as we be shown in this section. Instead, we can utilize the dynamic dispatching technique, which also is used for method invocations. This way, we only generate the allocation instructions once, and then simply jump to the entry point from different locations in the program. Figure 4.6 outlines the structure for this approach. By using the **SWAPBR** instruction we can jump from multiple points of origin in the compiled program and internally for the recursive needs of the algorithm itself.

```
1 if (csize < osize) then
2     // outer if-then
3 else
4     if freelists[counter] != 0 then
5         // inner if-then
6     else
7         // inner else
8     fi (freelists[counter] = 0 ||
9         p - csize != freelists[counter])
10 fi csize < osize
```

```
(7)                 ......                  ; Code for r_fl ← addr(fl[c])
(8)                 ......                  ; Code for r_block ← [[fl[c]]]
(9)                 ......                  ; Code for r_e1o ← [[c_size < object_size]]
(10)                XOR  r_t   r_e1o        ; Copy value of c_size < object_size into r_t
(11)                ......                  ; Inverse of (9)
(12)  o_test :      BEQ  r_t   r_0   o_testf ; Receive jump
(13)                XORI r_t   1            ; Clear r_t
(14-21)             ......                  ; Code for outer if-then statement
(22)                XORI r_t   1            ; Set r_t = 1
(23)  o_assert_t :  BRA  o_assert           ; Jump
(24)  o_test_f :    BRA  o_test             ; Receive jump
(25)                ......                  ; Code for r_e1i ← [[addr(fl[c]) ≠ 0]]
(26)                XOR  r_t2  r_e1i        ; Copy value of r_e1i into r_t2
(27)                ......                  ; Inverse of (25)
(28)  i_test :      BEQ  r_t2  r_0   i_testf ; Receive jump
(29)                XORI r_t2  1            ; Clear r_t2
(30-34)             ......                  ; Code for inner if-then statement
(35)                XORI r_t2  1            ; Set r_t2 = 1
(36)  i_assert_t :  BRA  i_assert           ; Jump
(37)  i_test_f :    BRA  i_test             ; Receive jump
(38-47)             ......                  ; Code for inner else statement
(48)  i_assert :    BNE  r_t2  r_0   i_assert_t ; Receive jump
(49)                EXCH r_tmp r_fl         ; Load address of head of current free list
(50)                SUB  r_p   r_cs         ; Set p to previous block address
(51)                ......                  ; r_e2i1 ← [[p − c_size ≠ addr(fl[c])]]
(52)                ......                  ; r_e2i2 ← [[addr(fl[c]) = 0]]
(53)                ......                  ; r_e2i3 ← [[(p − c_size ≠ addr(fl[c])) ∨ (addr(fl[c]) = 0)]]
(54)                XOR  r_r2  r_e2i3       ; Copy value of r_e2i3 into r_t2
(55)                ......                  ; Inverse of (53)
(56)                ......                  ; Inverse of (52)
(57)                ......                  ; Inverse of (51)
(58)                ADD  r_p   r_cs         ; Inverse of (50)
(59)                EXCH r_tmp r_fl         ; Inverse of (49)
(60)  o_assert :    BNE  r_t   r_0   o_assert_t ; Receive jump
(61)                ......                  ; Code for r_e2o ← [[c_size < object_size]]
(62)                XOR  r_t   r_e2o        ; Copy value of c_size < object_size into r_t
(63)                ......                  ; Inverse of (61)
```

Figure 4.7: PISA translation of the nested conditionals in the Buddy Memory algorithm

The main recursion body of the algorithm, **malloc1** from listing 3.3, page 56 consists of two conditionals, in which one is nested in the else branch of the outer conditional. Figure 4.7 shows the translation structure of the nested conditional pair, using the translation techniques for conditionals presented in [1].

The nested conditionals contain large amounts of boilerplate code for evaluating the various expressions of the conditionals. As these conditionals requires comparisons with contents of the free lists, we must be careful with extracting and storing the values in the free list.

We have three statements to translate from here. The outer **if-then** statement, the inner **if-then** statement and the inner **else** statement.

```
1 counter += 1
2 call double(csize)
3 call malloc1(p, osize, freelists,
4              counter, csize)
5 uncall double(csize)
6 counter -= 1
```

```
(14) ADDI r_c 1          ; Counter++
(15) RL   r_sc 1         ; Call double(c_size)
(16) ·····               ; Inverse of (7)
(17) ·····               ; Code for pushing temp reg values to stack
(18) BRA  malloc1_entry  ; Call malloc1())
(19) ·····               ; Inverse of (17)
(20) RR   r_sc 1         ; Inverse of (15)
(21) SUBI r_c 1          ; Inverse of (14)
```

Figure 4.8: Pisa translation of the outer **if-then** statement for the Buddy Memory algorithm

Figure 4.8 shows the translation of the outer **if-then** statement. As briefly mentioned, we can utilize the right bit shift instruction of Pisa, **RL**, in place of the **double** helper procedure from the Janus implementation. By using a simple bit shift, we are able to maintain reversibility elegantly when doubling or halving numbers in the power-of-two. This statement also contains one of the careful storage operations of the free list values, in instruction **(16)**. Before we recursively branch to the entry point, we must place the previously extracted address of the head of the free list back into the free list. This is also the reason for instruction **(3)** in figure 4.6. Furthermore, we must push all temporary evaluated expression values to the stack, so they can be popped when we return.

```
1 p += freelists[counter]
2 freelists[counter] -= p
3
4 // Swap head of free list
5 // with p's next block
6 freelists[counter] ^= M(p)
7 M(p) ^= freelists[counter]
8 freelists[counter] ^= M(p)
```

```
(30) ADD  r_p   r_block ; Copy address of the current block to p
(31) SUB  r_block r_p   ; Clear r_block
(32) EXCH r_tmp r_p     ; Load address of next block
(33) EXCH r_tmp r_fl    ; Set address of next block as new head of free list
(34) XOR  r_tmp r_p     ; Clear address of next block
```

Figure 4.9: Pisa translation of the inner **if-then** statement for the Buddy Memory algorithm

Figure 4.9 shows the translation of the inner **if-then** statement. This statement translates easily using the **EXCH** instructions to swap with memory locations as simulated in the Janus code.

```
1 counter += 1
2 call double(csize)
3 call malloc1(p, osize, freelists,
4              counter, csize)
5 uncall double(csize)
6 counter -= 1
7 freelists[counter] += p
8 p += csize
```

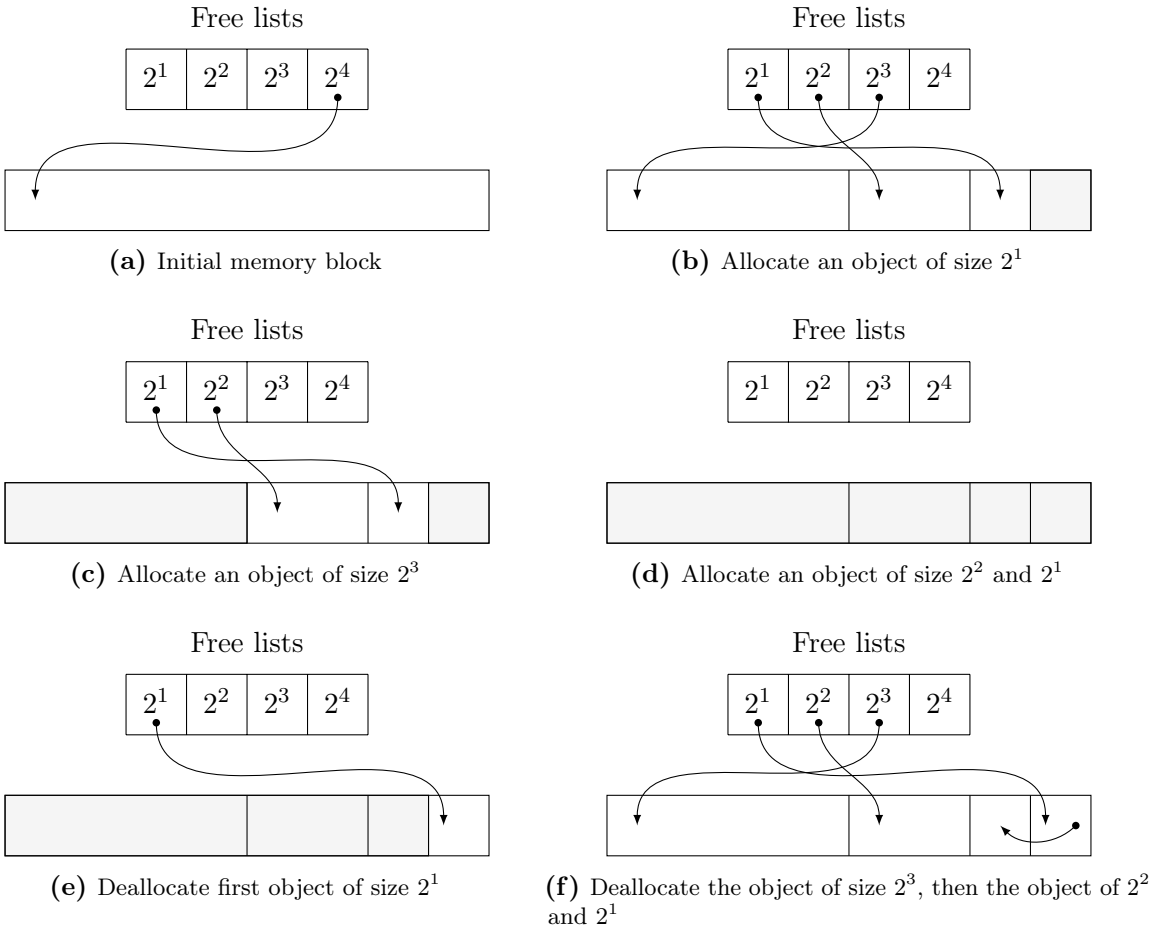
```
(38) ADDI r_c 1          ; Counter++
(39) RL   r_sc 1         ; Call double(c_size)
(40) ·····               ; Push temp reg values to stack
(41) BRA  malloc1_entry  ; Call malloc1())
(42) ·····               ; Inverse of (40)
(43) RR   r_sc 1         ; Inverse of (39)
(44) SUBI r_c 1          ; Inverse of (38)
(45) XOR  r_tmp r_p      ; Copy current address of p
(46) EXCH r_tmp r_fl     ; Store address of p in free list
(47) ADD  r_p   r_cs     ; Split block by p = other half of block
```

Figure 4.10: Pisa translation of the inner **else** statement for the Buddy Memory algorithm

The last statement translation is the inner **else** statement shown in figure 4.10. This statement is almost identical to the outer **if-then** with the addition of the block splitting code. The block

splitting is done in three instructions. First, the current block we are examining is set as the new head of the current free list. Afterwards the current free list block size is added to out pointer p, resulting in an effectively split block.

During the design of the reversible Buddy Memory algorithm limitations on the merging and splitting conditions were required to ensure reversibility. Since a split block is always added as two blocks to an empty free list, we can only merge adjacent blocks if they are the only two blocks in a free list. In the irreversible Buddy Memory algorithm block merging can occur in any place of the free list, but in the reversible version, we can only merge blocks at the start of the free list to maintain reversibility. The effect of this limitation prevents us from returning to one final block of free memory, if the deallocation order is not exactly opposite of the allocation order.



(a) Initial memory block

(b) Allocate an object of size 2^1

(c) Allocate an object of size 2^3

(d) Allocate an object of size 2^2 and 2^1

(e) Deallocate first object of size 2^1

(f) Deallocate the object of size 2^3, then the object of 2^2 and 2^1

Figure 4.11: Non-opposite deallocation results in a different free list after termination

Figure 4.11 shows how alternative deallocation orders results in different free lists, compared to the original given to some function. However, as discussed in section 3.2, we can consider every collection of Buddy Memory free lists equivalent, as a later computation can take another set of free lists and still execute its function, as long as the free lists have the required blocks available.

4.6 Object Allocation and Deallocation

Now that we have the main allocation mechanism in place and a method of accessing it through a label and a **SWAPBR** instruction, we can continue translating the **malloc** procedure entry point from listing 3.3 on page 56.

```
1 procedure malloc(int p, int osize,
2                  int freelists[])
3     local int counter = 0
4     local int csize = 2
5     call malloc1(p, osize, freelists,
6                  counter, csize)
7     delocal int csize = 2
8     delocal int counter = 0
```

```
(1)  l_malloc_top :  BRA     l_malloc_bot      ; Receive jump
(2)  l_malloc :      SWAPBR  r_o               ; Entry and exit point
(3)                  NEG     r_o               ; Negate return offset
(4)                  ADDI    c_size    2       ; Init c_size
(5)                  XOR     r_counter r_0     ; Init counter
(6)                  ···                       ; Pop r_p and object_size from stack
(7)                  PUSH    r_0               ; Push r_o
(8)                  BRA     l_malloc1         ; call malloc1()
(9)                  POP     r_0               ; Inverse of (7)
(10)                 ···     r_0               ; Inverse of (6)
(11)                 XOR     r_counter r_0     ; Inverse of (5)
(12)                 SUBI    c_size    2       ; Inverse of (4)
(13) l_malloc_bot :  BRA     l_malloc_top      ; Jump
```

Figure 4.12: PISA translation of the **malloc** procedure entry point of Buddy Memory algorithm

Figure 4.12 shows the translated **malloc** procedure. In addition to the original procedure, we also push the current return offset register value to the stack before we branch to the **malloc1** implementation, to ensure we have a zero-cleared register before starting the allocation process. The translated procedure assumes that the pointer to the object we are allocating and its size are on top of the stack before entering the block. This translated procedure serves as the entry point for the allocation subroutine as it is also only generated once. Each **new** and **delete** statement branches to the l_{malloc} label to begin an allocation or a deallocation.

new c x

```
(1)  ···                          ; Push registers
(2)  ···                          ; Code for r_t ← x_size
(3)  PUSH  r_t                    ; Push r_t
(4)  PUSH  r_p                    ; Push r_p
(5)  BRA   l_malloc               ; Allocate
(6)  POP   r_p                    ; Inverse of (4)
(7)  POP   r_t                    ; Inverse of (3)
(8)  ···                          ; Inverse of (2)
(9)  ···                          ; Inverse of (1)
(10) ···                          ; Code for r_v ← [[addr(x)]]
(11) XORI  r_t   label_vt         ; Store address of vtable in r_t
(12) EXCH  r_t   r_p              ; Store vtable in new object
(13) ADDI  r_p   offset_ref       ; Index to ref count pos
(14) XORI  r_t   1                ; Init ref count
(15) EXCH  r_t   r_p              ; Store ref count
(16) SUBI  r_p   offset_ref       ; Inverse of (13)
(17) EXCH  r_p   r_v              ; Store address in variable
(18) ···                          ; Inverse of (10)
```

delete c x

```
(1)  ···                          ; Code for r_p ← [[addr(x)]]
(2)  EXCH  r_t   r_p              ; extract vtable from object
(3)  XORI  r_t   label_vt         ; clear address of vtable in r_t
(4)  ADDI  r_p   offset_ref       ; Index to ref count pos
(5)  EXCH  r_t   r_p              ; Extract ref count
(6)  XORI  r_t   1                ; Clear ref count
(7)  SUBI  r_p   offset_ref       ; Inverse of (4)
(8)  ···                          ; Push registers except r_p, r_t
(9)  ···                          ; Code for r_t ← x_size
(10) PUSH  r_t                    ; Push r_t
(11) PUSH  r_p                    ; Push r_p
(12) RBRA  l_malloc               ; Deallocate
(13) POP   r_p                    ; Inverse of (11)
(14) POP   r_t                    ; Inverse of (10)
(15) ···                          ; Inverse of (9)
(16) ···                          ; Inverse of (8)
(17) ···                          ; Inverse of (1)
```

Figure 4.13: PISA translation of heap allocation and deallocation for objects

Figure 4.13 shows how each **new** and **delete** statement for objects are translated during compilation. They are simply inverse of each other. For allocation, the object pointer and its size are pushed to the stack and then a jump to the malloc entry point is executed. After allocation, the virtual table and reference count are stored in the first two words of the allocated memory. Note how deallocation jumps and flips the direction of execution using the **RBRA** instruction, which then runs the allocation process in reverse. In the figure x_{size} denotes the computed size of objects with class c, plus two, to account for the virtual table pointer and reference count space, rounded up to nearest power-of-two.

construct c x $\quad s \quad$ **destruct** x

```
(1) XOR   r_x  r_sp   ; Store address of new object x in r_x
(2) PUSH  r_x         ; Push r_x to the stack
(3) ......            ; Code for new c x
(4) ......            ; Code for statement s
(5) ......            ; Code for delete c x
(6) POP   r_x         ; Pop r_x from the stack
(7) XOR   r_x  r_sp   ; Clear r_x
```

Figure 4.14: PISA translation of a ROOPL++ object block

Figure 4.14 shows the updated translation technique for object blocks. In ROOPL, the object blocks allocated their objects on the stack, but in ROOPL++, we can now allocate them on the heap. to facilitate this, we simply execute the exact same instructions as in **new** and **delete** statements, with body statement execution code in between. As described in section 2.6, the **construct**/**desctruct** block can be considered syntactic sugar, and its usage in a real world example would probably be limited.

4.7 Referencing

As mentioned, one of the main strengths of ROOPL++ in terms of increased expressiveness is allowance of multiple references to objects and arrays. When an object or array is constructed we allocate enough space to hold an additional *reference counter* which is initialized to 1. For each reference copied using the **copy**-statement, we incrementally increase the reference counter by one. When we **uncopy** a reference, the reference counter is decreased by one. The object or array cannot be deconstructed until its reference counter has been returned to 1 as we would have a reference pointer to cleared memory in the heap. Such references are known as dangling pointers.

Figure 4.15 shows the object layout of ROOPL++ objects with the added space for the reference counting from the original ROOPL model in figure 4.2 on page 59.

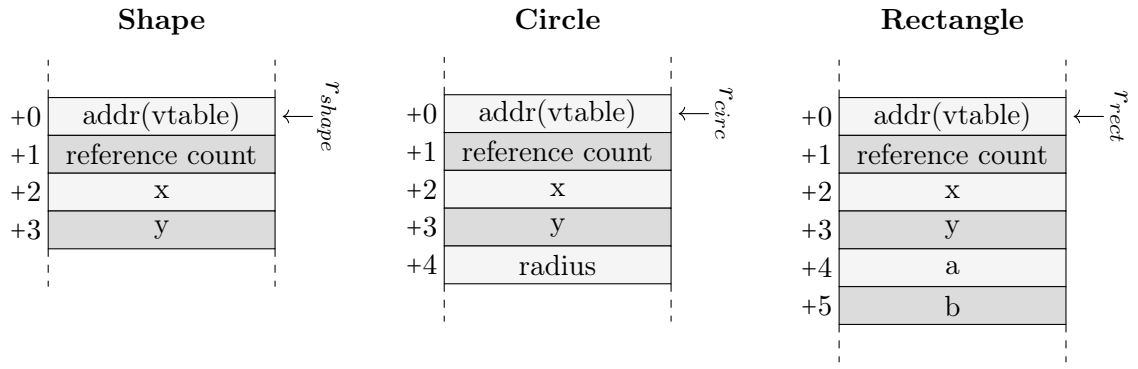


Figure 4.15: Illustration of prefixing in the memory layout of three ROOPL++ objects

Figure 4.16 shows the translated PISA code for the **copy** and **uncopy** statements. As shown, they are both very simple and each others inverse. For copying, the address of the passed variable x is simply copied into the zero-cleared value of x' and the reference count incremented by one. For deletion, the address is cleared and the reference count decremented. Copying and clearing is done through the **XOR** instruction. These translations features no error handling, but a solution is discussed in section 4.9.

copy $c\ x\ x'$

```
(1)  ...                          ; Code for r_p ← addr(x)
(2)  ...                          ; Code for r_cp ← value(x')
(3)  XOR   r_cp  r_p              ; Copy address of x into x'
(4)  ADDI  r_p   offset_ref       ; Index to reference count address
(5)  EXCH  r_t   r_p              ; Extract reference count
(6)  ADDI  r_t   1                ; Increment reference count
(7)  EXCH  r_t   r_p              ; Store updated reference count
(8)  SUBI  r_p   offset_ref       ; Inverse of (3)
(9)  ...                          ; Inverse of (2)
(10) ...                          ; Inverse of (1)
```

uncopy $c\ x\ x'$

```
(1)  ...                          ; Code for r_p ← addr(x)
(2)  ...                          ; Code for r_cp ← value(x')
(3)  XOR   r_cp  r_p              ; Clear address of x from x'
(4)  ADDI  r_p   offset_ref       ; Index to reference count address
(5)  EXCH  r_t   r_p              ; Extract reference count
(6)  SUBI  r_t   1                ; Decrement reference count
(7)  EXCH  r_t   r_p              ; Store updated reference count
(8)  SUBI  r_p   offset_ref       ; Inverse of (3)
(9)  ...                          ; Inverse of (2)
(10) ...                          ; Inverse of (1)
```

Figure 4.16: PISA translation of the reference copying and deletion statements

4.8 Arrays

The fixed-sized arrays in ROOPL++ are also heap allocated to allow dynamic lifetime. The array memory layout is presented in figure 4.17. As shown, the arrays feature two additional fields to store the size of the array and the reference count. Additionally, integer arrays store their values directly in the array, while object arrays are a simple pointer stores.

As the size of a ROOPL++ array is determined by a passed expression evaluation, it is unknown at compile time. This also means that out-of-bounds checking cannot be conducted during compilation. A possible solution for this is presented in section 4.9.

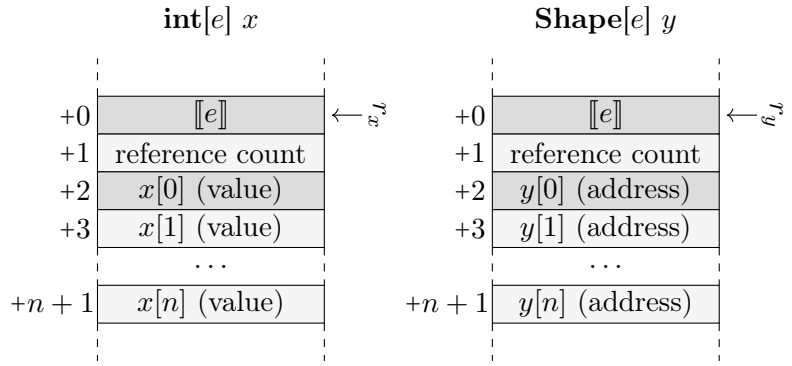


Figure 4.17: Illustration of prefixing in the memory layout of two Roopl++ arrays

4.8.1 Construction and Destruction

As Roopl++ arrays also are heap allocated, the buddy allocation implementation is also used for allocating arrays. The only difference between object and array allocation is that no virtual table is stored in the allocated space while the offsets for the reference counter are shared for both types. Due to this fact, **copy** and **uncopy** Pisa blocks generated during compile time are exactly the same for arrays and objects, as shown in the previous section.

new $a[e]$ x

```
(1)  ···                                  ; Push registers
(2)  ···                                  ; Code for r_t ← [[e]] + 2
(3)  PUSH  r_t                            ; Push r_t
(4)  PUSH  r_p                            ; Push r_p
(5)  BRA   l_malloc                       ; Allocate array
(6)  POP   r_p                            ; Inverse of (4)
(7)  POP   r_t                            ; Inverse of (3)
(9)  ···                                  ; Inverse of (1)
(10) ···                                  ; Code for r_v ← [[addr(x)]]
(11) SUBI  r_t   2                        ; r_t ← [[e]]
(12) EXCH  r_t   r_p                      ; Store size in new array
(13) ADDI  r_p   offset_ref               ; Index to ref count pos
(14) XORI  r_t   1                        ; Init ref count
(15) EXCH  r_t   r_p                      ; Store ref count
(16) SUBI  r_p   offset_ref               ; Inverse of (13)
(17) EXCH  r_p   r_v                      ; Store address in variable
(18) ···                                  ; Inverse of (10)
```

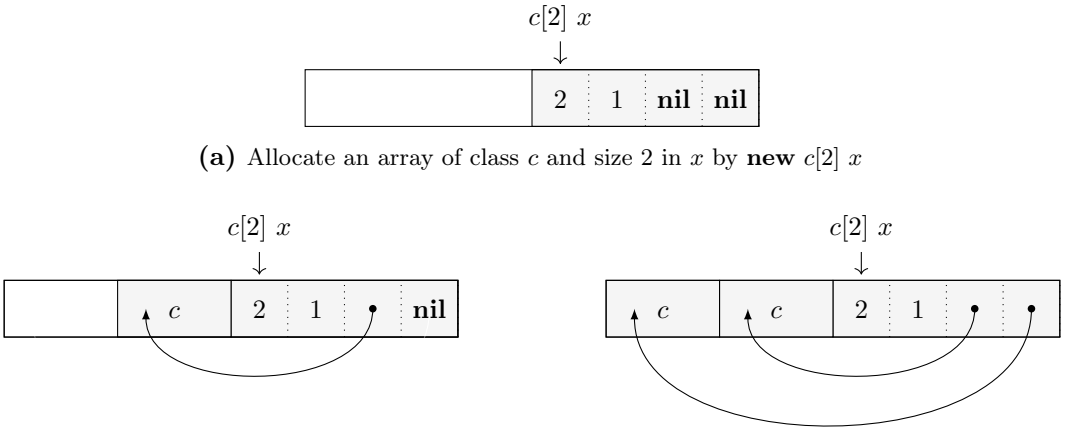
delete $a[e]$ x

```
(1)  ···                                  ; Code for r_p ← [[addr(x)]]
(2)  ···                                  ; Code for r_v ← [[e]]
(3)  ADDI  r_p   offset_ref               ; Index to ref count pos
(4)  EXCH  r_t   r_p                      ; Extract ref count
(5)  XORI  r_t   1                        ; Clear ref count
(6)  SUBI  r_p   offset_ref               ; Inverse of (3)
(7)  EXCH  r_t   r_p                      ; extract size from object
(8)  XORI  r_t   r_v                      ; clear size in r_t
(9)  ···                                  ; Push registers except r_p, r_v
(10) ADDI  r_v   2                        ; Actual size of array
(11) PUSH  r_v                            ; Push r_v
(12) PUSH  r_p                            ; Push r_p
(13) RBRA  l_malloc                       ; Deallocate array
(14) POP   r_p                            ; Inverse of (12)
(15) POP   r_v                            ; Inverse of (11)
(16) SUBI  r_v   2                        ; Inverse of (10)
(17) ···                                  ; Inverse of (9)
(18) ···                                  ; Inverse of (2)
(19) ···                                  ; Inverse of (1)
```

Figure 4.18: Pisa translations of array allocation and deallocation statements

Figure 4.18 shows the translation schemes used for array allocation and deallocation. As said, these are almost identical to the object allocation and deallocation schemes presented in figure 4.13 on page 66. Classes are analyzed during a compilation phase and their allocation size, the object size + 2 (for virtual table and reference counter) rounded up to nearest power-of-two. The

size of arrays cannot be determined during compilation, as that would require evaluating the expression passed to the initialization call, and as such, we add the overhead needed directly in the allocation and deallocation instructions. While the two blocks are code are not exact opposites they are functionally inverse of each other. An extra **XORI** instruction on line **(8)** in the deallocation block has been included to clear the stored array size using the value of the passed expression and further use this size for the inverse **malloc** subroutine.



(a) Allocate an array of class c and size 2 in x by **new** $c[2]$ x

(b) Assign first cell with **new** c $x[0]$

(c) Assign next cell with **new** c $x[1]$

Figure 4.19: Illustration of array memory storage layout

Figure 4.19 shows how object arrays simply contain pointers to allocated objects. For integer arrays, the cell values would stored directly in the allocated array space instead.

4.8.2 Array Element Access

Array elements are simply passed as any other variable to methods or statements. Based on the variable type, compilation of various statements individually determines whether the address or the value of the passed variable should be used for the compiling the statement. For arrays, this is no different. If an integer array element is passed, it is treated just liked a regular integer variable. For an object array element, it is treated just like a regular object variable.

4.9 Error Handling

While a program written in RoopL++ might be syntactically valid and well-typed, this is not a guarantee that it executes successfully. A number of conditions exist, which cannot be determined at compile time, which in turn results in erroneous executed code. Haulund describes the following conditions:

- If the entry expression of a conditional is **true**, then the exit assertion should also be **true** after executing the then-branch.

- If the entry expression of a conditional is **false**, then the exit assertion should also be **false** after executing the else-branch.
- The entry expression of a loop should initially be **true**.
- If the exit assertion of a loop is **false**, then the entry expression should also be **false** after executing the loop-statement.
- All instance variables should be zero-cleared within an object block before the object is deallocated.
- The value of a local variable should always match the value of the delocal-expression after the block statement has executed [11].

The extensions made to RoopL in RoopL++ brings forth a number of additional conditions:

- All fields of an object instance should be zero-cleared before the object is deallocated using the **delete** statement.
- All cells of an instance should be zero-cleared before the array is deallocated using the **delete** statement.
- Local object blocks should have their fields zero-cleared after the execution of the block statement.
- Local array blocks should have their cells zero-cleared after the execution of the block statement.
- If the value of a local object variable is exchanged during its block statement and the new value is an object reference, this object must have its fields zero-cleared after the execution of the block statement.
- If the value of a local array variable is exchanged during its block statement and the new value is an array reference, this array must have its cell zero-cleared after the execution of the block statement.
- The variable in the **new** statement must be zero-cleared beforehand.
- The variable in the **copy** statement must be zero-cleared beforehand.
- An object variable must be initialized using **new** or **copy** before its methods can be called.
- An array variable must be initialized using **new** or **copy** before its fields can be accessed.
- Array cell indices must be within bounds defined in the expression passed during initialization.
- Only one reference to an object or an array must exist when executing the **delete** statement.
- Swapping cell values between a subtype A variable and parent-type B array is only allowed if the value stored in the variable is also A afterwards.

It is the programmer's responsibility to meet these conditions. As these conditions, in general, cannot be determined at compile time, undefined program behaviour will occur as the termination will continue silently, resulting in erroneous program state. We can insert run time error checks in the generated instructions such that the program is terminated if one of the conditions does not

hold. The run time error checks can be added as dynamic error checks using error routines defined at labels, such as $label_{uninitialized\_object}$ which the program can jump to, if such a condition is unmet. Haulund presented an example for dynamic error checking for local blocks in [11]. Pisa and its simulator PendVM is, however, limited and does not support exit codes natively. To fully support dynamic error checking, PendVM could be extended to read from a value from a designated register to supply a more meaningful message for the programmer in the case of a run time exit.

4.10 Implementation

The Roopl++ compiler (Rooplppc) was implemented using techniques and translation schemes presented in this chapter, expanding upon the work of the original Roopl compiler (Rooplc). The compiler serves as a proof-of-concept and simply performs one-to-one translations of Roopl++ code to Pisa code without any optimizations along the way. The compiler is written in Haskell 7.10 and the translated output was tested on the Pendulum simulator, PendVM [5].

As with the Roopl compiler, the Roopl++ compiler is structured around the same six separate compilation phases.

1. **Parsing** consists of constructing an abstract syntax tree from the input program text using parser combinators from the Parsec library in Haskell.
2. **Class Analysis** verifies inheritance cycles, duplicated method names or fields and base classes. In this phase, we also compute the allocation size of each class
3. **Scope Analysis** constructs the virtual and symbol tables and maps every identifier to a unique variable or method.
4. **Type Checking** verifies that the parsed program is well-typed.
5. **Code Generation** translates the abstract syntax tree to blocks of Pisa code in a recursive descent.
6. **Macro Expansion** expands macros left by the code generator for i.e. configuration variables, etc.

Compiled Roopl programs have a size increase by a factor of 10 to 15 in terms of the lines of code. For Roopl++ the size increase is much larger, partially due to the increase of static code included in form of the memory manager using the buddy layout described in this chapter and partially because heap allocations are more costly than stack allocations in terms of lines of code.

The Roopl compiler was implemented in 1403 lines of Haskell and the Roopl++ compiler was extended to 2046 lines of Haskell.

The entire compiler source code as well as example programs and their compiled versions are provided in the appendices and in the supplied ZIP archive. It is also hosted on Github as open source software under the MIT license at `https://github.com/cservenka/ROOPLPPC`.

Building and usage of the compiler is supplied in the README.md file found in the ZIP archive and in appendix B.

4.11 Evaluation

For evaluating the results of the implemented compiler, it was tested against example code provided throughout this thesis. Tests programs utilizing the linked list, doubly-linked list and binary tree data structures and the RTM implementation are found in appendix C.

Program	ROOPL++ LOC	PISA LOC	Number of executed instructions
Linked List	61	1280	18015
Doubly-Linked List	66	1339	21825
Binary Tree	86	2056	6065
RTM Simulation	211	6716	64922

Figure 4.20: Lines of code comparison between target and compiled ROOPL++ programs

The linked list test programs simply instantiates ten cells and links them in their respective lists. The binary tree test program instantiates three nodes and adds them to the tree structure, which afterwards is traversed to determine the sum of the nodes and finally mirroring the tree. The Reversible Turing Machine implementing incrementation of a non-negative n-bit binary number by 1 originally described in [29] has been implemented in ROOPL++ and successfully converts its initial tape value in little endian form of 1101 to 0011 after termination. It should be noted that these test programs require additional stack space during their lengthy computations and as such has been compiled with twice the length between the stack and heap to allow further stack growth.

As discussed, the compiler is considered proof-of-concept and no noteworthy optimizations has been implemented. However, for the sake of giving the reader an idea of the size blowup of a compiled ROOPL++ program, figure 4.20 details this difference. The lines of translated PISA instructions includes the 204 instructions needed for the **malloc** and **malloc1** PISA-equivalent mechanisms. The last row of the table shows how many instructions are execution during simulation using PendVM.

CHAPTER 5

Conclusions

We formally presented a dynamic memory management extension for the reversible object-oriented programing language, ROOPL, in the form of the superset language ROOPL++. The extension expands upon the previously presented static typing system defining well-typedness. The language successfully extends the expressiveness of its predecessor by allowing more flexibility within the domain of reversible object-oriented programming. With ROOPL++ we, as reversible programmers, can now define and model non-trivial dynamic data structures in a reversible setting, such as lists, trees and graphs. We illustrated this by example programs such as a new reversible Turing machine simulator along with implementations for linked lists, doubly-linked lists and binary trees as well as techniques for traversing these. Besides expanding the expressiveness of ROOPL, we have also shown that complex dynamic data structures are not only feasible, but furthermore do not contradict the reversible computing paradigm.

We presented various dynamic memory management layouts and how each would translate into the reversible allocation algorithms. Weighing the advantages and disadvantages of each, the Buddy Memory layout was found to translate into reversible code very naturally with few side effects and addressed a number of disadvantages found in other considered layouts. With dynamically lifetimed objects the allocation and deallocation order is important in terms of a entirely garbage-free computation. In most cases with ROOPL++, we only obtain partially garbage-free computations, as our free lists might not be restored to their original form, without an effective garbage collector design for the memory manager.

Techniques for clean translations of extended parts of the language, such as the memory manager, the new fixed-sized array type and reference counting have been demonstrated and implemented in a proof-of-concept compiler for validation.

With the dynamic memory manager for reversible object-oriented programming languages allowing dynamic object-scopes and multiple references, exemplified by ROOPL++, we have successfully taking an additional step in the direction towards high-level abstractions reversible computations.

5.1 Future Work

Naturally with the discovery of feasibility of non-trivial, reversible data structures with the introduction of ROOPL++, further study of design and implementation of reversible algorithms

working with these data structures are an obvious contender for future research. Data structures such as lists, graphs and trees could potentially provide very interesting future reversible programs.

In terms of the future of reversible object-oriented languages, additional work could be made to extend the fixed-sized array type with a fully dynamic array supporting multiple dimensions. This addition could further help the discovery and research of reversible data structures such as trees and graphs. Such an extension could perhaps be added via a **put** and **take** statement pair, being each others inverse. After a dynamic array has been declared, it could automatically reallocate or upscale its internal space when putting new data outside of its current bounds. In reverse, the space could shrink or reallocate when removing the largest indexed value. The current memory management layout will still suffice for this extension.

Finally, more research could be conducted into reversible heap managers. We provided a simple manager which translated to our problem domain naturally. To obtain completely garbage free computations, a garbage collector could be designed to work with the reversible Buddy Memory memory manager. A reversible garbage collector for non-mutable objects has been designed and shown feasible for the reversible functional language RCFUN in [19]. Additionally, experimentation with implementing the Buddy Memory layout into other reversible languages with dynamic allocation and deallocation such as R-WHILE and R-CORE provides an interesting opportunity [8, 9].

References

[1] Axelsen, H. B. "Clean Translation of an Imperative Reversible Programming Language". In: *Compiler Construction - 20th International Conference, CC 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*. Ed. by J. Knoop. Vol. 6601. Lecture Notes in Computer Science. Springer, 2011, pp. 144–163.

[2] Axelsen, H. B. and Glück, R. "Reversible Representation and Manipulation of Constructor Terms in the Heap". In: *Reversible Computation - 5th International Conference, RC 2013, Victoria, BC, Canada, July 4-5, 2013. Proceedings*. Ed. by G. W. Dueck and D. M. Miller. Vol. 7948. Lecture Notes in Computer Science. Springer, 2013, pp. 96–109.

[3] Axelsen, H. B., Glück, R., and Yokoyama, T. "Reversible Machine Code and Its Abstract Processor Architecture". In: *Computer Science - Theory and Applications, Second International Symposium on Computer Science in Russia, CSR 2007, Ekaterinburg, Russia, September 3-7, 2007, Proceedings*. Ed. by V. Diekert, M. V. Volkov, and A. Voronkov. Vol. 4649. Lecture Notes in Computer Science. Springer, 2007, pp. 56–69.

[4] Baker, H. G. *'Use-Once' Variables and Linear Objects – Storage Management, Reflection and Multi-Threading*. URL: `http://home.pipeline.com/~hbaker1/Use1Var.html` (visited on 01/22/2018).

[5] Clark, C. R. *Improving the Reversible Programming Language R and its Supporting Tools*. URL: `http://www.cise.ufl.edu/research/revcomp/users/cclark/pendvm-fall2001/` (visited on 01/22/2018).

[6] Digiconomist. "Bitcoin Energy Consumption Index". In: *Digiconomist* (2017-12-06). URL: `https://digiconomist.net/bitcoin-energy-consumption` (visited on 01/22/2018).

[7] Frank, M. P. "The R Programming Language and Compiler". MIT Reversible Computing Project Memo #M8. 1997.

[8] Glück, R. and Yokoyama, T. "A Linear-Time Self-Interpreter of a Reversible Imperative Language". In: *Computer Software* 33.3 (2016), pp. 108–128.

[9] Glück, R. and Yokoyama, T. "A Minimalist's Reversible While Language". In: *IEICE Transactions* 100-D.5 (2017), pp. 1026–1034.

[10] Hansen, J. S. K. "Translation of a Reversible Functional Programming Language". Master's Thesis. University of Copenhagen, DIKU, 2014.

[11] Haulund, T. "Design and Implementation of a Reversible Object-Oriented Programming Language". In: *CoRR* abs/1707.07845 (2017).

[12] Haulund, T., Mogensen, T. Æ., and Glück, R. "Implementing Reversible Object-Oriented Language Features on Reversible Machines". In: *Reversible Computation - 9th International Conference, RC 2017, Kolkata, India, July 6-7, 2017, Proceedings*. Ed. by I. Phillips and H. Rahaman. Vol. 10301. Lecture Notes in Computer Science. Springer, 2017, pp. 66–73.

[13] Knuth, D. E. *The Art of Computer Programming: Fundamental Algorithms*. The Art of Computer Programming. Addison-Wesley, 1998, pp. 435–455. ISBN: 0-201-89683-4.

[14] Landauer, R. "Irreversibility and Heat Generation in the Computing Process". In: *IBM Journal of Research and Development* 5.3 (1961), pp. 183–191.

[15] Lee, W. H. and Chang, M. "A study of dynamic memory management in C++ programs". In: *Comput. Lang.* 28.3 (2002), pp. 237–272.

[16] Lutz, C. "Janus: a time-reversible language". Letter to R. Landauer. 1986.

[17] Mitchell, J. C. and Apt, K. *Concepts in Programming Languages*. New York, NY, USA: Cambridge University Press, 2001. ISBN: 0521780985.

[18] Mogensen, T. Æ. "Reference Counting for Reversible Languages". In: *Reversible Computation - 6th International Conference, RC 2014, Kyoto, Japan, July 10-11, 2014. Proceedings*. Ed. by S. Yamashita and S. Minato. Vol. 8507. Lecture Notes in Computer Science. Springer, 2014, pp. 82–94.

[19] Mogensen, T. Æ. "Garbage Collection for Reversible Functional Languages". In: *Reversible Computation - 7th International Conference, RC 2015, Grenoble, France, July 16-17, 2015, Proceedings*. Ed. by J. Krivine and J. Stefani. Vol. 9138. Lecture Notes in Computer Science. Springer, 2015, pp. 79–94.

[20] Mogensen, T. Æ. "Programming Language Design and Implementation (Unpublished)".

[21] *Python Software Foundation: Memory Management*. URL: `https://docs.python.org/2/c-api/memory.html` (visited on 01/22/2018).

[22] Schultz, U. P. and Axelsen, H. B. "Elements of a Reversible Object-Oriented Language - Work-in-Progress Report". In: *Reversible Computation - 8th International Conference, RC 2016, Bologna, Italy, July 7-8, 2016, Proceedings*. Ed. by S. J. Devitt and I. Lanese. Vol. 9720. Lecture Notes in Computer Science. Springer, 2016, pp. 153–159.

[23] Thomsen, M. K., Axelsen, H. B., and Glück, R. "A Reversible Processor Architecture and Its Reversible Logic Design". In: *Reversible Computation - Third International Workshop, RC 2011, Gent, Belgium, July 4-5, 2011. Revised Papers*. Ed. by A. D. Vos and R. Wille. Vol. 7165. Lecture Notes in Computer Science. Springer, 2011, pp. 30–42.

[24] Vance, A. "Inside the Arctic Circle, Where Your Facebook Data Lives". In: *Bloomberg* (2013-12-04). URL: `https://www.bloomberg.com/news/articles/2013-10-04/facebooks-new-data-center-in-sweden-puts-the-heat-on-hardware-makers` (visited on 01/22/2018).

[25] Venners, B. *The Java Virtual Machine*. URL: `http://www.artima.com/insidejvm/ed2/jvmP.html` (visited on 01/22/2018).

[26] Vieri, C. J. "Pendulum: A Reversible Computer Architecture". Master's Thesis. University of California at Berkeley, 1993.

[27] A. D. Vos and R. Wille, eds. *Reversible Computation - Third International Workshop, RC 2011, Gent, Belgium, July 4-5, 2011. Revised Papers*. Vol. 7165. Lecture Notes in Computer Science. Springer, 2012.

[28] Winskel, G. *The Formal Semantics of Programming Languages: An Introduction*. Cambridge, MA, USA: MIT Press, 1993.

[29] Yokoyama, T., Axelsen, H. B., and Glück, R. “Principles of a reversible programming language”. In: *Proceedings of the 5th Conference on Computing Frontiers, 2008, Ischia, Italy, May 5-7, 2008*. Ed. by A. Ramírez, G. Bilardi, and M. Gschwind. ACM, 2008, pp. 43–54.

[30] Yokoyama, T., Axelsen, H. B., and Glück, R. “Towards a Reversible Functional Language”. In: *Reversible Computation - Third International Workshop, RC 2011, Gent, Belgium, July 4-5, 2011. Revised Papers*. Ed. by A. D. Vos and R. Wille. Vol. 7165. Lecture Notes in Computer Science. Springer, 2011, pp. 14–29.

[31] Yokoyama, T. and Glück, R. “A reversible programming language and its invertible self-interpreter”. In: *Proceedings of the 2007 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation, 2007, Nice, France, January 15-16, 2007*. Ed. by G. Ramalingam and E. Visser. ACM, 2007, pp. 144–153.

APPENDIX A

Pisa Translated Buddy Memory

(1)	$malloc1_{top}$:	**BRA**	$malloc1_{bot}$			; Receive jump
(2)		**POP**	r_{ro}			; Pop return offset from the stack
(3)		.....				; Inverse of **(7)**
(4)	$malloc1_{entry}$:	**SWAPBR**	r_{ro}			; Malloc1 entry and exit point
(5)		**NEG**	r_{ro}			; Negate return offset
(6)		**PUSH**	r_{ro}			; Store return offset on stack
(7)		.....				; Code for $r_{fl} \leftarrow addr(freelists[counter])$
(8)		.....				; Code for $r_{block} \leftarrow [\![freelists[counter]]\!]$
(9)		.....				; Code for $r_{e1_o} \leftarrow [\![c_{size} < object_{size}]\!]$
(10)		**XOR**	r_t	r_{e1_o}		; Copy value of $c_{size} < object_{size}$ into r_t
(11)		.....				; Inverse of **(9)**
(12)	o_{test} :	**BEQ**	r_t	r_0	o_{test_f}	; Receive jump
(13)		**XORI**	r_t	1		; Clear r_t
(14)		**ADDI**	r_c	1		; $Counter$++
(15)		**RL**	r_{sc}	1		; Call $double(c_{size})$
(16)		.....				; Inverse of **(7)**
(17)		.....				; Code for pushing temp reg values to stack
(18)		**BRA**	$malloc1_{entry}$			; Call $malloc1())$
(19)		.....				; Inverse of **(17)**
(20)		**RR**	r_{sc}	1		; Inverse of **(15)**
(21)		**SUBI**	r_c	1		; Inverse of **(14)**
(22)		**XORI**	r_t	1		; Set $r_t = 1$
(23)	o_{assert_t} :	**BRA**	o_{assert}			; Jump
(24)	o_{test_f} :	**BRA**	o_{test}			; Receive jump
(25)		.....				; Code for $r_{e1_i} \leftarrow [\![addr(freelists[counter]) \neq 0]\!]$
(26)		**XOR**	r_{t2}	r_{e1_i}		; Copy value of r_{e1_i} into r_{t2}
(27)		.....				; Inverse of **(25)**
(28)	i_{test} :	**BEQ**	r_{t2}	r_0	i_{test_f}	; Receive jump
(29)		**XORI**	r_{t2}	1		; Clear r_{t2}
(30)		**ADD**	r_p	r_{block}		; Copy address of the current block to p
(31)		**SUB**	r_{block}	r_p		; Clear r_{block}
(32)		**EXCH**	r_{tmp}	r_p		; Load address of next block
(33)		**EXCH**	r_{tmp}	r_{fl}		; Set address of next block as new head of free list
(34)		**XOR**	r_{tmp}	r_p		; Clear address of next block
(35)		**XORI**	r_{t2}	1		; Set $r_{t2} = 1$
(36)	i_{assert_t} :	**BRA**	i_{assert}			; Jump
(37)	i_{test_f} :	**BRA**	i_{test}			; Receive jump
(38)		**ADDI**	r_c	1		; $Counter$++
(39)		**RL**	r_{sc}	1		; Call $double(c_{size})$
(40)		.....				; Code for pushing temp reg values to stack
(41)		**BRA**	$malloc1_{entry}$			; Call $malloc1())$
(42)		.....				; Inverse of **(40)**
(43)		**RR**	r_{sc}	1		; Inverse of **(39)**
(44)		**SUBI**	r_c	1		; Inverse of **(38)**

(45)		**XOR**	r_{tmp}	r_p	; Copy current address of p
(46)		**EXCH**	r_{tmp}	r_{fl}	; Store current address of p in current free list
(47)		**ADD**	r_p	r_{cs}	; Split block by setting p to second half of current block
(48)	i_{assert} :	**BNE**	r_{t2}	r_0 i_{assert_t}	; Receive jump
(49)		**EXCH**	r_{tmp}	r_{fl}	; Load address of head of current free list
(50)		**SUB**	r_p	r_{cs}	; Set p to previous block address
(51)		.....			; Code for $r_{e2_{i1}} \leftarrow [\![p - c_{size} \neq addr(freelists[counter])]\!]$
(52)		.....			; Code for $r_{e2_{i2}} \leftarrow [\![addr(freelists[counter]) = 0]\!]$
(53)		.....			; Code for $r_{e2_{i3}} \leftarrow [\![(p - c_{size} \neq addr(freelists[counter])) \vee (addr(freelists[counter]) = 0)]\!]$
(54)		**XOR**	r_{r2}	$r_{e2_{i3}}$	; Copy value of $r_{e2_{i3}}$ into r_{t2}
(55)		.....			; Inverse of **(53)**
(56)		.....			; Inverse of **(52)**
(57)		.....			; Inverse of **(51)**
(58)		**ADD**	r_p	r_{cs}	; Inverse of **(50)**
(59)		**EXCH**	r_{tmp}	r_{fl}	; Inverse of **(49)**
(60)	o_{assert} :	**BNE**	r_t	r_0 o_{assert_t}	; Receive jump
(61)		.....			; Code for $r_{e2_o} \leftarrow [\![c_{size} < object_{size}]\!]$
(62)		**XOR**	r_t	r_{e2_o}	; Copy value of $c_{size} < object_{size}$ into r_t
(63)		.....			; Inverse of **(61)**
(64)	$malloc1_{bot}$:	**BRA**	$malloc1_{top}$		; Jump

APPENDIX B

ROOPLPPC Source Code

README.md

```
# ROOPLPPC

*ROOPLPP* is a compiler translating source code written in **Reversible Object Oriented Programming Language++++ (ROOPL++) to the reversible assembly language Pendulum ISA (PISA).

The compiler is to be considered proof-of-concept in connection with my Master's Thesis on the ROOPL++ language.

## Requirements
ROOPLPPC uses [Stack](https://docs.haskellstack.org/en/stable/README/) to manage all dependencies and requirements.

## Building
Simply invoke
```
stack build
```
which compiles an executable into the `.stack-work` folder

## Usage
To compile a ROOPL++ program simply run
```
stack exec ROOPLPPC input.rplpp
```
which compiles the input program into Pisa and stores the compiled file as `input.pal` in the current directory.

To specify an output file name, simply provide it as an additional argument
```
stack exec ROOPLPP input.rplpp output.pal
```

## Examples
To see usage examples, please refer to `test/` for example programs.

## Running compiled programs
The PendVM simulator executes compiled Pisa code and is hosted on Github [here](https://github.com/TueHaulund/PendVM).
```

AST.hs

```
1  module AST where
2
3  import Text.Show.Pretty
4
5  {-- AST Primitives --}
6  type TypeName = String
7
8  type MethodName = String
9
10 data DataType = IntegerType
11               | ObjectType TypeName
12               | CopyType TypeName
13               |Â ObjectArrayType TypeName
14               | IntegerArrayType
15               | ArrayType
16               | ArrayElementType
17               | NilType
18   deriving (Show)
19
20 -- Types
21 instance Eq DataType where
22   IntegerType == IntegerType = True
23   IntegerArrayType == IntegerArrayType = True
24   NilType == NilType = True
25   NilType == (ObjectType _) = True
26   (ObjectType _) == NilType = True
27   (ObjectType t1) == (ObjectType t2) = t1 == t2
28   (CopyType t1) == (CopyType t2) = t1 == t2
29   (ObjectArrayType t1) == (ObjectArrayType t2) = t1 == t2
30   (CopyType t1) == (ObjectType t2) = t1 == t2
31   (ObjectType t1) == (CopyType t2) = t1 == t2
32   ArrayType == (ObjectArrayType _) = True
33   (ObjectArrayType _) == ArrayType = True
34   ArrayType == IntegerArrayType = True
35   IntegerArrayType == ArrayType = True
36   _ == _ = False
37
38 -- Binary Operators
39 data BinOp = Add
40            | Sub
41            | Xor
42            | Mul
43            | Div
44            | Mod
45            | BitAnd
46            | BitOr
47            | And
48            | Or
49            | Lt
50            | Gt
51            | Eq
52            | Neq
53            | Lte
54            | Gte
55   deriving (Show, Eq, Enum)
56
57 data ModOp = ModAdd
58            | ModSub
59            | ModXor
60   deriving (Show, Eq, Enum)
61
62 {-- Generic AST Definitions --}
63 --Expressions
```

```
64 data GExpr v = Constant Integer
65               | Variable v
66               | ArrayElement (v, GExpr v)
67               | Nil
68               | Binary BinOp (GExpr v) (GExpr v)
69   deriving (Show, Eq)
70
71 --Statements
72 data GStmt m v = Assign v ModOp (GExpr v)
73                 |Â AssignArrElem (v, GExpr v) ModOp (GExpr v)
74                 | Swap (v, Maybe (GExpr v)) (v, Maybe (GExpr v))
75                 | Conditional (GExpr v) [GStmt m v] [GStmt m v] (GExpr v)
76                 | Loop (GExpr v) [GStmt m v] [GStmt m v] (GExpr v)
77                 | ObjectBlock TypeName v [GStmt m v]
78                 | LocalBlock DataType v (GExpr v) [GStmt m v] (GExpr v)
79                 | LocalCall m [(v, Maybe (GExpr v))]
80                 | LocalUncall m [(v, Maybe (GExpr v))]
81                 | ObjectCall (v, Maybe (GExpr v)) MethodName [(v, Maybe (GExpr v))]
82                 | ObjectUncall (v, Maybe (GExpr v)) MethodName [(v, Maybe (GExpr v))]
83                 | ObjectConstruction TypeName (v, Maybe (GExpr v))
84                 | ObjectDestruction TypeName (v, Maybe (GExpr v))
85                 | CopyReference DataType (v, Maybe (GExpr v)) (v, Maybe (GExpr v))
86                 | UnCopyReference DataType (v, Maybe (GExpr v)) (v, Maybe (GExpr v))
87                 | ArrayConstruction (TypeName, GExpr v) v
88                 | ArrayDestruction (TypeName, GExpr v) v
89                 | Skip
90   deriving (Show, Eq)
91
92 --Field/Parameter declarations
93 data GDecl v = GDecl DataType v
94   deriving (Show, Eq)
95
96 --Method: Name, parameters, body
97 data GMDecl m v = GMDecl m [GDecl v] [GStmt m v]
98   deriving (Show, Eq)
99
100 --Class: Name, fields, methods
101 data GCDecl m v = GCDecl TypeName (Maybe TypeName) [GDecl v] [GMDecl m v]
102   deriving (Show, Eq)
103
104 --Program
105 newtype GProg m v = GProg [GCDecl m v]
106   deriving (Show, Eq)
107
108 {-- Specific AST Definitions --}
109 --Plain AST
110 type Identifier = String
111
112 type Expression = GExpr Identifier
113
114 type Statement = GStmt MethodName Identifier
115
116 type VariableDeclaration = GDecl Identifier
117
118 type MethodDeclaration = GMDecl MethodName Identifier
119
120 type ClassDeclaration = GCDecl MethodName Identifier
121
122 type Program = GProg MethodName Identifier
123
124 --Scoped AST
125 type SIdentifier = Integer
126
127 type SExpression = GExpr SIdentifier
128
129 type SStatement = GStmt SIdentifier SIdentifier
```

```haskell

type SVariableDeclaration = GDecl SIdentifier

type SMethodDeclaration = GMDecl SIdentifier SIdentifier

type SProgram = [(TypeName, GMDecl SIdentifier SIdentifier)]

{-- Other Definitions --}
type Offset = Integer

data Symbol = LocalVariable DataType Identifier
            | ClassField DataType Identifier TypeName Offset
            | MethodParameter DataType Identifier
            | Method [DataType] MethodName
  deriving (Show, Eq)

type SymbolTable = [(SIdentifier, Symbol)]

type Scope = [(Identifier, SIdentifier)]

printAST :: (Show t) => t -> String
printAST = ppShow
```

PISA.hs

```
{-# LANGUAGE FlexibleInstances, TypeSynonymInstances #-}

module PISA where

import Data.List (intercalate)
import Control.Arrow

import AST (TypeName, MethodName)

type Label = String

newtype Register = Reg Integer
    deriving (Eq)

{-- Generic PISA Definitions --}

data GInstr i = ADD Register Register
              | ADDI Register i
              | ANDX Register Register Register
              | ANDIX Register Register i
              | NORX Register Register Register
              | NEG Register
              | ORX Register Register Register
              | ORIX Register Register i
              | RL Register i
              | RLV Register Register
              | RR Register i
              | RRV Register Register
              | SLLX Register Register i
              | SLLVX Register Register Register
              | SRAX Register Register i
              | SRAVX Register Register Register
              | SRLX Register Register i
              | SRLVX Register Register Register
              | SUB Register Register
              | XOR Register Register
              | XORI Register i
              | BEQ Register Register Label
              | BGEZ Register Label
              | BGTZ Register Label
              | BLEZ Register Label
              | BLTZ Register Label
              | BNE Register Register Label
              | BRA Label
              | EXCH Register Register
              | SWAPBR Register
              | RBRA Label
              | START
              | FINISH
              | DATA i
              | SUBI Register i --Pseudo
    deriving (Eq)

newtype GProg i = GProg [(Maybe Label, GInstr i)]

{-- Macro PISA Definitions --}

data Macro = Immediate Integer
           | AddressMacro Label
           | SizeMacro TypeName
           | OffsetMacro TypeName MethodName
           | ProgramSize
           | FreeListsSize
```

```
64            | StackOffset
65            | InitialMemoryBlockSize
66            | ReferenceCounterIndex
67            | ArrayElementOffset
68      deriving (Show, Eq)
69
70  type MInstruction = GInstr Macro
71  type MProgram = GProg Macro
72
73  invertInstructions :: [(Maybe Label, MInstruction)] -> [(Maybe Label, MInstruction)]
74  invertInstructions = reverse . map (second invertInstruction . first (fmap (++ "_i")))
75      where invertInstruction (ADD r1 r2) = SUB r1 r2
76            invertInstruction (SUB r1 r2) = ADD r1 r2
77            invertInstruction (ADDI r i) = SUBI r i
78            invertInstruction (SUBI r i) = ADDI r i
79            invertInstruction (RL r i) = RR r i
80            invertInstruction (RLV r1 r2) = RRV r1 r2
81            invertInstruction (RR r i) = RL r i
82            invertInstruction (RRV r1 r2) = RLV r1 r2
83            invertInstruction (BEQ r1 r2 l) = BEQ r1 r2 $ l ++ "_i"
84            invertInstruction (BGEZ r l) = BGEZ r $ l ++ "_i"
85            invertInstruction (BGTZ r l) = BGTZ r $ l ++ "_i"
86            invertInstruction (BLEZ r l) = BLEZ r $ l ++ "_i"
87            invertInstruction (BLTZ r l) = BLTZ r $ l ++ "_i"
88            invertInstruction (BNE r1 r2 l) = BNE r1 r2 $ l ++ "_i"
89            invertInstruction (BRA l) = BRA $ l ++ "_i"
90            invertInstruction (RBRA l) = RBRA $ l ++ "_i"
91            invertInstruction inst = inst
92
93  {-- Output PISA Definitions --}
94
95  type Instruction = GInstr Integer
96  type Program = GProg Integer
97
98  instance Show Register where
99      show (Reg r) = "$" ++ show r
100
101 instance Show Instruction where
102     show (ADD r1 r2) = unwords ["ADD   ", show r1, show r2]
103     show (ADDI r i) = unwords ["ADDI  ", show r, show i]
104     show (ANDX r1 r2 r3) = unwords ["ANDX  ", show r1, show r2, show r3]
105     show (ANDIX r1 r2 i) = unwords ["ANDIX ", show r1, show r2, show i]
106     show (NORX r1 r2 r3) = unwords ["NORX  ", show r1, show r2, show r3]
107     show (NEG r) = unwords ["NEG   ", show r]
108     show (ORX r1 r2 r3) = unwords ["ORX   ", show r1, show r2, show r3]
109     show (ORIX r1 r2 i) = unwords ["ORIX  ", show r1, show r2, show i]
110     show (RL r i) = unwords ["RL    ", show r, show i]
111     show (RLV r1 r2) = unwords ["RLV   ", show r1, show r2]
112     show (RR r i) = unwords ["RR    ", show r, show i]
113     show (RRV r1 r2) = unwords ["RRV   ", show r1, show r2]
114     show (SLLX r1 r2 i) = unwords ["SLLX  ", show r1, show r2, show i]
115     show (SLLVX r1 r2 r3) = unwords ["SLLVX ", show r1, show r2, show r3]
116     show (SRAX r1 r2 i) = unwords ["SRAX  ", show r1, show r2, show i]
117     show (SRAVX r1 r2 r3) = unwords ["SRAVX ", show r1, show r2, show r3]
118     show (SRLX r1 r2 i) = unwords ["SRLX  ", show r1, show r2, show i]
119     show (SRLVX r1 r2 r3) = unwords ["SRLVX ", show r1, show r2, show r3]
120     show (SUB r1 r2) = unwords ["SUB   ", show r1, show r2]
121     show (XOR r1 r2) = unwords ["XOR   ", show r1, show r2]
122     show (XORI r i) = unwords ["XORI  ", show r, show i]
123     show (BEQ r1 r2 l) = unwords ["BEQ   ", show r1, show r2, l]
124     show (BGEZ r l) = unwords ["BGEZ  ", show r, l]
125     show (BGTZ r l) = unwords ["BGTZ  ", show r, l]
126     show (BLEZ r l) = unwords ["BLEZ  ", show r, l]
127     show (BLTZ r l) = unwords ["BLTZ  ", show r, l]
128     show (BNE r1 r2 l) = unwords ["BNE   ", show r1, show r2, l]
129     show (BRA l) = unwords ["BRA   ", l]
```

```
130     show (EXCH r1 r2) = unwords ["EXCH  ", show r1, show r2]
131     show (SWAPBR r) = unwords ["SWAPBR", show r]
132     show (RBRA l) = unwords ["RBRA  ", l]
133     show START = "START "
134     show FINISH = "FINISH"
135     show (DATA i) = unwords ["DATA  ", show i]
136     show (SUBI r i) = unwords ["ADDI  ", show r, show $ -i] --Expand pseudo
137
138 showProgram :: Program -> String
139 showProgram (GProg p) = ";; pendulum pal file\n" ++ intercalate "\n" (map showLine p)
140     where showLine (Nothing, i) = spaces 25 ++ show i
141           showLine (Just l, i) = l ++ ":" ++ spaces (24 - length l) ++ show i
142           spaces :: (Int -> String)
143           spaces n = [1..n] >> " "
144
145 writeProgram :: String -> Program -> IO ()
146 writeProgram output p = writeFile output $ showProgram p
```

Parser.hs

```
1  module Parser (parseString) where
2
3  import Control.Monad.Except
4  import Data.Functor.Identity
5  import Data.Bifunctor
6
7  import Text.Parsec
8  import Text.Parsec.String
9  import Text.Parsec.Expr
10 import Text.Parsec.Language
11 import qualified Text.Parsec.Token as Token
12
13 import Debug.Trace (trace, traceShow)
14
15 import AST
16
17 {-- Language Definition --}
18 keywords :: [String]
19 keywords =
20     ["class",
21      "inherits",
22      "method",
23      "call",
24      "uncall",
25      "construct",
26      "destruct",
27      "skip",
28      "from",
29      "do",
30      "loop",
31      "until",
32      "int",
33      "nil",
34      "if",
35      "then",
36      "else",
37      "fi",
38      "local",
39      "delocal",
40      "new",
41      "delete",
42      "copy",
43      "uncopy"]
44
45 --Operator precedence identical to C
46 operatorTable :: [[(String, BinOp)]]
47 operatorTable =
48     [ [("*", Mul), ("/", Div), ("%", Mod)],
49       [("+", Add), ("-", Sub)],
50       [("<", Lt), ("<=", Lte), (">", Gt), (">=", Gte)],
51       [("=", Eq), ("!=", Neq)],
52       [("&", BitAnd)],
53       [("^", Xor)],
54       [("|", BitOr)],
55       [("&&", And)],
56       [("||", Or)] ]
57
58 languageDef :: Token.LanguageDef st
59 languageDef =
60     emptyDef {
61         Token.commentLine     = "//",
62         Token.nestedComments  = False,
63         Token.identStart      = letter,
```

```
64          Token.identLetter     = alphaNum <|> oneOf "_'",
65          Token.reservedOpNames = concatMap (map fst) operatorTable,
66          Token.reservedNames   = keywords,
67          Token.caseSensitive   = True }
68
69  tokenParser :: Token.TokenParser st
70  tokenParser = Token.makeTokenParser languageDef
71
72  {-- Parser Primitives --}
73  identifier :: Parser String
74  identifier = Token.identifier tokenParser
75
76  arrElemIdentifier :: Parser (String, Expression)
77  arrElemIdentifier = do x <- identifier
78                         y <- brackets expression
79                         return (x, y)
80
81  anyIdentifier :: Parser (String, Maybe Expression)
82  anyIdentifier = do x <- identifier
83                     y <- optionMaybe $ brackets expression
84                     return (x, y)
85
86  reserved :: String -> Parser ()
87  reserved = Token.reserved tokenParser
88
89  reservedOp :: String -> Parser ()
90  reservedOp = Token.reservedOp tokenParser
91
92  integer :: Parser Integer
93  integer = Token.integer tokenParser
94
95  symbol :: String -> Parser String
96  symbol = Token.symbol tokenParser
97
98  parens :: Parser a -> Parser a
99  parens = Token.parens tokenParser
100
101 brackets :: Parser a -> Parser a
102 brackets = Token.brackets tokenParser
103
104 colon :: Parser String
105 colon = Token.colon tokenParser
106
107 commaSep :: Parser a -> Parser [a]
108 commaSep = Token.commaSep tokenParser
109
110 typeName :: Parser TypeName
111 typeName = identifier
112
113 arrayTypeName :: Parser (TypeName, Expression)
114 arrayTypeName = do x <- try typeName <|> string "int"
115                    y <- brackets expression
116                    return (x, y)
117
118 methodName :: Parser MethodName
119 methodName = identifier
120
121 {-- Expression Parsers --}
122 constant :: Parser Expression
123 constant = Constant <$> integer
124
125 variable :: Parser Expression
126 variable = Variable <$> identifier
127
128 arrayElementVariable :: Parser Expression
129 arrayElementVariable = ArrayElement <$> arrElemIdentifier
```

```
130
131 nil :: Parser Expression
132 nil = Nil <$ reserved "nil"
133
134 expression :: Parser Expression
135 expression = buildExpressionParser opTable $ constant <|> try arrayElementVariable <|>
        variable <|> nil
136     where binop (t, op) = Infix (Binary op <$ reservedOp t) AssocLeft
137           opTable = (map . map) binop operatorTable
138
139 {-- Statement Parsers --}
140 modOp :: Parser ModOp
141 modOp = ModAdd <$ symbol "+="
142     <|> ModSub <$ symbol "-="
143     <|> ModXor <$ symbol "^="
144
145 assign :: Parser Statement
146 assign = Assign <$> identifier <*> modOp <*> expression
147
148 assignArrElem :: Parser Statement
149 assignArrElem = AssignArrElem <$> arrElemIdentifier <*> modOp <*> expression
150
151 swap :: Parser Statement
152 swap = Swap <$> anyIdentifier <* symbol "<=>" <*> anyIdentifier
153
154 conditional :: Parser Statement
155 conditional =
156     reserved "if"
157     >> Conditional
158     <$> expression
159     <* reserved "then"
160     <*> block
161     <* reserved "else"
162     <*> block
163     <* reserved "fi"
164     <*> expression
165
166 loop :: Parser Statement
167 loop =
168     reserved "from"
169     >> Loop
170     <$> expression
171     <* reserved "do"
172     <*> block
173     <* reserved "loop"
174     <*> block
175     <* reserved "until"
176     <*> expression
177
178 localCall :: Parser Statement
179 localCall =
180     reserved "call"
181     >> LocalCall
182     <$> methodName
183     <*> parens (commaSep anyIdentifier)
184
185 localUncall :: Parser Statement
186 localUncall =
187     reserved "uncall"
188     >> LocalUncall
189     <$> methodName
190     <*> parens (commaSep anyIdentifier)
191
192 objectCall :: Parser Statement
193 objectCall =
194     reserved "call"
```

```
195     >> ObjectCall
196     <$> anyIdentifier
197     <* colon
198     <* colon
199     <*> methodName
200     <*> parens (commaSep anyIdentifier)
201
202 objectUncall :: Parser Statement
203 objectUncall =
204     reserved "uncall"
205     >> ObjectUncall
206     <$> anyIdentifier
207     <* colon
208     <* colon
209     <*> methodName
210     <*> parens (commaSep anyIdentifier)
211
212 objectConstruction :: Parser Statement
213 objectConstruction =
214     reserved "new"
215     >> ObjectConstruction
216     <$> typeName
217     <*> anyIdentifier
218
219 objectDestruction :: Parser Statement
220 objectDestruction =
221     reserved "delete"
222     >> ObjectDestruction
223     <$> typeName
224     <*> anyIdentifier
225
226 localBlock :: Parser Statement
227 localBlock =
228     reserved "local"
229     >> LocalBlock
230     <$> dataType
231     <*> identifier
232     <* symbol "="
233     <*> expression
234     <*> block
235     <* reserved "delocal"
236     <* dataType
237     <* identifier
238     <* symbol "="
239     <*> expression
240
241 objectBlock :: Parser Statement
242 objectBlock =
243     reserved "construct"
244     >> ObjectBlock
245     <$> typeName
246     <*> identifier
247     <*> block
248     <* reserved "destruct"
249     <* identifier
250
251 skip :: Parser Statement
252 skip = Skip <$ reserved "skip"
253
254 copyReference :: Parser Statement
255 copyReference =
256     reserved "copy"
257     >> CopyReference
258     <$> dataType
259     <*> anyIdentifier
260     <*> anyIdentifier
```

```
261
262 unCopyReference :: Parser Statement
263 unCopyReference =
264     reserved "uncopy"
265     >> UnCopyReference
266     <$> dataType
267     <*> anyIdentifier
268     <*> anyIdentifier
269
270 arrayConstruction :: Parser Statement
271 arrayConstruction =
272     reserved "new"
273     >> ArrayConstruction
274     <$> arrayTypeName
275     <*> identifier
276
277 arrayDestruction :: Parser Statement
278 arrayDestruction =
279     reserved "delete"
280     >> ArrayDestruction
281     <$> arrayTypeName
282     <*> identifier
283
284 statement :: Parser Statement
285 statement = try assign
286         <|> try assignArrElem <|> swap
287         <|> conditional
288         <|> loop
289         <|> try localCall
290         <|> try localUncall
291         <|> objectCall
292         <|> objectUncall
293         <|> localBlock
294         <|> objectBlock
295         <|> try arrayConstruction <|> objectConstruction
296         <|> try arrayDestruction <|> objectDestruction
297         <|> skip
298         <|> copyReference
299         <|> unCopyReference
300
301 block :: Parser [Statement]
302 block = many1 statement
303
304 {-- Top Level Parsers --}
305 dataType :: Parser DataType
306 dataType = try (IntegerArrayType <$ reserved "int" <* symbol "[" <* symbol "]")
307            <|> IntegerType <$ reserved "int"
308        <|> try (ObjectArrayType <$> typeName <* symbol "[" <* symbol "]")
309            <|> ObjectType <$> typeName
310
311
312 variableDeclaration :: Parser VariableDeclaration
313 variableDeclaration = GDecl <$> dataType <*> identifier
314
315 methodDeclaration :: Parser MethodDeclaration
316 methodDeclaration =
317     reserved "method"
318     >> GMDecl
319     <$> methodName
320     <*> parens (commaSep variableDeclaration)
321     <*> block
322
323 classDeclaration :: Parser ClassDeclaration
324 classDeclaration =
325     reserved "class"
326     >> GCDecl
```

```
327     <$> typeName
328     <*> optionMaybe (reserved "inherits" >> typeName)
329     <*> many variableDeclaration
330     <*> many1 methodDeclaration
331
332 program :: Parser Program
333 program = spaces >> GProg <$> many1 classDeclaration <* eof
334
335 parseString :: String -> Except String Program
336 parseString s = ExceptT (Identity $ first show $ parse program "" s)
```

ClassAnalyzer.hs

```haskell
{-# LANGUAGE GeneralizedNewtypeDeriving, FlexibleContexts #-}

module ClassAnalyzer
  ( classAnalysis
  , printCAState
  , CAState(..)
  ) where

import Data.List
import Data.Maybe

import Control.Monad
import Control.Monad.Except
import Control.Monad.State
import Text.Pretty.Simple (pPrint)

import Debug.Trace (trace, traceShow)

import AST

type Size = Integer

-- | The Class Analyzer State consists of a list of classes, sizes, methods
-- | and a main class
data CAState = CAState {
    classes :: [(TypeName, ClassDeclaration)],
    subClasses :: [(TypeName, [TypeName])],
    superClasses :: [(TypeName, [TypeName])],
    classSize :: [(TypeName, Size)],
    classMethods :: [(TypeName, [MethodDeclaration])],
    mainClass :: Maybe TypeName
} deriving (Show, Eq)

-- | The Class Analyzer monad
newtype ClassAnalyzer a = ClassAnalyzer { runCA :: StateT CAState (Except String) a }
    deriving (Functor, Applicative, Monad, MonadState CAState, MonadError String)

-- | Initializes the Class Analyzer State with empty lists and Nothing for the mainClass
initialState :: CAState
initialState = CAState {
    classes = [],
    subClasses = [],
    superClasses = [],
    classSize = [],
    classMethods = [],
    mainClass = Nothing
}

-- | Returns a class from the Class Analyzer State if passed typename matches
getClass :: TypeName -> ClassAnalyzer ClassDeclaration
getClass n = gets classes >>= \cs ->
    case lookup n cs of
        (Just c) -> return c
        Nothing -> throwError $ "ICE: Unknown class " ++ n

-- | Returns the base class inherited from
getBaseClass :: TypeName -> ClassAnalyzer (Maybe TypeName)
getBaseClass n = getClass n >>= getBase
    where getBase (GCDecl _ b _ _) = return b

-- | Throws error if class is defined multiple times
checkDuplicateClasses :: ClassDeclaration -> ClassAnalyzer ()
checkDuplicateClasses (GCDecl n _ _ _) = gets classes >>= \cs ->
```

```
64      when (count cs > 1) (throwError $ "Multiple definitions of class " ++ n)
65      where count = length . filter ((== n) . fst)
66
67  -- | Ensures legal inheritance
68  checkBaseClass :: ClassDeclaration -> ClassAnalyzer ()
69  checkBaseClass (GCDecl _ Nothing _ _) = return ()
70  checkBaseClass (GCDecl n (Just b) _ _) =
71      do when (n == b) (throwError $ "Class " ++ n ++ " cannot inherit from itself")
72         cs <- gets classes
73         when (isNothing $ lookup b cs) (throwError $ "Class " ++ n ++ " cannot inherit from
               unknown class " ++ b)
74
75  -- |Â Checks duplicated field declarations
76  checkDuplicateFields :: ClassDeclaration -> ClassAnalyzer ()
77  checkDuplicateFields (GCDecl n _ fs _) = mapM_ checkField fs
78      where count v = length . filter (\(GDecl _ v') -> v' == v) $ fs
79            checkField (GDecl _ v) = when (count v > 1) (throwError $ "Multiple declarations of
                  field " ++ v ++ " in class " ++ n)
80
81  -- | Checks duplicated method declaration in classes
82  checkDuplicateMethods :: ClassDeclaration -> ClassAnalyzer ()
83  checkDuplicateMethods (GCDecl n _ _ ms) = mapM_ checkMethod ms'
84      where ms' = map (\(GMDecl n' _ _) -> n') ms
85            count m = length . filter (== m) $ ms'
86            checkMethod m = when (count m > 1) (throwError $ "Multiple definitions of method "
                  ++ m ++ " in class " ++ n)
87
88  -- | Checks cyclic inheritance
89  checkCyclicInheritance :: ClassDeclaration -> ClassAnalyzer ()
90  checkCyclicInheritance (GCDecl _ Nothing _ _) = return ()
91  checkCyclicInheritance (GCDecl n b _ _) = checkInheritance b [n]
92      where checkInheritance Nothing _ = return ()
93            checkInheritance (Just b') visited =
94                do when (b' `elem` visited) (throwError $ "Cyclic inheritance involving class "
                       ++ n)
95                   next <- getBaseClass b'
96                   checkInheritance next (b' : visited)
97
98  -- | Sets the main class in the Class Analyzer State
99  setMainClass :: ClassDeclaration -> ClassAnalyzer ()
100 setMainClass (GCDecl n _ _ ms) = when ("main" `elem` ms') (gets mainClass >>= set)
101     where
102         ms' = map (\(GMDecl n' _ _) -> n') ms
103         set (Just m) = throwError $ "Method main already defined in class " ++ m ++ " but
                redefined in class " ++ n
104         set Nothing = modify $ \s -> s {mainClass = Just n}
105
106 -- | Adds classes to the state
107 setClasses :: ClassDeclaration -> ClassAnalyzer ()
108 setClasses c@(GCDecl n _ _ _) = modify $ \s -> s {classes = (n, c) : classes s}
109
110 -- | Add subclasses to the state
111 setSubClasses :: ClassDeclaration -> ClassAnalyzer ()
112 setSubClasses (GCDecl n b _ _) = modify (\s -> s { subClasses = (n, []) : subClasses s }) >>
      addSubClass n b
113
114 -- | Adds a subclass to the list of subclasses
115 addSubClass :: TypeName -> Maybe TypeName -> ClassAnalyzer ()
116 addSubClass _ Nothing = return ()
117 addSubClass n (Just b) = gets subClasses >>= \sc ->
118     case lookup b sc of
119         Nothing -> modify $ \s -> s { subClasses = (b, [n]) : sc }
120         (Just sc') -> modify $ \s -> s { subClasses = (b, n : sc') : delete (b, sc') sc }
121
122 -- | Sets super classes in the state
123 setSuperClasses :: ClassDeclaration -> ClassAnalyzer ()
```

```haskell
setSuperClasses (GCDecl n _ _ _) = gets subClasses >>= \sc ->
    modify $ \s -> s { superClasses = (n, map fst $ filter (\(_, sub) -> n `elem` sub) sc) :
        superClasses s }

-- | Returns the nearest 2^n as size for given class
getClassSize :: ClassDeclaration -> ClassAnalyzer Size
getClassSize (GCDecl _ Nothing fs _) =
    return $ 2 ^ (ceiling :: Double -> Integer) (logBase 2 (2 + genericLength fs))
getClassSize (GCDecl _ (Just b) fs _) =
    getClass b >>= getClassSize >>= \sz ->
        return $ 2 ^ (ceiling :: Double -> Integer) (logBase 2 (fromIntegral $ sz +
            genericLength fs))

-- | Set class size in state
setClassSize :: ClassDeclaration -> ClassAnalyzer ()
setClassSize c@(GCDecl n _ _ _) =
  getClassSize c >>= \sz -> modify $ \s -> s {classSize = (n, sz) : classSize s}

-- | Returns class methods of a passed class
resolveClassMethods :: ClassDeclaration -> ClassAnalyzer [MethodDeclaration]
resolveClassMethods (GCDecl _ Nothing _ ms) = return ms
resolveClassMethods (GCDecl n (Just b) _ ms) = getClass b >>= resolveClassMethods >>= combine
    where checkSignature (GMDecl m ps _, GMDecl m' ps' _) = when (m == m' && ps /= ps') (
        throwError $ "Method " ++ m ++ " in class " ++ n ++ " has invalid method signature")
          compareName (GMDecl m _ _) (GMDecl m' _ _) = m == m'
          combine ms' = mapM_ checkSignature ((,) <$> ms <*> ms') >> return (unionBy
              compareName ms ms')

-- | Adds the methods of a class in the Class Analyzer State
setClassMethods :: ClassDeclaration -> ClassAnalyzer ()
setClassMethods c@(GCDecl n _ _ _) = resolveClassMethods c >>= \cm ->
    modify $ \s -> s { classMethods = (n, cm) : classMethods s }

-- | Class Analyzes a program
caProgram :: Program -> ClassAnalyzer Program
caProgram (GProg p) = do
    mapM_ setClasses p
    mapM_ setSubClasses p
    mapM_ setSuperClasses p
    mapM_ setClassSize p
    mapM_ setClassMethods p
    mapM_ checkDuplicateClasses p
    mapM_ checkDuplicateFields p
    mapM_ checkDuplicateMethods p
    mapM_ checkBaseClass p
    mapM_ checkCyclicInheritance p
    mapM_ setMainClass p
    mc <- gets mainClass
    when (isNothing mc) (throwError "No main method defined")
    return $ GProg rootClasses
    where
        rootClasses = filter noBase p
        noBase (GCDecl _ Nothing _ _) = True
        noBase _ = False

-- | Performs Class Analysis on the program
classAnalysis :: Program -> Except String (Program, CAState)
classAnalysis p = runStateT (runCA $ caProgram p) initialState

-- | Pretty prints the Class Analyzer State
printCAState :: (Program, CAState) -> IO ()
printCAState (_, s) = pPrint s
```

ScopeAnalyzer.hs

```haskell
{-# LANGUAGE GeneralizedNewtypeDeriving, FlexibleContexts #-}

module ScopeAnalyzer
  ( scopeAnalysis
  , printSAState
  , SAState(..)
  ) where

import Data.Maybe
import Data.List
import Data.Typeable

import Control.Monad.State
import Control.Monad.Except

import Debug.Trace (trace, traceShow)

import Text.Pretty.Simple (pPrint)

import AST
import ClassAnalyzer

data SAState =
    SAState {
        symbolIndex :: SIdentifier,
        symbolTable :: SymbolTable,
        scopeStack :: [Scope],
        virtualTables :: [(TypeName, [SIdentifier])],
        caState :: CAState,
        mainMethod :: SIdentifier
    } deriving (Show, Eq)

newtype ScopeAnalyzer a = ScopeAnalyzer { runSA :: StateT SAState (Except String) a }
    deriving (Functor, Applicative, Monad, MonadState SAState, MonadError String)

initialState :: CAState -> SAState
initialState s = SAState { symbolIndex = 0, symbolTable = [], scopeStack = [], virtualTables =
     [], caState = s, mainMethod = 0 }

-- | Add an empty scope to the scope stack
enterScope :: ScopeAnalyzer ()
enterScope = modify $ \s -> s { scopeStack = [] : scopeStack s }

-- | Leaves the current scope by removing it from the scope stack
leaveScope :: ScopeAnalyzer ()
leaveScope = modify $ \s -> s { scopeStack = drop 1 $ scopeStack s }

-- | Returns the top scope at the scope stack
topScope :: ScopeAnalyzer Scope
topScope = gets scopeStack >>= \ss ->
    case ss of
        (s:_) -> return s
        [] -> throwError "ICE: Empty scope stack"

-- | Add a symbol to the current scope
addToScope :: (Identifier, SIdentifier) -> ScopeAnalyzer ()
addToScope b =
    do ts <- topScope
       modify $ \s -> s { scopeStack = (b : ts) : drop 1 (scopeStack s) }

-- | Inserts an identifier and symbol pair into the symbol table and current scope
saInsert :: Symbol -> Identifier -> ScopeAnalyzer SIdentifier
saInsert sym n =
```

```
63      do ts <- topScope
64         when (isJust $ lookup n ts) (throwError $ "Redeclaration of symbol: " ++ n)
65         i <- gets symbolIndex
66         modify $ \s -> s { symbolTable = (i, sym) : symbolTable s, symbolIndex = 1 + i }
67         addToScope (n, i)
68         return i
69
70  -- | Looks up an identifier in the scope
71  saLookup :: Identifier -> ScopeAnalyzer SIdentifier
72  saLookup n = gets scopeStack >>= \ss ->
73      case listToMaybe $ mapMaybe (lookup n) ss of
74          Nothing -> throwError $ "Undeclared symbol: " ++ n
75          Just i -> return i
76
77  -- | Scope Analyses Expressions
78  saExpression :: Expression -> ScopeAnalyzer SExpression
79  saExpression (Constant v) = pure $ Constant v
80  saExpression (Variable n) = Variable <$> saLookup n
81  saExpression Nil = pure Nil
82  saExpression (ArrayElement (n, e)) =
83      do n' <- saLookup n
84         e' <- saExpression e
85         return $ ArrayElement (n', e')
86  saExpression (Binary binop e1 e2) =
87      Binary binop
88      <$> saExpression e1
89      <*> saExpression e2
90
91  -- | Scope Analyses Statements
92  saStatement :: Statement -> ScopeAnalyzer SStatement
93  saStatement s =
94      case s of
95          (Assign n modop e) ->
96              when (elem n $ var e) (throwError "Irreversible variable assignment")
97              >> Assign
98              <$> saLookup n
99              <*> pure modop
100             <*> saExpression e
101
102         (AssignArrElem (n, e1) modop e2) ->
103             when (elem (n, e1) $ varArr e2) (throwError "Irreversible variable assignment")
104             >> AssignArrElem
105             <$> saArrayCell n e1
106             <*> pure modop
107             <*> saExpression e2
108
109         (Swap (n1, e1) (n2, e2)) ->
110             Swap
111             <$> maybeArrayCell n1 e1
112             <*> maybeArrayCell n2 e2
113
114         (Conditional e1 s1 s2 e2) ->
115             Conditional
116             <$> saExpression e1
117             <*> mapM saStatement s1
118             <*> mapM saStatement s2
119             <*> saExpression e2
120
121         (Loop e1 s1 s2 e2) ->
122             Loop
123             <$> saExpression e1
124             <*> mapM saStatement s1
125             <*> mapM saStatement s2
126             <*> saExpression e2
127
128         (LocalBlock t n e1 stmt e2) ->
```

```
129             do e1' <- saExpression e1
130                enterScope
131                n' <- saInsert (LocalVariable t n) n
132                stmt' <- mapM saStatement stmt
133                leaveScope
134                e2' <- saExpression e2
135                return $ LocalBlock t n' e1' stmt' e2'
136
137         (LocalCall m args) ->
138             LocalCall
139             <$> saLookup m
140             <*> localCall m args
141
142         (LocalUncall m args) ->
143             LocalUncall
144             <$> saLookup m
145             <*> localCall m args
146
147         (ObjectCall (o, e) m args) ->
148             do when (args /= nub args || (o, e) `elem` args) (throwError $ "Irreversible
                    invocation of method " ++ m)
149                >> ObjectCall
150                <$> maybeArrayCell o e
151                <*> pure m
152                <*> saArgs args
153
154         (ObjectUncall (o, e) m args) ->
155             when (args /= nub args || (o, e) `elem` args) (throwError $ "Irreversible
                    invocation of method " ++ m)
156             >> ObjectUncall
157             <$> maybeArrayCell o e
158             <*> pure m
159             <*> saArgs args
160
161         (ObjectConstruction tp (n, e)) ->
162             ObjectConstruction
163             <$> pure tp
164             <*> maybeArrayCell n e
165
166         (ObjectDestruction tp (n, e)) ->
167             ObjectDestruction
168             <$> pure tp
169             <*> maybeArrayCell n e
170
171         (ObjectBlock tp n stmt) ->
172             do enterScope
173                n' <- saInsert (LocalVariable (ObjectType tp) n) n
174                stmt' <- mapM saStatement stmt
175                leaveScope
176                return $ ObjectBlock tp n' stmt'
177
178         Skip -> pure Skip
179
180         (CopyReference tp (n, e1) (m, e2)) ->
181             CopyReference
182             <$> pure tp
183             <*> maybeArrayCell n e1
184             <*> maybeArrayCell m e2
185
186         (UnCopyReference tp (n, e1) (m, e2)) ->
187             UnCopyReference
188             <$> pure tp
189             <*> maybeArrayCell n e1
190             <*> maybeArrayCell m e2
191
192         (ArrayConstruction (tp, e) n) ->
```

```
193             do n' <- saLookup n
194                e' <- saExpression e
195                return $ ArrayConstruction (tp, e') n'
196
197         (ArrayDestruction (tp, e) n) ->
198             do n' <- saLookup n
199                e' <- saExpression e
200                return $ ArrayDestruction (tp, e') n'
201
202     where var (Variable n) = [n]
203           var (Binary _ e1 e2) = var e1 ++ var e2
204           var _ = []
205
206           varArr (ArrayElement (n, e)) = [(n, e)]
207           varArr _ = []
208
209           isCF ClassField{} = True
210           isCF _ = False
211
212           rlookup = flip lookup
213
214           localCall :: MethodName -> [(Identifier, Maybe Expression)] -> ScopeAnalyzer [(
                  SIdentifier, Maybe SExpression)]
215           localCall m args =
216             do when (args /= nub args) (throwError $ "Irreversible invocation of method " ++ m
                     )
217                args' <- saArgs args
218                st <- gets symbolTable
219                when (any isCF $ mapMaybe (rlookup st . fst) args') (throwError $ "Irreversible
                       invocation of method " ++ m)
220                return args'
221
222           saArgs :: [(Identifier, Maybe Expression)] -> ScopeAnalyzer [(SIdentifier, Maybe
                  SExpression)]
223           saArgs args =
224             do (ns, es) <- pure $ unzip args
225                ns' <- mapM saLookup ns
226                es' <- mapM (mapM saExpression) es
227                return $ zip ns' es'
228
229           maybeArrayCell :: Identifier -> Maybe Expression -> ScopeAnalyzer (SIdentifier,
                  Maybe SExpression)
230           maybeArrayCell n e =
231             do n' <- saLookup n
232                e' <- mapM saExpression e
233                return (n', e')
234
235           saArrayCell :: Identifier -> Expression -> ScopeAnalyzer (SIdentifier, SExpression)
236           saArrayCell n e =
237             do n' <- saLookup n
238                e' <- saExpression e
239                return (n', e')
240
241 -- | Set the main method in the Scope Analyzer state
242 setMainMethod :: SIdentifier -> ScopeAnalyzer ()
243 setMainMethod i = modify $ \s -> s { mainMethod = i }
244
245 -- | Scope Analyses Methods
246 saMethod :: (TypeName, MethodDeclaration) -> ScopeAnalyzer (TypeName, SMethodDeclaration)
247 saMethod (t, GMDecl m ps body) =
248     do m' <- saLookup m
249        when (m == "main") (setMainMethod m')
250        enterScope
251        ps' <- mapM insertMethodParameter ps
252        body' <- mapM saStatement body
253        leaveScope
```

```
254         return (t, GMDecl m' ps' body')
255     where insertMethodParameter (GDecl tp n) = GDecl tp <$> saInsert (MethodParameter tp n) n
256
257 -- | Returns subclasses for a given type name
258 getSubClasses :: TypeName -> ScopeAnalyzer [ClassDeclaration]
259 getSubClasses n =
260     do cs <- gets $ classes . caState
261        sc <- gets $ subClasses . caState
262        case lookup n sc of
263            Nothing -> throwError $ "ICE: Unknown class " ++ n
264            (Just sc') -> return $ mapMaybe (rlookup cs) sc'
265     where rlookup = flip lookup
266
267 -- | Returns method name at given index
268 getMethodName :: SIdentifier -> ScopeAnalyzer (SIdentifier, MethodName)
269 getMethodName i = gets symbolTable >>= \st ->
270     case lookup i st of
271         (Just (Method _ m)) -> return (i, m)
272         _ -> throwError $ "ICE: Invalid method index " ++ show i
273
274 -- | Prefixes the virtual table
275 prefixVtable :: [(SIdentifier, MethodName)] -> (SIdentifier, MethodName) -> [(SIdentifier,
        MethodName)]
276 prefixVtable [] m' = [m']
277 prefixVtable (m:ms) m' = if comp m m' then m':ms else m : prefixVtable ms m'
278     where comp (_, n) (_, n') = n == n'
279
280 -- | Scope Analyses a passed class
281 saClass :: Offset -> [SIdentifier] -> ClassDeclaration -> ScopeAnalyzer [(TypeName,
        SMethodDeclaration)]
282 saClass offset pids (GCDecl c _ fs ms) =
283     do enterScope
284        mapM_ insertClassField $ zip [offset..] fs
285        m1 <- mapM getMethodName pids
286        m2 <- mapM insertMethod ms
287        let m3 = map fst $ foldl prefixVtable m1 m2
288            offset' = genericLength fs + offset
289        modify $ \s -> s { virtualTables = (c, m3) : virtualTables s }
290        sc <- getSubClasses c
291        ms' <- concat <$> mapM (saClass offset' m3) sc
292        ms'' <- mapM saMethod $ zip (repeat c) ms
293        leaveScope
294        return $ ms' ++ ms''
295     where insertClassField (o, GDecl tp n) = saInsert (ClassField tp n c o) n
296           insertMethod (GMDecl n ps _) = saInsert (Method (map getType ps) n) n >>=
                  getMethodName
297           getType (GDecl tp _) = tp
298
299 -- | Analyses Programs
300 saProgram :: Program -> ScopeAnalyzer SProgram
301 saProgram (GProg cs) = concat <$> mapM (saClass 2 []) cs
302
303 -- | Performs scope analysis on the entire program
304 scopeAnalysis :: (Program, CAState) -> Except String (SProgram, SAState)
305 scopeAnalysis (p, s) = runStateT (runSA $ saProgram p) $ initialState s
306
307 -- | Pretty prints the current Scope Analysis State Monad
308 printSAState :: (Show a, MonadIO m) => (t, a) -> m ()
309 printSAState (_, s) = pPrint s
```

TypeChecker.hs

```
1  {-# LANGUAGE GeneralizedNewtypeDeriving #-}
2
3  module TypeChecker (typeCheck) where
4
5  import Data.List
6  import Data.Maybe
7
8  import Control.Monad.Reader
9  import Control.Monad.Except
10 import Control.Exception
11
12 import Debug.Trace (trace, traceShow)
13
14 import AST
15 import ClassAnalyzer
16 import ScopeAnalyzer
17
18 newtype TypeChecker a = TypeChecker { runTC :: ReaderT SAState (Except String) a }
19     deriving (Functor, Applicative, Monad, MonadReader SAState, MonadError String)
20
21 getType :: SIdentifier -> TypeChecker DataType
22 getType i = asks symbolTable >>= \st ->
23     case lookup i st of
24         (Just (LocalVariable t _)) -> return t
25         (Just (ClassField t _ _ _)) -> return t
26         (Just (MethodParameter t _)) -> return t
27         _ -> throwError $ "ICE: Invalid index " ++ show i
28
29 getParameterTypes :: SIdentifier -> TypeChecker [DataType]
30 getParameterTypes i = asks symbolTable >>= \st ->
31     case lookup i st of
32         (Just (Method ps _)) -> return ps
33         _ -> throwError $ "ICE: Invalid index " ++ show i
34
35 expectType :: DataType -> DataType -> TypeChecker ()
36 expectType t1 t2 = unless (t1 == t2) (throwError $ "Expected type: " ++ show t1 ++ "\nActual
       type: " ++ show t2)
37
38 getClassMethods :: TypeName -> TypeChecker [MethodDeclaration]
39 getClassMethods n = asks (classMethods . caState) >>= \cm ->
40     case lookup n cm of
41         Nothing -> throwError $ "ICE: Unknown class " ++ n
42         (Just ms) -> return ms
43
44 getDynamicParameterTypes :: TypeName -> MethodName -> TypeChecker [DataType]
45 getDynamicParameterTypes n m = getClassMethods n >>= \ms ->
46     case find (\(GMDecl m' _ _) -> m == m') ms of
47         Nothing -> throwError $ "Class " ++ n ++ " does not support method " ++ m
48         (Just (GMDecl _ ps _)) -> return $ map (\(GDecl tp _) -> tp) ps
49
50 getArrayType :: DataType -> DataType
51 getArrayType tp = case tp of
52                     IntegerArrayType -> IntegerType
53                     ObjectArrayType t -> ObjectType t
54
55 checkCall :: [(SIdentifier, Maybe SExpression)] -> [DataType] -> TypeChecker ()
56 checkCall args ps =
57     when (la /= lp) (throwError err)
58     >> mapM (mapM tcExpression . snd) args
59     >> mapM (getType . fst) args
60     >>= \as -> mapM_ checkArgument (zip as ps)
61     where la = length args
62           lp = length ps
```

```
63            err = "Passed " ++ show la ++ " argument(s) to method expecting " ++ show lp ++ "
                  argument(s)"
64
65  checkArgument :: (DataType, DataType) -> TypeChecker ()
66  checkArgument (ObjectType ca, ObjectType cp) = asks (superClasses . caState) >>= \sc ->
67      unless (ca == cp || maybe False (elem cp) (lookup ca sc)) (throwError $ "Class " ++ ca ++
            " not a subtype of class " ++ cp)
68  checkArgument (ObjectType ca, ObjectArrayType cp) = asks (superClasses . caState) >>= \sc ->
69      unless (ca == cp || maybe False (elem cp) (lookup ca sc)) (throwError $ "Class " ++ ca ++
            " not a subtype of class " ++ cp)
70  checkArgument (ObjectArrayType ca, ObjectType cp) = asks (superClasses . caState) >>= \sc ->
71      unless (ca == cp || maybe False (elem cp) (lookup ca sc)) (throwError $ "Class " ++ ca ++
            " not a subtype of class " ++ cp)
72  checkArgument (IntegerArrayType, tp) = expectType (getArrayType IntegerArrayType) tp
73  checkArgument (ta, IntegerArrayType) = expectType (getArrayType IntegerArrayType) ta
74  checkArgument (ta, tp) = expectType tp ta
75
76  tcExpression :: SExpression -> TypeChecker DataType
77  tcExpression (Constant _) = pure IntegerType
78  tcExpression (Variable n) = getType n
79  tcExpression Nil = pure NilType
80  tcExpression (ArrayElement (n, e)) =
81      do t <- getType n
82         expectType ArrayType t
83         e' <- tcExpression e
84         expectType IntegerType e'
85         return $ getArrayType t
86  tcExpression (Binary binop e1 e2)
87      | binop == Eq || binop == Neq =
88          do t1 <- tcExpression e1
89             t2 <- tcExpression e2
90             expectType t1 t2
91             pure IntegerType
92      | otherwise =
93          do t1 <- tcExpression e1
94             t2 <- tcExpression e2
95             expectType t1 IntegerType
96             expectType t2 IntegerType
97             pure IntegerType
98
99  tcStatement :: SStatement -> TypeChecker ()
100 tcStatement s =
101     case s of
102         (Assign n _ e) ->
103             getType n
104             >>= expectType IntegerType
105             >> tcExpression e
106             >>= expectType IntegerType
107
108         (AssignArrElem (n, e1) _ e2) ->
109             getType n
110             >>= expectType IntegerArrayType
111             >> tcExpression e1
112             >>= expectType IntegerType
113             >> tcExpression e2
114             >>= expectType IntegerType
115
116         (Swap (n1, e1) (n2, e2)) ->
117             do t1 <- getType n1
118                t2 <- getType n2
119                if isNothing e1 /= isNothing e2
120                  then catchError (checkArgument (t1, t2)) (\_ -> checkArgument (t2, t1))
121                  else expectType (if isNothing e1 then t1 else getArrayType t1) (if isNothing
                         e2 then t2 else getArrayType t2)
122
123         (Conditional e1 s1 s2 e2) ->
```

```
124             tcExpression e1
125             >>= expectType IntegerType
126             >> mapM_ tcStatement s1
127             >> mapM_ tcStatement s2
128             >> tcExpression e2
129             >>= expectType IntegerType
130
131         (Loop e1 s1 s2 e2) ->
132             tcExpression e1
133             >>= expectType IntegerType
134             >> mapM_ tcStatement s1
135             >> mapM_ tcStatement s2
136             >> tcExpression e2
137             >>= expectType IntegerType
138
139         (ObjectBlock _ _ stmt) ->
140             mapM_ tcStatement stmt
141
142         (LocalBlock t n e1 stmt e2) ->
143             getType n
144             >> tcExpression e1
145             >>= expectType (if t == IntegerType then IntegerType else NilType)
146             >> mapM_ tcStatement stmt
147             >> tcExpression e2
148             >>= expectType (if t == IntegerType then IntegerType else NilType)
149
150         (LocalCall m args) ->
151             getParameterTypes m
152             >>= checkCall args
153
154         (LocalUncall m args) ->
155             getParameterTypes m
156             >>= checkCall args
157
158         (ObjectCall (o, e) m args) ->
159             do t <- getType o
160                e' <- mapM tcExpression e
161                case t of
162                    (ObjectType tn) -> getDynamicParameterTypes tn m >>= checkCall args
163                    (ObjectArrayType tn) ->
164                     case e' of
165                         Nothing -> throwError $ "Non-object type " ++ show t ++ " does not
                                support method invocation"
166                         _ -> getDynamicParameterTypes tn m >>= checkCall args
167                    _ -> throwError $ "Non-object type " ++ show t ++ " does not support method
                         invocation"
168
169         (ObjectUncall (o, e) m args) ->
170             do t <- getType o
171                e' <- mapM tcExpression e
172                case t of
173                    (ObjectType tn) -> getDynamicParameterTypes tn m >>= checkCall args
174                    (ObjectArrayType tn) ->
175                     case e' of
176                         Nothing -> throwError $ "Non-object type " ++ show t ++ " does not
                                support method invocation"
177                         _ -> getDynamicParameterTypes tn m >>= checkCall args
178                    _ -> throwError $ "Non-object type " ++ show t ++ " does not support method
                         invocation"
179
180         Skip -> pure ()
181
182         (ObjectConstruction tp (n, e)) ->
183             do t <- getType n
184                e' <- mapM tcExpression e
185                case e' of
```

```
186                 Nothing -> expectType t (ObjectType tp)
187                 _       -> checkArgument (ObjectType tp, t)
188
189         (ObjectDestruction tp (n, e)) ->
190             do t <- getType n
191                _ <- mapM tcExpression e
192                case t of
193                 (ObjectType _) -> expectType t (ObjectType tp)
194                 (ObjectArrayType _) -> checkArgument (ObjectType tp, t)
195                 _ -> throwError $ "Expected type: " ++ show (ObjectType tp) ++ " Actual type:
                        " ++ show t
196
197         -- Allow copying with a copy type
198         CopyReference _ (n, e1) (m, e2) ->
199             do t1 <- getType n
200                t2 <- getType m
201                e1' <- mapM tcExpression e1
202                e2' <- mapM tcExpression e2
203                when (t1 == IntegerType || t2 == IntegerType) (throwError "Integer types does
                       not support reference copying")
204                if isNothing e1 /= isNothing e2
205                  then catchError (checkArgument (t1, t2)) (\_ -> checkArgument (t2, t1))
206                  else expectType (if isNothing e1 then t1 else getArrayType t1) (if isNothing
                         e2 then t2 else getArrayType t2)
207
208         -- Allow uncopying with two identical copies
209         UnCopyReference _ (n, e1) (m, e2) ->
210             do t1 <- getType n
211                t2 <- getType m
212                e1' <- mapM tcExpression e1
213                e2' <- mapM tcExpression e2
214                when (t1 == IntegerType || t2 == IntegerType) (throwError "Integer types does
                       not support reference copying")
215                if isNothing e1 /= isNothing e2
216                  then catchError (checkArgument (t1, t2)) (\_ -> checkArgument (t2, t1))
217                  else expectType (if isNothing e1 then t1 else getArrayType t1) (if isNothing
                         e2 then t2 else getArrayType t2)
218
219
220         (ArrayConstruction (tp, e) n) ->
221             do t <- getType n
222                _ <- tcExpression e
223                case tp of
224                  "int" -> expectType t IntegerArrayType
225                  _     -> expectType t (ObjectArrayType tp)
226
227         (ArrayDestruction (tp, e) n) ->
228             do t <- getType n
229                _ <- tcExpression e
230                case tp of
231                  "int" -> expectType t IntegerArrayType
232                  _     -> checkArgument (ObjectArrayType tp, t)
233
234 getMethodName :: SIdentifier -> TypeChecker Identifier
235 getMethodName i = asks symbolTable >>= \st ->
236     case lookup i st of
237         (Just (Method _ n)) -> return n
238         _ -> throwError $ "ICE: Invalid index " ++ show i
239
240 tcMethod :: (TypeName, SMethodDeclaration) -> TypeChecker ()
241 tcMethod (_, GMDecl _ [] body) = mapM_ tcStatement body
242 tcMethod (_, GMDecl i (_:_) body) = getMethodName i >>= \n ->
243     when (n == "main") (throwError "Method main has invalid signature")
244     >> mapM_ tcStatement body
245
246 tcProgram :: SProgram -> TypeChecker (SProgram, SAState)
```

```
247 tcProgram p = (,) p <$> (mapM_ tcMethod p >> ask)
248
249 typeCheck :: (SProgram, SAState) -> Except String (SProgram, SAState)
250 typeCheck (p, s) = runReaderT (runTC $ tcProgram p) s
```

CodeGenerator.hs

```haskell
{-# LANGUAGE GeneralizedNewtypeDeriving #-}
{-# LANGUAGE ScopedTypeVariables        #-}

module CodeGenerator(
    generatePISA,
    showPISAProgram
) where

import Data.List

import Control.Arrow
import Control.Monad.Except
import Control.Monad.State

import Debug.Trace (trace, traceShow)

import Text.Pretty.Simple (pPrint)

import AST
import ClassAnalyzer
import PISA
import ScopeAnalyzer

{-# ANN module "HLint: ignore Reduce duplication" #-}

data CGState =
    CGState {
        labelIndex :: SIdentifier,
        registerIndex :: Integer,
        labelTable :: [(SIdentifier, Label)],
        registerStack :: [(SIdentifier, Register)],
        saState :: SAState
    } deriving (Show, Eq)

newtype CodeGenerator a = CodeGenerator { runCG :: StateT CGState (Except String) a }
    deriving (Functor, Applicative, Monad, MonadState CGState, MonadError String)


initialState :: SAState -> CGState
initialState s = CGState { labelIndex = 0, registerIndex = 6, labelTable = [], registerStack =
     [], saState = s }

-- | Register containing 0
registerZero :: Register
registerZero = Reg 0

-- | Register containing Stack pointer
registerSP :: Register
registerSP = Reg 1

-- | Register R0
registerR0 :: Register
registerR0 = Reg 2

-- | Register holding 'this'
registerThis :: Register
registerThis = Reg 3

-- | Register containing Free list pointers
registerFLPs :: Register
registerFLPs = Reg 4

-- | Register containing Heap pointer
```

```
63 registerHP :: Register
64 registerHP = Reg 5
65
66 -- | Pushes a new register to the register stack
67 pushRegister :: SIdentifier -> CodeGenerator Register
68 pushRegister i = do ri <- gets registerIndex
69                     modify $ \s -> s { registerIndex = 1 + ri, registerStack = (i, Reg ri) :
                           registerStack s }
70                     return $ Reg ri
71
72 -- | Pop a register from the register stack
73 popRegister :: CodeGenerator ()
74 popRegister = modify $ \s -> s { registerIndex = (-1) + registerIndex s, registerStack = drop
       1 $ registerStack s }
75
76 -- | Reserve a tmp register
77 tempRegister :: CodeGenerator Register
78 tempRegister =
79     do ri <- gets registerIndex
80        modify $ \s -> s { registerIndex = 1 + ri }
81        return $ Reg ri
82
83 -- | Clear reverved tmp register
84 popTempRegister :: CodeGenerator ()
85 popTempRegister = modify $ \s -> s { registerIndex = (-1) + registerIndex s }
86
87 -- | Lookup register of given identifier
88 lookupRegister :: SIdentifier -> CodeGenerator Register
89 lookupRegister i = gets registerStack >>= \rs ->
90     case lookup i rs of
91         Nothing -> throwError $ "ICE: No register reserved for index " ++ show i
92         (Just r) -> return r
93
94 -- | Returns the method name of a valid method identifier
95 getMethodName :: SIdentifier -> CodeGenerator MethodName
96 getMethodName i = gets (symbolTable . saState) >>= \st ->
97     case lookup i st of
98         (Just (Method _ n)) -> return n
99         _ -> throwError $ "ICE: Invalid method index " ++ show i
100
101 -- | Inserts a unique method label in the label table for a given method identifier
102 insertMethodLabel :: SIdentifier -> CodeGenerator ()
103 insertMethodLabel m =
104     do n <- getMethodName m
105        i <- gets labelIndex
106        modify $ \s -> s { labelIndex = 1 + i, labelTable = (m, "l_" ++ n ++ "_" ++ show i) :
               labelTable s }
107
108 -- | Returns the Method label for a method identifier
109 getMethodLabel :: SIdentifier -> CodeGenerator Label
110 getMethodLabel m = gets labelTable >>= \lt ->
111     case lookup m lt of
112         (Just l) -> return l
113         Nothing -> insertMethodLabel m >> getMethodLabel m
114
115 -- | Returns a unique label by appending the label index to a passed label type
116 getUniqueLabel :: Label -> CodeGenerator Label
117 getUniqueLabel l =
118     do i <- gets labelIndex
119        modify $ \s -> s { labelIndex = 1 + i }
120        return $ l ++ "_" ++ show i
121
122 -- | Returns the address to the variable of a given identifier
123 loadVariableAddress :: SIdentifier -> CodeGenerator (Register, [(Maybe Label, MInstruction)],
       CodeGenerator ())
124 loadVariableAddress n = gets (symbolTable . saState) >>= \st ->
```

```
125     case lookup n st of
126         (Just (ClassField _ _ _ o)) -> tempRegister >>= \r -> return (r, [(Nothing, ADD r
                registerThis), (Nothing, ADDI r $ Immediate o)], popTempRegister)
127         (Just (LocalVariable _ _)) -> lookupRegister n >>= \r -> return (r, [], return ())
128         (Just (MethodParameter _ _)) -> lookupRegister n >>= \r -> return (r, [], return ())
129         _ -> throwError $ "ICE: Invalid variable index " ++ show n
130
131 -- | Returns the value of a variable of given identifier
132 loadVariableValue :: SIdentifier -> CodeGenerator (Register, [(Maybe Label, MInstruction)],
        CodeGenerator ())
133 loadVariableValue n =
134     do (ra, la, ua) <- loadVariableAddress n
135        rv <- tempRegister
136        return (rv, la ++ [(Nothing, EXCH rv ra)] ++ invertInstructions la, popTempRegister >>
               ua)
137
138 -- | Returns address an array element
139 loadArrayElementVariableAddress :: SIdentifier -> SExpression -> CodeGenerator (Register, [(
        Maybe Label, MInstruction)], CodeGenerator ())
140 loadArrayElementVariableAddress n e =
141     do (ra, la, ua) <- loadVariableAddress n
142        (re, le, ue) <- cgExpression e
143        rv <- tempRegister
144        rt <- tempRegister
145        return (rv, la ++ le ++ [(Nothing, EXCH rt ra), (Nothing, XOR rv rt), (Nothing, EXCH rt
               ra), (Nothing, ADDI rv ArrayElementOffset), (Nothing, ADD rv re)] ++
              invertInstructions (la ++ le), popTempRegister >> popTempRegister >> ue >> ua)
146
147 -- | Returns the value of an array element
148 loadArrayElementVariableValue :: SIdentifier -> SExpression -> CodeGenerator (Register, [(
        Maybe Label, MInstruction)], CodeGenerator ())
149 loadArrayElementVariableValue n e =
150     do (ra, la, ua) <- loadArrayElementVariableAddress n e
151        rv <- tempRegister
152        return (rv, la ++ [(Nothing, EXCH rv ra)] ++ invertInstructions la , popTempRegister >>
               ua)
153
154 --Â | Returns pointer to free list at given index
155 loadFreeListAddress :: Register -> CodeGenerator (Register, [(Maybe Label, MInstruction)],
        CodeGenerator ())
156 loadFreeListAddress index = tempRegister >>= \rt -> return (rt, [(Nothing, XOR rt registerFLPs
        ), (Nothing, ADD rt index)], popTempRegister)
157
158 -- |Â Returns a copy of the pointer to the head of the free list at the given register
159 loadHeadAtFreeList :: Register -> CodeGenerator (Register, [(Maybe Label, MInstruction)],
        CodeGenerator ())
160 loadHeadAtFreeList rFreeList =
161     do rv <- tempRegister
162        rt <- tempRegister
163        let copyAddress = [(Nothing, EXCH rt rFreeList),
164                           (Nothing, XOR rv rt),
165                           (Nothing, EXCH rt rFreeList)]
166        return (rv, copyAddress, popTempRegister >> popTempRegister)
167
168 -- | Code generation for binary operators
169 cgBinOp :: BinOp -> Register -> Register -> CodeGenerator (Register, [(Maybe Label,
        MInstruction)], CodeGenerator ())
170 cgBinOp Add r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, XOR rt r1), (Nothing, ADD
        rt r2)], popTempRegister)
171 cgBinOp Sub r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, XOR rt r1), (Nothing, SUB
        rt r2)], popTempRegister)
172 cgBinOp Xor r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, XOR rt r1), (Nothing, XOR
        rt r2)], popTempRegister)
173 cgBinOp BitAnd r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, ANDX rt r1 r2)],
        popTempRegister)
```

```
174 cgBinOp BitOr r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, ORX rt r1 r2)],
      popTempRegister)
175 cgBinOp Lt r1 r2 =
176     do rt <- tempRegister
177        rc <- tempRegister
178        l_top <- getUniqueLabel "cmp_top"
179        l_bot <- getUniqueLabel "cmp_bot"
180        let cmp = [(Nothing, XOR rt r1),
181                   (Nothing, SUB rt r2),
182                   (Just l_top, BGEZ rt l_bot),
183                   (Nothing, XORI rc $ Immediate 1),
184                   (Just l_bot, BGEZ rt l_top)]
185        return (rc, cmp, popTempRegister >> popTempRegister)
186 cgBinOp Gt r1 r2 =
187     do rt <- tempRegister
188        rc <- tempRegister
189        l_top <- getUniqueLabel "cmp_top"
190        l_bot <- getUniqueLabel "cmp_bot"
191        let cmp = [(Nothing, XOR rt r1),
192                   (Nothing, SUB rt r2),
193                   (Just l_top, BLEZ rt l_bot),
194                   (Nothing, XORI rc $ Immediate 1),
195                   (Just l_bot, BLEZ rt l_top)]
196        return (rc, cmp, popTempRegister >> popTempRegister)
197 cgBinOp Eq r1 r2 =
198     do rt <- tempRegister
199        l_top <- getUniqueLabel "cmp_top"
200        l_bot <- getUniqueLabel "cmp_bot"
201        let cmp = [(Just l_top, BNE r1 r2 l_bot),
202                   (Nothing, XORI rt $ Immediate 1),
203                   (Just l_bot, BNE r1 r2 l_top)]
204        return (rt, cmp, popTempRegister)
205 cgBinOp Neq r1 r2 =
206     do rt <- tempRegister
207        l_top <- getUniqueLabel "cmp_top"
208        l_bot <- getUniqueLabel "cmp_bot"
209        let cmp = [(Just l_top, BEQ r1 r2 l_bot),
210                   (Nothing, XORI rt $ Immediate 1),
211                   (Just l_bot, BEQ r1 r2 l_top)]
212        return (rt, cmp, popTempRegister)
213 cgBinOp Lte r1 r2 =
214     do rt <- tempRegister
215        rc <- tempRegister
216        l_top <- getUniqueLabel "cmp_top"
217        l_bot <- getUniqueLabel "cmp_bot"
218        let cmp = [(Nothing, XOR rt r1),
219                   (Nothing, SUB rt r2),
220                   (Just l_top, BGTZ rt l_bot),
221                   (Nothing, XORI rc $ Immediate 1),
222                   (Just l_bot, BGTZ rt l_top)]
223        return (rc, cmp, popTempRegister >> popTempRegister)
224 cgBinOp Gte r1 r2 =
225     do rt <- tempRegister
226        rc <- tempRegister
227        l_top <- getUniqueLabel "cmp_top"
228        l_bot <- getUniqueLabel "cmp_bot"
229        let cmp = [(Nothing, XOR rt r1),
230                   (Nothing, SUB rt r2),
231                   (Just l_top, BLTZ rt l_bot),
232                   (Nothing, XORI rc $ Immediate 1),
233                   (Just l_bot, BLTZ rt l_top)]
234        return (rc, cmp, popTempRegister >> popTempRegister)
235 cgBinOp _ _ _ = throwError "ICE: Binary operator not implemented"
236
237 -- | Code generation for expressions
```

```
238 cgExpression :: SExpression -> CodeGenerator (Register, [(Maybe Label, MInstruction)],
        CodeGenerator ())
239 cgExpression (Constant 0) = return (registerZero, [], return ())
240 cgExpression (Constant n) = tempRegister >>= \rt -> return (rt, [(Nothing, XORI rt $ Immediate
         n)], popTempRegister)
241 cgExpression (Variable i) = loadVariableValue i
242 cgExpression (ArrayElement (n, e)) = loadArrayElementVariableValue n e
243 cgExpression Nil = return (registerZero, [], return ())
244 cgExpression (Binary op e1 e2) =
245     do (r1, l1, u1) <- cgExpression e1
246        (r2, l2, u2) <- cgExpression e2
247        (ro, lo, uo) <- cgBinOp op r1 r2
248        return (ro, l1 ++ l2 ++ lo, uo >> u2 >> u1)
249
250 -- | Code generation for binary expressions
251 cgBinaryExpression :: SExpression -> CodeGenerator (Register, [(Maybe Label, MInstruction)],
        CodeGenerator ())
252 cgBinaryExpression e =
253     do (re, le, ue) <- cgExpression e
254        rt <- tempRegister
255        l_top <- getUniqueLabel "f_top"
256        l_bot <- getUniqueLabel "f_bot"
257        let flatten = [(Just l_top, BEQ re registerZero l_bot),
258                       (Nothing, XORI rt $ Immediate 1),
259                       (Just l_bot, BEQ re registerZero l_top)]
260        return (rt, le ++ flatten, popTempRegister >> ue)
261
262 -- | Code generation for assignments
263 cgAssign :: SIdentifier -> ModOp -> SExpression -> CodeGenerator [(Maybe Label, MInstruction)]
264 cgAssign n modop e =
265     do (rt, lt, ut) <- loadVariableValue n
266        (re, le, ue) <- cgExpression e
267        ue >> ut
268        return $ lt ++ le ++ [(Nothing, cgModOp modop rt re)] ++ invertInstructions (lt ++ le)
269     where cgModOp ModAdd = ADD
270           cgModOp ModSub = SUB
271           cgModOp ModXor = XOR
272
273 -- | Code generation for assignments
274 cgAssignArrElem :: (SIdentifier, SExpression) -> ModOp -> SExpression -> CodeGenerator [(Maybe
         Label, MInstruction)]
275 cgAssignArrElem (n, e1) modop e2 =
276     do (rt, lt, ut) <- loadArrayElementVariableValue n e1
277        (re, le, ue) <- cgExpression e2
278        l <- getUniqueLabel "assArrElem"
279        ue >> ut
280        return $ lt ++ le ++ [(Just l, cgModOp modop rt re)] ++ invertInstructions (lt ++ le)
281     where cgModOp ModAdd = ADD
282           cgModOp ModSub = SUB
283           cgModOp ModXor = XOR
284
285 -- | Ensures correct loads for swapping
286 loadForSwap :: (SIdentifier, Maybe SExpression) -> CodeGenerator (Register, [(Maybe Label,
        MInstruction)], CodeGenerator ())
287 loadForSwap (n, x) = gets (symbolTable . saState) >>= \st ->
288     case lookup n st of
289         (Just (ClassField IntegerArrayType _ _ _)) -> case x of
290                                                         Just x' ->
                                                             loadArrayElementVariableValue n x'
291                                                       _ -> loadVariableValue n
292         (Just (ClassField (ObjectArrayType _) _ _ _)) -> case x of
293                                                           Just x' ->
                                                               loadArrayElementVariableValue n x'
294                                                         _ -> loadVariableValue n
295         (Just ClassField {}) -> loadVariableValue n
296         (Just (LocalVariable IntegerType _)) -> loadVariableValue n
```

```
297         (Just (LocalVariable (ObjectType _) _)) -> loadVariableValue n
298         (Just (LocalVariable (CopyType _) _)) -> loadVariableValue n
299         (Just (LocalVariable IntegerArrayType _)) -> case x of
300                                                        Just x' ->
                                                            loadArrayElementVariableValue n x'
301                                                        _ -> loadVariableValue n
302         (Just (LocalVariable (ObjectArrayType _) _)) -> case x of
303                                                         Just x' ->
                                                             loadArrayElementVariableValue n x'
304                                                         _ -> loadVariableValue n
305         (Just (MethodParameter IntegerType _)) -> loadVariableValue n
306         (Just (MethodParameter (ObjectType _) _)) -> loadVariableValue n
307         (Just (MethodParameter (CopyType _) _)) -> loadVariableValue n
308         (Just (MethodParameter IntegerArrayType _)) -> case x of
309                                                          Just x' ->
                                                              loadArrayElementVariableValue n x
                                                              '
310                                                          _ -> loadVariableValue n
311         (Just (MethodParameter (ObjectArrayType _) _)) -> case x of
312                                                             Just x' ->
                                                                 loadArrayElementVariableValue
                                                                 n x'
313                                                             _ -> loadVariableValue n
314         _ -> throwError $ "ICE: Invalid variable index " ++ show n
315
316 -- | Code generation for swaps
317 cgSwap :: (SIdentifier, Maybe SExpression) -> (SIdentifier, Maybe SExpression) ->
        CodeGenerator [(Maybe Label, MInstruction)]
318 cgSwap n1 n2 = if n1 == n2 then return [] else
319     do (r1, l1, u1) <- loadForSwap n1
320        (r2, l2, u2) <- loadForSwap n2
321        u2 >> u1
322        l <- getUniqueLabel "swap"
323        let swap = [(Just l, XOR r1 r2), (Nothing, XOR r2 r1), (Nothing, XOR r1 r2)]
324        return $ l1 ++ l2 ++ swap ++ invertInstructions (l1 ++ l2)
325
326 -- | Code generation for conditionals
327 cgConditional :: SExpression -> [SStatement] -> [SStatement] -> SExpression -> CodeGenerator
        [(Maybe Label, MInstruction)]
328 cgConditional e1 s1 s2 e2 =
329     do l_test <- getUniqueLabel "test"
330        l_assert_t <- getUniqueLabel "assert_true"
331        l_test_f <- getUniqueLabel "test_false"
332        l_assert <- getUniqueLabel "assert"
333        rt <- tempRegister
334        (re1, le1, ue1) <- cgBinaryExpression e1
335        ue1
336        s1' <- concat <$> mapM cgStatement s1
337        s2' <- concat <$> mapM cgStatement s2
338        (re2, le2, ue2) <- cgBinaryExpression e2
339        ue2 >> popTempRegister --rt
340        return $ le1 ++ [(Nothing, XOR rt re1)] ++ invertInstructions le1 ++
341                 [(Just l_test, BEQ rt registerZero l_test_f), (Nothing, XORI rt $ Immediate 1)
                     ] ++
342                 s1' ++ [(Nothing, XORI rt $ Immediate 1), (Just l_assert_t, BRA l_assert), (
                     Just l_test_f, BRA l_test)] ++
343                 s2' ++ [(Just l_assert, BNE rt registerZero l_assert_t)] ++
344                 le2 ++ [(Nothing, XOR rt re2)] ++ invertInstructions le2
345
346 -- | Code generation for loops
347 cgLoop :: SExpression -> [SStatement] -> [SStatement] -> SExpression -> CodeGenerator [(Maybe
        Label, MInstruction)]
348 cgLoop e1 s1 s2 e2 =
349     do l_entry <- getUniqueLabel "entry"
350        l_test <- getUniqueLabel "test"
351        l_assert <- getUniqueLabel "assert"
```

```
352         l_exit <- getUniqueLabel "exit"
353         rt <- tempRegister
354         (re1, le1, ue1) <- cgBinaryExpression e1
355         ue1
356         s1' <- concat <$> mapM cgStatement s1
357         s2' <- concat <$> mapM cgStatement s2
358         (re2, le2, ue2) <- cgBinaryExpression e2
359         ue2 >> popTempRegister --rt
360         return $ [(Nothing, XORI rt $ Immediate 1), (Just l_entry, BEQ rt registerZero l_assert
                )] ++
361                  le1 ++ [(Nothing, XOR rt re1)] ++ invertInstructions le1 ++
362                  s1' ++ le2 ++ [(Nothing, XOR rt re2)] ++ invertInstructions le2 ++
363                  [(Just l_test, BNE rt registerZero l_exit)] ++ s2' ++
364                  [(Just l_assert, BRA l_entry), (Just l_exit, BRA l_test), (Nothing, XORI rt $
                        Immediate 1)]
365
366 -- | Code generation for object blocks
367 cgObjectBlock :: TypeName -> SIdentifier -> [SStatement] -> CodeGenerator [(Maybe Label,
        MInstruction)]
368 cgObjectBlock tp n stmt =
369     do rn <- pushRegister n
370        let push = [(Nothing, XOR rn registerSP), (Nothing, SUBI registerSP $ Immediate 1)]
371        malloc <- cgObjectConstruction tp (n, Nothing)
372        stmt' <- concat <$> mapM cgStatement stmt
373        free <- cgObjectDestruction tp (n, Nothing)
374        popRegister
375        return $ push ++ malloc ++ stmt' ++ free ++ invertInstructions push
376
377 -- | Code generation for local blocks
378 cgLocalBlock :: SIdentifier -> SExpression -> [SStatement] -> SExpression -> CodeGenerator [(
        Maybe Label, MInstruction)]
379 cgLocalBlock n e1 stmt e2 =
380     do rn <- pushRegister n
381        (re1, le1, ue1) <- cgExpression e1
382        rt1 <- tempRegister
383        popTempRegister >> ue1
384        stmt' <- concat <$> mapM cgStatement stmt
385        (re2, le2, ue2) <- cgExpression e2
386        rt2 <- tempRegister
387        popTempRegister >> ue2
388        popRegister --rn
389        l <- getUniqueLabel "localBlock"
390        let create re rt = [(Just l, XOR rn registerSP),
391                            (Nothing, XOR rt re),
392                            (Nothing, EXCH rt registerSP),
393                            (Nothing, SUBI registerSP $ Immediate 1)]
394            load = le1 ++ create re1 rt1 ++ invertInstructions le1
395            clear = le2 ++ invertInstructions (create re2 rt2) ++ invertInstructions le2
396        return $ load ++ stmt' ++ clear
397
398 -- | Code generation for calls
399 cgCall :: [(SIdentifier, Maybe SExpression)] -> [(Maybe Label, MInstruction)] -> Register ->
        CodeGenerator [(Maybe Label, MInstruction)]
400 cgCall args jump this =
401     do (ra, la, ua) <- unzip3 <$> mapM loadAddr args
402        sequence_ ua
403        rs <- gets registerStack
404        let rr = (registerThis : map snd rs) \\ (this : ra)
405            store = concatMap push $ rr ++ ra ++ [this]
406        return $ concat la ++ store ++ jump ++ invertInstructions store ++ invertInstructions (
               concat la)
407     where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]
408           loadAddr (n, e) =
409             case e of
410                 Nothing -> loadVariableAddress n
411                 Just e' -> loadArrayElementVariableAddress n e'
```

```
412
413 -- | Code generation for local calling
414 cgLocalCall :: SIdentifier -> [(SIdentifier, Maybe SExpression)]-> CodeGenerator [(Maybe Label
      , MInstruction)]
415 cgLocalCall m args = getMethodLabel m >>= \l_m -> cgCall args [(Nothing, BRA l_m)]
      registerThis
416
417 -- | Code generation for local uncalling
418 cgLocalUncall :: SIdentifier -> [(SIdentifier, Maybe SExpression)] -> CodeGenerator [(Maybe
      Label, MInstruction)]
419 cgLocalUncall m args = getMethodLabel m >>= \l_m -> cgCall args [(Nothing, RBRA l_m)]
      registerThis
420
421 -- | Returns the type associated with a given identifier
422 getType :: SIdentifier -> CodeGenerator TypeName
423 getType i = gets (symbolTable . saState) >>= \st ->
424     case lookup i st of
425         (Just (LocalVariable (ObjectType tp) _)) -> return tp
426         (Just (ClassField (ObjectType tp) _ _ _)) -> return tp
427         (Just (MethodParameter (ObjectType tp) _)) -> return tp
428         (Just (LocalVariable (ObjectArrayType tp) _)) -> return tp
429         (Just (ClassField (ObjectArrayType tp) _ _ _)) -> return tp
430         (Just (MethodParameter (ObjectArrayType tp) _)) -> return tp
431         _ -> throwError $ "ICE: Invalid object variable index " ++ show i
432
433 -- | Load the return offset for methods
434 loadMethodAddress :: (SIdentifier, Register) -> MethodName -> CodeGenerator (Register, [(Maybe
       Label, MInstruction)])
435 loadMethodAddress (o, ro) m =
436     do rv <- tempRegister
437        rt <- tempRegister
438        rtgt <- tempRegister
439        offsetMacro <- OffsetMacro <$> getType o <*> pure m
440        l <- getUniqueLabel "loadMetAdd"
441        let load = [(Just l, EXCH rv ro),
442                    (Nothing, ADDI rv offsetMacro),
443                    (Nothing, EXCH rt rv),
444                    (Nothing, XOR rtgt rt),
445                    (Nothing, EXCH rt rv),
446                    (Nothing, SUBI rv offsetMacro),
447                    (Nothing, EXCH rv ro)]
448        return (rtgt, load)
449
450 -- | Load address or value needed for calls
451 loadForCall :: (SIdentifier, Maybe SExpression) -> CodeGenerator (Register, [(Maybe Label,
      MInstruction)], CodeGenerator ())
452 loadForCall (n, e) = gets (symbolTable . saState) >>= \st ->
453     case lookup n st of
454         (Just (ClassField (ObjectArrayType _) _ _ _)) ->
455             case e of
456                 Just x' -> loadArrayElementVariableValue n x'
457                 _ -> throwError $ "ICE: Invalid variable index " ++ show n
458         (Just ClassField {}) -> loadVariableValue n
459         (Just (LocalVariable (ObjectType _) _)) -> loadVariableValue n
460         (Just (LocalVariable (CopyType _) _)) -> loadVariableValue n
461         (Just (LocalVariable (ObjectArrayType _) _)) ->
462             case e of
463                 Just x' -> loadArrayElementVariableValue n x'
464                 _ -> throwError $ "ICE: Invalid variable index " ++ show n
465         (Just _) -> loadVariableValue n
466         _ -> throwError $ "ICE: Invalid variable index " ++ show n
467
468 -- | Code generation for object calls
469 cgObjectCall :: (SIdentifier, Maybe SExpression) -> MethodName -> [(SIdentifier, Maybe
      SExpression)] -> CodeGenerator [(Maybe Label, MInstruction)]
470 cgObjectCall (o, e) m args =
```

```
471     do (ro, lo, uo) <- loadForCall (o, e)
472        rt <- tempRegister
473        (rtgt, loadAddress) <- loadMethodAddress (o, rt) m
474        l_jmp <- getUniqueLabel "l_jmp"
475        let jp = [(Nothing, SUBI rtgt $ AddressMacro l_jmp),
476                  (Just l_jmp, SWAPBR rtgt),
477                  (Nothing, NEG rtgt),
478                  (Nothing, ADDI rtgt $ AddressMacro l_jmp)]
479        call <- cgCall args jp rt
480        popTempRegister >> popTempRegister >> popTempRegister -- rv, rt & rtgt from loadMethod
               Addr
481        popTempRegister >> uo
482        let load = lo ++ [(Nothing, XOR rt ro)] ++ loadAddress ++ invertInstructions lo
483        return $ load ++ call ++ invertInstructions load
484
485 -- | Code generation for object uncalls
486 cgObjectUncall :: (SIdentifier, Maybe SExpression) -> MethodName -> [(SIdentifier, Maybe
       SExpression)] -> CodeGenerator [(Maybe Label, MInstruction)]
487 cgObjectUncall (o, e) m args =
488     do (ro, lo, uo) <- loadForCall (o, e)
489        rt <- tempRegister
490        (rtgt, loadAddress) <- loadMethodAddress (o, rt) m
491        l_jmp <- getUniqueLabel "l_jmp"
492        l_rjmp_top <- getUniqueLabel "l_rjmp_top"
493        l_rjmp_bot <- getUniqueLabel "l_rjmp_bot"
494        let jp = [(Nothing, SUBI rtgt $ AddressMacro l_jmp),
495                  (Just l_rjmp_top, RBRA l_rjmp_bot),
496                  (Just l_jmp, SWAPBR rtgt),
497                  (Nothing, NEG rtgt),
498                  (Just l_rjmp_bot, BRA l_rjmp_top),
499                  (Nothing, ADDI rtgt $ AddressMacro l_jmp)]
500        call <- cgCall args jp rt
501        popTempRegister >> popTempRegister >> popTempRegister -- rv, rt & rtgt from loadMethod
               Addr
502        popTempRegister >> uo
503        let load = lo ++ [(Nothing, XOR rt ro)] ++ loadAddress ++ invertInstructions lo
504        return $ load ++ call ++ invertInstructions load
505
506 -- | Code generation for object construction
507 cgObjectConstruction :: TypeName -> (SIdentifier, Maybe SExpression) -> CodeGenerator [(Maybe
       Label, MInstruction)]
508 cgObjectConstruction tp (n, e) =
509     do (rv, lv, uv) <- case e of
510                          Nothing -> loadVariableAddress n
511                          Just e' -> loadArrayElementVariableAddress n e'
512        rp <- tempRegister
513        rt <- tempRegister
514        popTempRegister >> popTempRegister
515        l  <- getUniqueLabel "obj_con"
516        rs <- gets registerStack
517        let rr = (registerThis : map snd rs) \\ [rp, rt]
518            store = concatMap push rr
519            malloc = [(Just l, ADDI rt $ SizeMacro tp)] ++ push rt ++ push rp
520            lb = l ++ "_bot"
521            setVtable = [(Nothing, XORI rt $ AddressMacro $ "l_" ++ tp ++ "_vt"),
522                         (Nothing, EXCH rt rp),
523                         (Nothing, ADDI rp ReferenceCounterIndex),
524                         (Nothing, XORI rt $ Immediate 1),
525                         (Nothing, EXCH rt rp),
526                         (Just lb, SUBI rp ReferenceCounterIndex),
527                         (Nothing, EXCH rp rv)]
528        uv
529        return $ store ++ malloc ++ [(Nothing, BRA "l_malloc")] ++ invertInstructions malloc ++
               invertInstructions store ++ lv ++ setVtable ++ invertInstructions lv
530     where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]
531
```

```
532 -- | Code generation for object destruction
533 cgObjectDestruction :: TypeName -> (SIdentifier, Maybe SExpression) -> CodeGenerator [(Maybe
      Label, MInstruction)]
534 cgObjectDestruction tp (n, e) =
535     do (rp, la, ua) <- case e of
536                          Nothing -> loadVariableValue n
537                          Just e' -> loadArrayElementVariableValue n e'
538        rt <- tempRegister
539        l  <- getUniqueLabel "obj_des"
540        popTempRegister >> ua
541        rs <- gets registerStack
542        let removeVtable = [(Just lt, EXCH rt rp),
543                            (Nothing, XORI rt $ AddressMacro $ "l_" ++ tp ++ "_vt"),
544                            (Nothing, ADDI rp ReferenceCounterIndex),
545                            (Nothing, EXCH rt rp),
546                            (Nothing, XORI rt $ Immediate 1),
547                            (Nothing, SUBI rp ReferenceCounterIndex)]
548            rr = (registerThis : map snd rs) \\ [rp, rt]
549            store = concatMap push rr
550            free = [(Just l, ADDI rt $ SizeMacro tp)] ++ push rt ++ push rp
551            lt = l ++ "_top"
552        return $ la ++ removeVtable ++ store ++ free ++ [(Nothing, RBRA "l_malloc")] ++
              invertInstructions (la ++ store ++ free)
553     where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]
554
555 -- | Code generation for reference construction
556 cgCopyReference :: (SIdentifier, Maybe SExpression) -> (SIdentifier, Maybe SExpression) ->
      CodeGenerator [(Maybe Label, MInstruction)]
557 cgCopyReference (n, e1) (m, e2) =
558     do (rcp, lp, up) <- case e2 of
559                           Nothing -> loadVariableValue m
560                           Just e2' -> loadArrayElementVariableValue m e2'
561        (rp, la, ua) <- case e1 of
562                          Nothing -> loadVariableValue n
563                          Just e1' -> loadArrayElementVariableValue n e1'
564        rt <- tempRegister
565        up >> ua >> popTempRegister
566        l <- getUniqueLabel "copy"
567        let reference = [(Just l, XOR rcp rp),
568                         (Nothing, ADDI rp ReferenceCounterIndex),
569                         (Nothing, EXCH rt rp),
570                         (Nothing, ADDI rt $ Immediate 1),
571                         (Nothing, EXCH rt rp),
572                         (Nothing, SUBI rp ReferenceCounterIndex)]
573        return $ lp ++ la ++ reference ++ invertInstructions (lp ++ la)
574
575 -- | Code generation for reference destruction
576 cgUnCopyReference :: (SIdentifier, Maybe SExpression) -> (SIdentifier, Maybe SExpression) ->
      CodeGenerator [(Maybe Label, MInstruction)]
577 cgUnCopyReference (n, e1) (m, e2) =
578     do (rcp, la1, ua1) <- case e2 of
579                             Nothing -> loadVariableValue m
580                             Just e2' -> loadArrayElementVariableValue m e2'
581        (rp, la2, ua2) <- case e1 of
582                            Nothing -> loadVariableValue n
583                            Just e1' -> loadArrayElementVariableValue n e1'
584        rt <- tempRegister
585        l <- getUniqueLabel "uncopy"
586        ua1 >> ua2 >> popTempRegister
587        let reference = [(Just l, XOR rcp rp),
588                         (Nothing, ADDI rp ReferenceCounterIndex),
589                         (Nothing, EXCH rt rp),
590                         (Nothing, SUBI rt $ Immediate 1),
591                         (Nothing, EXCH rt rp),
592                         (Nothing, SUBI rp ReferenceCounterIndex)]
593     --    removeRegister (m, rcp)
```

```
594         return $ la1 ++ la2 ++ reference ++ invertInstructions (la1 ++ la2)
595
596 -- | Code generation for array construction
597 cgArrayConstruction :: SExpression -> SIdentifier -> CodeGenerator [(Maybe Label, MInstruction
        )]
598 cgArrayConstruction e n =
599     do (ra, la, ua) <- loadVariableAddress n
600        (re, le, ue) <- cgExpression e
601        rp <- tempRegister
602        rt <- tempRegister
603        popTempRegister >> popTempRegister
604        l  <- getUniqueLabel "arr_con"
605        rs <- gets registerStack
606        let rr = (registerThis : map snd rs) \\ [rp, rt]
607            store = le ++ [(Just l, ADDI rt ArrayElementOffset), (Nothing, ADD rt re)] ++
                invertInstructions le ++ concatMap push rr
608            malloc = push rt ++ push rp
609            lb = l ++ "_bot"
610            initArray = la ++ le ++
611                        [(Nothing, XOR rt re),
612                         (Nothing, EXCH rt rp),
613                         (Nothing, ADDI rp ReferenceCounterIndex),
614                         (Nothing, XORI rt $ Immediate 1),
615                         (Nothing, EXCH rt rp),
616                         (Nothing, SUBI rp ReferenceCounterIndex),
617                         (Just lb, EXCH rp ra)] ++
618                        invertInstructions (la ++ le)
619        ue >> ua
620        return $ store ++ malloc ++ [(Nothing, BRA "l_malloc")] ++ invertInstructions (store ++
             malloc) ++ initArray
621     where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]
622
623 -- | Code generation for array destruction
624 cgArrayDestruction :: SExpression -> SIdentifier -> CodeGenerator [(Maybe Label, MInstruction)
        ]
625 cgArrayDestruction e n =
626     do (rp, lp, up) <- loadVariableValue n
627        (re, le, ue) <- cgExpression e
628        rt <- tempRegister
629        l  <- getUniqueLabel "obj_des"
630        popTempRegister >> ue >> up
631        rs <- gets registerStack
632        let removeArray = [(Just lt, EXCH rt rp),
633                           (Nothing, XOR rt re),
634                           (Nothing, ADDI rp ReferenceCounterIndex),
635                           (Nothing, EXCH rt rp),
636                           (Nothing, XORI rt $ Immediate 1),
637                           (Nothing, SUBI rp ReferenceCounterIndex)]
638            rr = (registerThis : map snd rs) \\ [rp, rt]
639            store = concatMap push rr
640            free = [(Just l, ADDI rt ArrayElementOffset), (Nothing, ADD rt re)] ++ push rt ++
                push rp
641            lt = l ++ "_top"
642        return $ lp ++ le ++ removeArray ++ store ++ free ++ [(Nothing, RBRA "l_malloc")] ++
            invertInstructions (lp ++ le ++ store ++ free)
643     where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]
644
645 -- | Code generation for statements
646 cgStatement :: SStatement -> CodeGenerator [(Maybe Label, MInstruction)]
647 cgStatement (Assign n modop e) = cgAssign n modop e
648 cgStatement (AssignArrElem (n, e1) modop e2) = cgAssignArrElem (n, e1) modop e2
649 cgStatement (Swap (n1, e1) (n2, e2)) = cgSwap (n1, e1) (n2, e2)
650 cgStatement (Conditional e1 s1 s2 e2) = cgConditional e1 s1 s2 e2
651 cgStatement (Loop e1 s1 s2 e2) = cgLoop e1 s1 s2 e2
652 cgStatement (ObjectBlock tp n stmt) = cgObjectBlock tp n stmt
653 cgStatement (LocalBlock _ n e1 stmt e2) = cgLocalBlock n e1 stmt e2
```

```
654 cgStatement (LocalCall m args) = cgLocalCall m args
655 cgStatement (LocalUncall m args) = cgLocalUncall m args
656 cgStatement (ObjectCall o m args) = cgObjectCall o m args
657 cgStatement (ObjectUncall o m args) = cgObjectUncall o m args
658 cgStatement (ObjectConstruction tp n) = cgObjectConstruction tp n
659 cgStatement (ObjectDestruction tp n)  = cgObjectDestruction tp n
660 cgStatement Skip = return []
661 cgStatement (CopyReference _ n m) = cgCopyReference n m
662 cgStatement (UnCopyReference _ n m)  = cgUnCopyReference n m
663 cgStatement (ArrayConstruction (_, e) n) = cgArrayConstruction e n
664 cgStatement (ArrayDestruction (_, e) n) = cgArrayDestruction e n
665
666 -- | Code generation for methods
667 cgMethod :: (TypeName, SMethodDeclaration) -> CodeGenerator [(Maybe Label, MInstruction)]
668 cgMethod (_, GMDecl m ps body) =
669     do l <- getMethodLabel m
670        rs <- addParameters
671        body' <- concat <$> mapM cgStatement body
672        clearParameters
673        let lt = l ++ "_top"
674            lb = l ++ "_bot"
675            mp = [(Just lt, BRA lb),
676                  (Nothing, ADDI registerSP $ Immediate 1),
677                  (Nothing, EXCH registerRO registerSP)]
678                  ++ concatMap pushParameter rs ++
679                 [(Nothing, EXCH registerThis registerSP),
680                  (Nothing, SUBI registerSP $ Immediate 1),
681                  (Just l, SWAPBR registerRO),
682                  (Nothing, NEG registerRO),
683                  (Nothing, ADDI registerSP $ Immediate 1),
684                  (Nothing, EXCH registerThis registerSP)]
685                  ++ invertInstructions (concatMap pushParameter rs) ++
686                 [(Nothing, EXCH registerRO registerSP),
687                  (Nothing, SUBI registerSP $ Immediate 1)]
688        return $ mp ++ body' ++ [(Just lb, BRA lt)]
689     where addParameters = mapM (pushRegister . (\(GDecl _ p) -> p)) ps
690           clearParameters = replicateM_ (length ps) popRegister
691           pushParameter r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $
                  Immediate 1)]
692
693 cgMalloc1 :: CodeGenerator [(Maybe Label, MInstruction)]
694 cgMalloc1 =
695     do -- Temp registers needed for malloc
696        r_p <- tempRegister -- Pointer to new obj
697        r_object_size <- tempRegister -- Object size
698        r_counter  <- tempRegister -- Free list index
699        r_csize  <- tempRegister -- Current cell size
700        rt <- tempRegister
701        rt2 <- tempRegister
702        r_tmp <- tempRegister
703
704        -- Expressions and sub routines
705        (r_e1_outer, l_e1_outer, u_e1_outer) <- cgBinOp Lt r_csize r_object_size
706        (r_e2_outer, l_e2_outer, u_e2_outer) <- cgBinOp Lt r_csize r_object_size
707        (r_fl, l_fl, u_fl) <- loadFreeListAddress r_counter
708        (r_block, l_block, u_block) <- loadHeadAtFreeList r_fl
709        (r_e1_inner, l_e1_inner, u_e1_inner) <- cgBinOp Neq r_block registerZero
710        (r_e2_i1, l_e2_i1, u_e2_i1) <- cgBinOp Neq r_p r_tmp
711        (r_e2_i2, l_e2_i2, u_e2_i2) <- cgBinOp Eq r_tmp registerZero
712        (r_e2_i3, l_e2_i3, u_e2_i3) <- cgBinOp BitOr r_e2_i1 r_e2_i2
713
714        let tmpRegisterList = [rt, rt2, r_tmp, r_e1_outer, r_e2_outer, r_fl, r_block,
               r_e1_inner, r_e2_i1, r_e2_i2, r_e2_i3]
715
716        -- Update state after evaluating expressions and subroutines
717        u_e2_i3 >> u_e2_i2 >> u_e2_i1 >> u_e1_inner
```

```
718         u_block >> u_fl >> u_e2_outer >> u_e1_outer
719         popTempRegister >> popTempRegister >> popTempRegister >> popTempRegister >>
                popTempRegister >> popTempRegister >> popTempRegister
720
721         let l_o_test = "l_o_test"
722             l_o_assert_t = "l_o_assert_true"
723             l_o_test_f = "l_o_test_false"
724             l_o_assert = "l_o_assert"
725             l_i_test = "l_i_test"
726             l_i_assert_t = "l_i_assert_true"
727             l_i_test_f = "l_i_test_false"
728             l_i_assert = "l_i_assert"
729             l_m_top = "l_malloc1_top"
730             l_m_bot = "l_malloc1_bot"
731             l_m_entry = "l_malloc1"
732             malloc = [(Just l_m_top, BRA l_m_bot),                  --
733                       (Nothing, ADDI registerSP $ Immediate 1),
734                       (Nothing, EXCH registerRO registerSP)]        -- Pop return offset from
                              stack
735                      ++ invertInstructions l_fl ++
736                      [(Just l_m_entry, SWAPBR registerRO),          -- Malloc1 entry/exit point
737                       (Nothing, NEG registerRO),                    -- Restore return offset
738                       (Nothing, EXCH registerRO registerSP),        -- Push return offset to stack
739                       (Nothing, SUBI registerSP $ Immediate 1)]
740                     ++ l_fl
741                     ++ l_block
742                     ++ l_e1_outer                                  -- Set r_e1 -> c_size <
                            obj_size
743                     ++ [(Nothing, XOR rt r_e1_outer)]              -- r_t = r_e1_o
744                     ++ invertInstructions l_e1_outer ++            -- Clear r_e1_o
745                     [(Just l_o_test, BEQ rt registerZero l_o_test_f),
746                      (Nothing, XORI rt $ Immediate 1),             -- S1_outer start
747                      (Nothing, ADDI r_counter $ Immediate 1)]      -- counter++
748                      ++ invertInstructions l_block ++
749                      [(Nothing, RL r_csize $ Immediate 1)]         -- call double(csize)
750                     ++ concatMap pushRegisterToStack tmpRegisterList
751                     ++
752                     [(Nothing, BRA l_m_entry)]                     -- call malloc1()
753                     ++ invertInstructions(concatMap pushRegisterToStack tmpRegisterList)
754                     ++
755                     [(Nothing, RR r_csize $ Immediate 1),          -- uncall double(csize)
756                      (Nothing, SUBI r_counter $ Immediate 1),      -- counter++
757                      (Nothing, XORI rt $ Immediate 1),             -- S1_outer end
758                      (Just l_o_assert_t, BRA l_o_assert),
759                      (Just l_o_test_f, BRA l_o_test)]
760                      ++ l_e1_inner ++                              -- Set r_e1_i -> r_block != 0
                             (S2_OUTER)
761                     [(Nothing, XOR rt2 r_e1_inner)]                -- Set rt2 -> r_e1_i
762                      ++ invertInstructions l_e1_inner ++           -- Clear r_e1_i
763                     [(Just l_i_test, BEQ rt2 registerZero l_i_test_f),
764                      (Nothing, XORI rt2 $ Immediate 1),            -- S1_inner start
765                      (Nothing, ADD r_p r_block),                   -- Set address of p to said
                            block
766                      (Nothing, SUB r_block r_p),                   -- Clear r_block
767                      (Nothing, EXCH r_tmp r_p),                    -- Load address of next block
768                      (Nothing, EXCH r_tmp r_fl),                   -- Set address of next block
                            as head of current free list
769                      (Nothing, XOR r_tmp r_p),                     -- Clear address of next block
770                      (Nothing, XORI rt2 $ Immediate 1),            -- S1_inner end
771                      (Just l_i_assert_t, BRA l_i_assert),
772                      (Just l_i_test_f, BRA l_i_test),
773                      (Nothing, ADDI r_counter $ Immediate 1),      -- S2_inner start
774                      (Nothing, RL r_csize $ Immediate 1)]          -- call double(csize)
775                     ++ concatMap pushRegisterToStack tmpRegisterList ++
776                     [(Nothing, BRA l_m_entry)]                     -- call malloc1()
777                     ++ invertInstructions(concatMap pushRegisterToStack tmpRegisterList) ++
```

```
778                     [(Nothing, RR r_csize $ Immediate 1),       -- uncall double(csize)
779                      (Nothing, SUBI r_counter $ Immediate 1),   -- counter -= 1
780                      (Nothing, XOR r_tmp r_p),                  -- Copy current address of p
781                      (Nothing, EXCH r_tmp r_fl),                -- Store address in current
                             free list
782                      (Nothing, ADD r_p r_csize),                -- Set p to other half of the
                             block we're splitting
783                      (Just l_i_assert, BNE rt2 registerZero l_i_assert_t),
784                      (Nothing, EXCH r_tmp r_fl),
785                      (Nothing, SUB r_p r_csize)]
786                     ++ l_e2_i1                                  -- set r_e2_i1 <- p - csize !=
                              free_list[counter]
787                      ++ l_e2_i2                                 -- set r_e2_i2 <- free_list[
                             counter] = 0
788                      ++ l_e2_i3                                 -- set r_e2_i3 <- r_e2_i1 ||
                             Â r_e2_i2
789                      ++ [(Nothing, XOR rt2 r_e2_i3)]            -- Set rt2 -> r_i_2
790                      ++ invertInstructions l_e2_i3              -- Clear r_i_2
791                      ++ invertInstructions l_e2_i2
792                      ++ invertInstructions l_e2_i1 ++
793                      [(Nothing, ADD r_p r_csize),
794                       (Nothing, EXCH r_tmp r_fl),               -- S2_outer end
795                       (Just l_o_assert, BNE rt registerZero l_o_assert_t)]
796                      ++ l_e2_outer                              -- Set r_e2 -> c_size <
                             obj_size
797                      ++ [(Nothing, XOR rt r_e2_outer)]          -- r_t = r_e1_o
798                      ++ invertInstructions l_e2_outer           -- Clear r_e1_o
799                      ++ [(Just l_m_bot, BRA l_m_top)]           -- Go to top
800         return malloc
801     where pushRegisterToStack r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $
            Immediate 1)]
802
803 cgMalloc :: CodeGenerator [(Maybe Label, MInstruction)]
804 cgMalloc =
805     do rp <- tempRegister -- Pointer to new obj
806        ros <- tempRegister -- Object size
807        rc  <- tempRegister -- Free list index
808        rs  <- tempRegister -- Current cell size
809        popTempRegister >> popTempRegister >> popTempRegister >> popTempRegister
810        let malloc = [(Just "l_malloc_top", BRA "l_malloc_bot")]
811                     ++
812                     [(Just "l_malloc", SWAPBR registerRO),
813                      (Nothing, NEG registerRO),
814                      (Nothing, ADDI rs $ Immediate 2),
815                      (Nothing, XOR rc registerZero)]
816                     ++ concatMap pop [rp, ros]
817                     ++ push registerRO ++
818                     [(Nothing, BRA "l_malloc1")]
819                     ++ pop registerRO
820                     ++ concatMap push [ros, rp] ++
821                     [(Nothing, XOR rc registerZero),
822                      (Nothing, SUBI rs $ Immediate 2),
823                      (Just "l_malloc_bot", BRA "l_malloc_top")]
824        return malloc
825     where pop r = [(Nothing, ADDI registerSP $ Immediate 1), (Nothing, EXCH r registerSP)]
826           push r = invertInstructions (pop r)
827
828 -- | Code generation for virtual tables
829 cgVirtualTables :: CodeGenerator [(Maybe Label, MInstruction)]
830 cgVirtualTables = concat <$> (gets (virtualTables . saState) >>= mapM vtInstructions)
831     where vtInstructions (n, ms) = zip (vtLabel n) <$> mapM vtData ms
832           vtData m = DATA . AddressMacro <$> getMethodLabel m
833           vtLabel n = (Just $ "l_" ++ n ++ "_vt") : repeat Nothing
834
835 -- | Returns the main class label
836 getMainLabel :: CodeGenerator Label
```

```
837 getMainLabel = gets (mainMethod . saState) >>= getMethodLabel
838
839 -- | Fetches the main class from the class analysis state
840 getMainClass :: CodeGenerator TypeName
841 getMainClass = gets (mainClass . caState . saState) >>= \mc ->
842     case mc of
843         (Just tp) -> return tp
844         Nothing -> throwError "ICE: No main method defined"
845
846 -- | Fetches the field of a given type name
847 getFields :: TypeName -> CodeGenerator [VariableDeclaration]
848 getFields tp =
849     do cs <- gets (classes . caState . saState)
850        case lookup tp cs of
851            (Just (GCDecl _ _ fs _)) -> return fs
852            Nothing -> throwError $ "ICE: Unknown class " ++ tp
853
854 -- | Code generation for output
855 cgOutput :: TypeName -> CodeGenerator ([(Maybe Label, MInstruction)], [(Maybe Label,
        MInstruction)])
856 cgOutput tp =
857     do mfs <- getFields tp
858        co <- concat <$> mapM cgCopyOutput (zip [1..] mfs)
859        return (map cgStatic mfs, co)
860     where cgStatic (GDecl _ n) = (Just $ "l_r_" ++ n, DATA $ Immediate 0)
861           cgCopyOutput(o, GDecl _ n) =
862               do rt <- tempRegister
863                  ra <- tempRegister
864                  popTempRegister >> popTempRegister
865                  let copy = [ADDI registerThis ReferenceCounterIndex,
866                              ADDI registerThis $ Immediate o,
867                              EXCH rt registerThis,
868                              XORI ra $ AddressMacro $ "l_r_" ++ n,
869                              EXCH rt ra,
870                              XORI ra $ AddressMacro $ "l_r_" ++ n,
871                              SUBI registerThis $ Immediate o,
872                              SUBI registerThis ReferenceCounterIndex]
873                  return $ zip (repeat Nothing) copy
874
875 -- | Generates code for the program entry point
876 cgProgram :: SProgram -> CodeGenerator PISA.MProgram
877 cgProgram p =
878     do vt <- cgVirtualTables
879        malloc <- cgMalloc
880        malloc1 <- cgMalloc1
881        rv <- tempRegister -- V table register
882        rb <- tempRegister -- Memory block register
883        popTempRegister >> popTempRegister
884        ms <- concat <$> mapM cgMethod p
885        l_main <- getMainLabel
886        mtp <- getMainClass
887        (out, co) <- cgOutput mtp
888        let mvt = "l_" ++ mtp ++ "_vt"
889            mn = [(Just "start", BRA "top"),
890                  (Nothing, START),
891                  (Nothing, ADDI registerFLPs ProgramSize),    -- Init free list pointer list
892                  (Nothing, XOR registerHP registerFLPs),      -- Init heap pointer
893                  (Nothing, ADDI registerHP FreeListsSize),    -- Init space for FLPs list
894                  (Nothing, XOR rb registerHP),                -- Store address of initial
                        memory block in rb
895                  (Nothing, ADDI registerFLPs FreeListsSize), -- Index to end of free lists
896                  (Nothing, SUBI registerFLPs $ Immediate 1), -- Index to last element of free
                         lists
897                  (Nothing, EXCH rb registerFLPs),             -- Store address of first block
                        in last element of free lists
898                  (Nothing, ADDI registerFLPs $ Immediate 1), -- Index to end of free lists
```

```
899                 (Nothing, SUBI registerFLPs FreeListsSize),  -- Index to beginning of free
                        lists
900                 (Nothing, XOR registerSP registerHP),        -- Init stack pointer 1/2
901                 (Nothing, ADDI registerSP StackOffset),      -- Init stack pointer 2/2
902                 (Nothing, SUBI registerSP $ SizeMacro mtp),  -- Allocate space for main on
                        stack
903                 (Nothing, XOR registerThis registerSP),      -- Store address of main object
904                 (Nothing, XORI rv $ AddressMacro mvt),       -- Store address of vtable in rv
905                 (Nothing, EXCH rv registerThis),             -- Add address of vtable to
                        stack
906                 (Nothing, SUBI registerSP $ Immediate 1),    -- Add address of vtable to
                        stack
907                 (Nothing, EXCH registerThis registerSP),     -- Push 'this' to stack
908                 (Nothing, SUBI registerSP $ Immediate 1),    -- Push 'this' to stack
909                 (Nothing, BRA l_main),                       -- Execute main
910                 (Nothing, ADDI registerSP $ Immediate 1),    -- Pop 'this'
911                 (Nothing, EXCH registerThis registerSP)]     -- Pop 'this'
912                  ++ co ++
913                [(Nothing, ADDI registerSP $ Immediate 1),    -- Pop vtable address
914                 (Nothing, EXCH rv registerThis),             -- Pop vtable address
915                 (Nothing, XORI rv $ AddressMacro mvt),       -- Clear rv
916                 (Nothing, XOR registerThis registerSP),      -- Clear 'this'
917                 (Nothing, ADDI registerSP $ SizeMacro mtp),  -- Deallocate space for main
918                 (Nothing, SUBI registerSP StackOffset),      -- Clear stack pointer
919                 (Nothing, XOR registerSP registerHP),        -- Clear stack pointer
920                 (Nothing, SUBI registerHP FreeListsSize),    -- Reset Heap pointer (For
                        pretty output)
921                 (Nothing, XOR registerHP registerFLPs),      -- Reset Heap pointer (For
                        pretty output)
922                 (Nothing, SUBI registerFLPs ProgramSize),    -- Reset Free lists pointer (For
                         pretty output)
923                 (Just "finish", FINISH)]
924         return $ PISA.GProg $ [(Just "top", BRA "start")] ++ out ++ vt ++ malloc ++ malloc1 ++
                ms ++ mn
925
926
927 -- | Generates code for a program
928 generatePISA :: (SProgram, SAState) -> Except String (PISA.MProgram, SAState)
929 generatePISA (p, s) = second saState <$> runStateT (runCG $ cgProgram p) (initialState s)
930
931 showPISAProgram :: (Show a, MonadIO m) => (t, a) -> m ()
932 showPISAProgram (_, s) = pPrint s
```

MacroExpander.hs

```haskell
{-# LANGUAGE GeneralizedNewtypeDeriving #-}

module MacroExpander (expandMacros) where

import Data.Maybe
import Data.List

import Control.Monad.Reader
import Control.Monad.Except
import Control.Arrow

import AST hiding (Program, GProg, Offset)
import PISA

import Debug.Trace (trace, traceShow)

import ScopeAnalyzer
import ClassAnalyzer

type Size = Integer
type Address = Integer
type Offset = Integer

data MEState = MEState {
    addressTable :: [(Label, Address)],
    sizeTable :: [(TypeName, Size)],
    offsetTable :: [(TypeName, [(MethodName, Offset)])],
    programSize :: Size,
    freeListsSize :: Size,
    stackOffset :: Offset,
    initialMemoryBlockSize :: Size,
    referenceCounterIndex :: Offset,
    arrayElementOffset :: Offset
} deriving (Show, Eq)

newtype MacroExpander a = MacroExpander { runME :: ReaderT MEState (Except String) a }
    deriving (Functor, Applicative, Monad, MonadReader MEState, MonadError String)

-- | Returns the offset table generated from the an indexed virtual table
getOffsetTable :: SAState -> [(TypeName, [(MethodName, Offset)])]
getOffsetTable s = map (second (map toOffset)) indexedVT
    where indexedVT = map (second $ zip [0..]) $ virtualTables s
          toOffset (i, m) = (getName $ lookup m $ symbolTable s, i)
          getName (Just (Method _ n)) = n
          getName _ = error "ICE: Invalid method index"

-- | Initializes the macro state containing the address, size, offset tables and the program size
initialState :: MProgram -> SAState -> MEState
initialState (GProg p) s = MEState {
    addressTable = mapMaybe toPair $ zip [0..] p,
    sizeTable = (classSize . caState) s,
    offsetTable = getOffsetTable s,
    programSize = genericLength p,
    freeListsSize = 10,
    stackOffset = 2048,
    initialMemoryBlockSize = 1024,
    referenceCounterIndex = 1,
    arrayElementOffset = 2
}
    where toPair (a, (Just l, _)) = Just (l, a)
          toPair _ = Nothing

```

```
63 -- | Returns the address of a given label
64 getAddress :: Label -> MacroExpander Address
65 getAddress l = asks addressTable >>= \at ->
66     case lookup l at of
67         (Just i) -> return i
68         Nothing -> throwError $ "ICE: Unknown label " ++ l
69
70 -- | Returns the size of a given class name
71 getSize :: TypeName -> MacroExpander Size
72 getSize tn = asks sizeTable >>= \st ->
73     case lookup tn st of
74         (Just s) -> return s
75         Nothing -> throwError $ "ICE: Unknown type " ++ tn
76
77 -- | Returns the off set of a method for a given class name and method
78 getOffset :: TypeName -> MethodName -> MacroExpander Offset
79 getOffset tn mn = asks offsetTable >>= \ot ->
80     case lookup tn ot of
81         Nothing -> throwError $ "ICE: Unknown type " ++ tn
82         (Just mo) -> case lookup mn mo of
83                         Nothing -> throwError $ "ICE: Unknown method " ++ mn
84                         (Just o) -> return o
85
86 -- | Macro definitions
87 meMacro :: Macro -> MacroExpander Integer
88 meMacro (Immediate i) = return i
89 meMacro (AddressMacro l) = getAddress l
90 meMacro (SizeMacro tn) = getSize tn
91 meMacro (OffsetMacro tn mn) = getOffset tn mn
92 meMacro ProgramSize = asks programSize
93 meMacro FreeListsSize = asks freeListsSize
94 meMacro StackOffset = asks stackOffset
95 meMacro InitialMemoryBlockSize = asks initialMemoryBlockSize
96 meMacro ReferenceCounterIndex = asks referenceCounterIndex
97 meMacro ArrayElementOffset = asks arrayElementOffset
98
99 -- | Macro instructions
100 meInstruction :: MInstruction -> MacroExpander Instruction
101 meInstruction (ADD r1 r2) = return $ ADD r1 r2
102 meInstruction (ADDI r m) = ADDI r <$> meMacro m
103 meInstruction (ANDX r1 r2 r3) = return $ ANDX r1 r2 r3
104 meInstruction (ANDIX r1 r2 m) = ANDIX r1 r2 <$> meMacro m
105 meInstruction (NORX r1 r2 r3) = return $ NORX r1 r2 r3
106 meInstruction (NEG r) = return $ NEG r
107 meInstruction (ORX r1 r2 r3) = return $ ORX r1 r2 r3
108 meInstruction (ORIX r1 r2 m) = ORIX r1 r2 <$> meMacro m
109 meInstruction (RL r m) = RL r <$> meMacro m
110 meInstruction (RLV r1 r2 ) = return $ RLV r1 r2
111 meInstruction (RR r m) = RR r <$> meMacro m
112 meInstruction (RRV r1 r2 ) = return $ RRV r1 r2
113 meInstruction (SLLX r1 r2 m) = SLLX r1 r2 <$> meMacro m
114 meInstruction (SLLVX r1 r2 r3) = return $ SLLVX r1 r2 r3
115 meInstruction (SRAX r1 r2 m) = SRAX r1 r2 <$> meMacro m
116 meInstruction (SRAVX r1 r2 r3) = return $ SRAVX r1 r2 r3
117 meInstruction (SRLX r1 r2 m) = SRLX r1 r2 <$> meMacro m
118 meInstruction (SRLVX r1 r2 r3) = return $ SRLVX r1 r2 r3
119 meInstruction (SUB r1 r2) = return $ SUB r1 r2
120 meInstruction (XOR r1 r2) = return $ XOR r1 r2
121 meInstruction (XORI r m) = XORI r <$> meMacro m
122 meInstruction (BEQ r1 r2 l) = return $ BEQ r1 r2 l
123 meInstruction (BGEZ r l) = return $ BGEZ r l
124 meInstruction (BGTZ r l) = return $ BGTZ r l
125 meInstruction (BLEZ r l) = return $ BLEZ r l
126 meInstruction (BLTZ r l) = return $ BLTZ r l
127 meInstruction (BNE r1 r2 l) = return $ BNE r1 r2 l
128 meInstruction (BRA l) = return $ BRA l
```

```
129 meInstruction (EXCH r1 r2) = return $ EXCH r1 r2
130 meInstruction (SWAPBR r) = return $ SWAPBR r
131 meInstruction (RBRA l) = return $ RBRA l
132 meInstruction START = return START
133 meInstruction FINISH = return FINISH
134 meInstruction (DATA m) = DATA <$> meMacro m
135 meInstruction (SUBI r m) = SUBI r <$> meMacro m
136
137 -- | Macro Expand a program
138 meProgram :: MProgram -> MacroExpander Program
139 meProgram (GProg p) = GProg <$> mapM expandPair p
140     where expandPair (l, i) = (,) l <$> meInstruction i
141
142 -- | Interface for starting the macro extension process
143 expandMacros :: (MProgram, SAState) -> Except String Program
144 expandMacros (p, s) = runReaderT (runME $ meProgram p) $ initialState p s
```

ROOPLPPC.hs

```
1  import Control.Monad.Except
2  import System.IO
3  import System.Environment
4
5  import PISA
6  import Parser
7  import ClassAnalyzer
8  import ScopeAnalyzer
9  import TypeChecker
10 import CodeGenerator
11 import MacroExpander
12
13 import Data.List.Split
14
15 type Error = String
16
17 main :: IO ()
18 main =
19     do args <- getArgs
20        when (null args) (error "Supply input filename.\nUsage: ROOPLPPC input.rplpp output.pal
               \n")
21        when (length args > 2) (error "Too many arguments.\nUsage: ROOPLPPC input.rplpp output.
               pal\n")
22        handle <- openFile (head args) ReadMode
23        input <- hGetContents handle
24        let output =  if length args == 2 then last args else head (splitOn "." (head args)) ++
                ".pal"
25        either (hPutStrLn stderr) (writeProgram output) $ compileProgram input
26
27 compileProgram :: String -> Either Error PISA.Program
28 compileProgram s =
29     runExcept $
30     parseString s
31     >>= classAnalysis
32     >>= scopeAnalysis
33     >>= typeCheck
34     >>= generatePISA
35     >>= expandMacros
```

APPENDIX C

Example Ouput

LinkedList.rplpp

```
class Cell
    Cell next
    int data

    method constructor(int value)
        data ^= value

    method append(Cell cell)
        if next = nil & cell != nil then
            next <=> cell                    // Store as next cell if current cell is end of
                list
        else skip
        fi next != nil & cell = nil

        if next != nil then
            call next::append(cell)          // Recusively search until we reach end of list
        else skip
        fi next != nil

class LinkedList
    Cell head
    int listLength

    method insertHead(Cell cell)
        if head = nil & cell != nil then
            head <=> cell                // Set cell as head of list if list is empty
        else skip
        fi head != nil & cell = nil

    method appendCell(Cell cell)
        call insertHead(cell)            // Insert as head if empty list

        if head != nil then
            call head::append(cell)      // Iterate list until we reach the end, then insert
                the node
        else skip
        fi head != nil

        listLength += 1                  // Increment lenght

    method prependCell(Cell cell)
        call insertHead(cell)            // Insert as head if empty list

        if cell != nil & head != nil then
            call cell::append(head)      // Set cell.next = head. head = nil after execution
```

```
44          else skip
45          fi cell != nil & head = nil
46
47          if cell != nil & head = nil then
48              cell <=> head               // Set head = cell. Cell is nil after execution
49          else skip
50          fi cell = nil & head != nil
51
52          listLength += 1                 // Increment length
53
54      method length(int result)
55          result ^= listLength
56
57  class Program
58      LinkedList linkedList
59      int sumResult
60      int listLength
61
62      method main()
63          new LinkedList linkedList           // Init new linked linkedList
64          listLength += 10
65
66          local int x = 0
67          from x = 0 do
68              skip
69          loop
70              local Cell cell = nil
71              new Cell cell                       // Instantiate new cell
72              call cell::constructor(x)           // Set value of cell
73              call linkedList::appendCell(cell)   // Append it to the linkedList
74              delocal Cell cell = nil
75              x += 1
76          until x = listLength
77          delocal int x = listLength
```

LinkedList.pal

```
;; pendulum pal file
top:                    BRA     start
l_r_linkedList:         DATA    0
l_r_sumResult:          DATA    0
l_r_listLength:         DATA    0
l_Program_vt:           DATA    972
l_LinkedList_vt:        DATA    459
                        DATA    562
                        DATA    704
                        DATA    945
l_Cell_vt:              DATA    223
                        DATA    252
l_malloc_top:           BRA     l_malloc_bot
l_malloc:               SWAPBR  $2
                        NEG     $2
                        ADDI    $9 2
                        XOR     $8 $0
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
                        BRA     l_malloc1
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        XOR     $8 $0
                        ADDI    $9 -2
l_malloc_bot:           BRA     l_malloc_top
l_malloc1_top:          BRA     l_malloc1_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        SUB     $17 $8
                        XOR     $17 $4
l_malloc1:              SWAPBR  $2
                        NEG     $2
                        EXCH    $2 $1
                        ADDI    $1 -1
                        XOR     $17 $4
                        ADD     $17 $8
                        EXCH    $19 $17
                        XOR     $18 $19
                        EXCH    $19 $17
                        XOR     $13 $9
                        SUB     $13 $7
cmp_top_7:              BGEZ    $13 cmp_bot_8
                        XORI    $14 1
cmp_bot_8:              BGEZ    $13 cmp_top_7
                        XOR     $10 $14
cmp_bot_8_i:            BGEZ    $13
    cmp_top_7_i
                        XORI    $14 1
cmp_top_7_i:            BGEZ    $13
    cmp_bot_8_i
                        ADD     $13 $7
                        XOR     $13 $9
l_o_test:               BEQ     $10 $0
    l_o_test_false
                        XORI    $10 1
                        ADDI    $8 1
                        EXCH    $19 $17
                        XOR     $18 $19
                        EXCH    $19 $17
                        RL      $9 1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $11 $1
                        ADDI    $1 -1
                        EXCH    $12 $1
                        ADDI    $1 -1
                        EXCH    $14 $1
                        ADDI    $1 -1
                        EXCH    $16 $1
                        ADDI    $1 -1
                        EXCH    $17 $1
                        ADDI    $1 -1
                        EXCH    $18 $1
                        ADDI    $1 -1
                        EXCH    $20 $1
                        ADDI    $1 -1
                        EXCH    $21 $1
                        ADDI    $1 -1
                        EXCH    $22 $1
                        ADDI    $1 -1
                        EXCH    $23 $1
                        ADDI    $1 -1
                        BRA     l_malloc1
                        ADDI    $1 1
                        EXCH    $23 $1
                        ADDI    $1 1
                        EXCH    $22 $1
                        ADDI    $1 1
                        EXCH    $21 $1
                        ADDI    $1 1
                        EXCH    $20 $1
                        ADDI    $1 1
                        EXCH    $18 $1
                        ADDI    $1 1
                        EXCH    $17 $1
                        ADDI    $1 1
                        EXCH    $16 $1
                        ADDI    $1 1
                        EXCH    $14 $1
                        ADDI    $1 1
                        EXCH    $12 $1
                        ADDI    $1 1
                        EXCH    $11 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        RR      $9 1
                        ADDI    $8 -1
                        XORI    $10 1
l_o_assert_true:        BRA     l_o_assert
l_o_test_false:         BRA     l_o_test
cmp_top_11:             BEQ     $18 $0
    cmp_bot_12
                        XORI    $20 1
cmp_bot_12:             BEQ     $18 $0
    cmp_top_11
                        XOR     $11 $20
```

```
cmp_bot_12_i:           BEQ    $18 $0
    cmp_top_11_i
                        XORI   $20 1
cmp_top_11_i:           BEQ    $18 $0
    cmp_bot_12_i
l_i_test:               BEQ    $11 $0
    l_i_test_false
                        XORI   $11 1
                        ADD    $6 $18
                        SUB    $18 $6
                        EXCH   $12 $6
                        EXCH   $12 $17
                        XOR    $12 $6
                        XORI   $11 1
l_i_assert_true:        BRA    l_i_assert
l_i_test_false:         BRA    l_i_test
                        ADDI   $8 1
                        RL     $9 1
                        EXCH   $10 $1
                        ADDI   $1 -1
                        EXCH   $11 $1
                        ADDI   $1 -1
                        EXCH   $12 $1
                        ADDI   $1 -1
                        EXCH   $14 $1
                        ADDI   $1 -1
                        EXCH   $16 $1
                        ADDI   $1 -1
                        EXCH   $17 $1
                        ADDI   $1 -1
                        EXCH   $18 $1
                        ADDI   $1 -1
                        EXCH   $20 $1
                        ADDI   $1 -1
                        EXCH   $21 $1
                        ADDI   $1 -1
                        EXCH   $22 $1
                        ADDI   $1 -1
                        EXCH   $23 $1
                        ADDI   $1 -1
                        BRA    l_malloc1
                        ADDI   $1 1
                        EXCH   $23 $1
                        ADDI   $1 1
                        EXCH   $22 $1
                        ADDI   $1 1
                        EXCH   $21 $1
                        ADDI   $1 1
                        EXCH   $20 $1
                        ADDI   $1 1
                        EXCH   $18 $1
                        ADDI   $1 1
                        EXCH   $17 $1
                        ADDI   $1 1
                        EXCH   $16 $1
                        ADDI   $1 1
                        EXCH   $14 $1
                        ADDI   $1 1
                        EXCH   $12 $1
                        ADDI   $1 1
                        EXCH   $11 $1
                        ADDI   $1 1
                        EXCH   $10 $1
                        RR     $9 1
                        ADDI   $8 -1
                        XOR    $12 $6
                        EXCH   $12 $17
                        ADD    $6 $9
l_i_assert:             BNE    $11 $0
    l_i_assert_true
                        EXCH   $12 $17
                        SUB    $6 $9
cmp_top_13:             BEQ    $6 $12
    cmp_bot_14
                        XORI   $21 1
cmp_bot_14:             BEQ    $6 $12
    cmp_top_13
cmp_top_15:             BNE    $12 $0
    cmp_bot_16
                        XORI   $22 1
cmp_bot_16:             BNE    $12 $0
    cmp_top_15
                        ORX    $23 $21 $22
                        XOR    $11 $23
                        ORX    $23 $21 $22
cmp_bot_16_i:           BNE    $12 $0
    cmp_top_15_i
                        XORI   $22 1
cmp_top_15_i:           BNE    $12 $0
    cmp_bot_16_i
cmp_bot_14_i:           BEQ    $6 $12
    cmp_top_13_i
                        XORI   $21 1
cmp_top_13_i:           BEQ    $6 $12
    cmp_bot_14_i
                        ADD    $6 $9
                        EXCH   $12 $17
l_o_assert:             BNE    $10 $0
    l_o_assert_true
                        XOR    $15 $9
                        SUB    $15 $7
cmp_top_9:              BGEZ   $15 cmp_bot_10
                        XORI   $16 1
cmp_bot_10:             BGEZ   $15 cmp_top_9
                        XOR    $10 $16
cmp_bot_10_i:           BGEZ   $15
    cmp_top_9_i
                        XORI   $16 1
cmp_top_9_i:            BGEZ   $15
    cmp_bot_10_i
                        ADD    $15 $7
                        XOR    $15 $9
l_malloc1_bot:          BRA    l_malloc1_top
l_constructor_5_top:    BRA
    l_constructor_5_bot
                        ADDI   $1 1
                        EXCH   $2 $1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $3 $1
                        ADDI   $1 -1
l_constructor_5:        SWAPBR $2
                        NEG    $2
                        ADDI   $1 1
                        EXCH   $3 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        EXCH   $2 $1
                        ADDI   $1 -1
                        ADD    $7 $3
                        ADDI   $7 3
                        EXCH   $8 $7
```

```
                        ADDI    $7 -3
                        SUB     $7 $3
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADD     $7 $3
                        ADDI    $7 3
                        EXCH    $8 $7
                        ADDI    $7 -3
                        SUB     $7 $3
l_constructor_5_bot:    BRA     l_constructor_5_top
l_append_6_top:         BRA     l_append_6_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $3 $1
                        ADDI    $1 -1
l_append_6:             SWAPBR  $2
                        NEG     $2
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
cmp_top_21:             BNE     $9 $0
    cmp_bot_22
                        XORI    $10 1
cmp_bot_22:             BNE     $9 $0
    cmp_top_21
                        EXCH    $11 $6
cmp_top_23:             BEQ     $11 $0
    cmp_bot_24
                        XORI    $12 1
cmp_bot_24:             BEQ     $11 $0
    cmp_top_23
                        ANDX    $13 $10 $12
f_top_25:               BEQ     $13 $0
    f_bot_26
                        XORI    $14 1
f_bot_26:               BEQ     $13 $0
    f_top_25
                        XOR     $7 $14
f_bot_26_i:             BEQ     $13 $0
    f_top_25_i
                        XORI    $14 1
f_top_25_i:             BEQ     $13 $0
    f_bot_26_i
                        ANDX    $13 $10 $12
cmp_bot_24_i:           BEQ     $11 $0
    cmp_top_23_i
                        XORI    $12 1
cmp_top_23_i:           BEQ     $11 $0
    cmp_bot_24_i
                        EXCH    $11 $6
cmp_bot_22_i:           BNE     $9 $0
    cmp_top_21_i
                        XORI    $10 1
cmp_top_21_i:           BNE     $9 $0
    cmp_bot_22_i
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
test_17:                BEQ     $7 $0
    test_false_19
                        XORI    $7 1
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
                        EXCH    $10 $6
swap_27:                XOR     $9 $10
                        XOR     $10 $9
                        XOR     $9 $10
                        EXCH    $10 $6
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
                        XORI    $7 1
assert_true_18:         BRA     assert_20
test_false_19:          BRA     test_17
assert_20:              BNE     $7 $0
    assert_true_18
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
cmp_top_28:             BEQ     $9 $0
    cmp_bot_29
                        XORI    $10 1
cmp_bot_29:             BEQ     $9 $0
    cmp_top_28
                        EXCH    $11 $6
cmp_top_30:             BNE     $11 $0
    cmp_bot_31
                        XORI    $12 1
cmp_bot_31:             BNE     $11 $0
    cmp_top_30
                        ANDX    $13 $10 $12
f_top_32:               BEQ     $13 $0
    f_bot_33
                        XORI    $14 1
f_bot_33:               BEQ     $13 $0
    f_top_32
                        XOR     $7 $14
f_bot_33_i:             BEQ     $13 $0
    f_top_32_i
                        XORI    $14 1
f_top_32_i:             BEQ     $13 $0
    f_bot_33_i
                        ANDX    $13 $10 $12
cmp_bot_31_i:           BNE     $11 $0
    cmp_top_30_i
                        XORI    $12 1
cmp_top_30_i:           BNE     $11 $0
    cmp_bot_31_i
                        EXCH    $11 $6
cmp_bot_29_i:           BEQ     $9 $0
    cmp_top_28_i
```

```
342                         XORI    $10 1
343 cmp_top_28_i:           BEQ     $9 $0
        cmp_bot_29_i
344                         ADD     $8 $3
345                         ADDI    $8 2
346                         EXCH    $9 $8
347                         ADDI    $8 -2
348                         SUB     $8 $3
349                         ADD     $8 $3
350                         ADDI    $8 2
351                         EXCH    $9 $8
352                         ADDI    $8 -2
353                         SUB     $8 $3
354 cmp_top_38:              BEQ     $9 $0
        cmp_bot_39
355                         XORI    $10 1
356 cmp_bot_39:             BEQ     $9 $0
        cmp_top_38
357 f_top_40:               BEQ     $10 $0
        f_bot_41
358                         XORI    $11 1
359 f_bot_41:               BEQ     $10 $0
        f_top_40
360                         XOR     $7 $11
361 f_bot_41_i:             BEQ     $10 $0
        f_top_40_i
362                         XORI    $11 1
363 f_top_40_i:             BEQ     $10 $0
        f_bot_41_i
364 cmp_bot_39_i:           BEQ     $9 $0
        cmp_top_38_i
365                         XORI    $10 1
366 cmp_top_38_i:           BEQ     $9 $0
        cmp_bot_39_i
367                         ADD     $8 $3
368                         ADDI    $8 2
369                         EXCH    $9 $8
370                         ADDI    $8 -2
371                         SUB     $8 $3
372 test_34:                 BEQ     $7 $0
        test_false_36
373                         XORI    $7 1
374                         ADD     $8 $3
375                         ADDI    $8 2
376                         EXCH    $9 $8
377                         ADDI    $8 -2
378                         SUB     $8 $3
379                         XOR     $10 $9
380 loadMetAdd_42:          EXCH    $11 $10
381                         ADDI    $11 1
382                         EXCH    $12 $11
383                         XOR     $13 $12
384                         EXCH    $12 $11
385                         ADDI    $11 -1
386                         EXCH    $11 $10
387                         ADD     $8 $3
388                         ADDI    $8 2
389                         EXCH    $9 $8
390                         ADDI    $8 -2
391                         SUB     $8 $3
392                         EXCH    $3 $1
393                         ADDI    $1 -1
394                         EXCH    $6 $1
395                         ADDI    $1 -1
396                         EXCH    $10 $1
397                         ADDI    $1 -1
398                         ADDI    $13 -397
399 l_jmp_43:               SWAPBR  $13
400                         NEG     $13
401                         ADDI    $13 397
402                         ADDI    $1 1
403                         EXCH    $10 $1
404                         ADDI    $1 1
405                         EXCH    $6 $1
406                         ADDI    $1 1
407                         EXCH    $3 $1
408                         ADD     $8 $3
409                         ADDI    $8 2
410                         EXCH    $9 $8
411                         ADDI    $8 -2
412                         SUB     $8 $3
413                         EXCH    $11 $10
414                         ADDI    $11 1
415                         EXCH    $12 $11
416                         XOR     $13 $12
417                         EXCH    $12 $11
418                         ADDI    $11 -1
419 loadMetAdd_42_i:        EXCH    $11 $10
420                         XOR     $10 $9
421                         ADD     $8 $3
422                         ADDI    $8 2
423                         EXCH    $9 $8
424                         ADDI    $8 -2
425                         SUB     $8 $3
426                         XORI    $7 1
427 assert_true_35:         BRA     assert_37
428 test_false_36:           BRA     test_34
429 assert_37:              BNE     $7 $0
        assert_true_35
430                         ADD     $8 $3
431                         ADDI    $8 2
432                         EXCH    $9 $8
433                         ADDI    $8 -2
434                         SUB     $8 $3
435 cmp_top_44:              BEQ     $9 $0
        cmp_bot_45
436                         XORI    $10 1
437 cmp_bot_45:             BEQ     $9 $0
        cmp_top_44
438 f_top_46:               BEQ     $10 $0
        f_bot_47
439                         XORI    $11 1
440 f_bot_47:               BEQ     $10 $0
        f_top_46
441                         XOR     $7 $11
442 f_bot_47_i:             BEQ     $10 $0
        f_top_46_i
443                         XORI    $11 1
444 f_top_46_i:             BEQ     $10 $0
        f_bot_47_i
445 cmp_bot_45_i:           BEQ     $9 $0
        cmp_top_44_i
446                         XORI    $10 1
447 cmp_top_44_i:           BEQ     $9 $0
        cmp_bot_45_i
448                         ADD     $8 $3
449                         ADDI    $8 2
450                         EXCH    $9 $8
451                         ADDI    $8 -2
452                         SUB     $8 $3
453 l_append_6_bot:          BRA     l_append_6_top
454 l_insertHead_1_top:      BRA
```

```
    l_insertHead_1_bot
455                         ADDI   $1 1
456                         EXCH   $2 $1
457                         EXCH   $6 $1
458                         ADDI   $1 -1
459                         EXCH   $3 $1
460                         ADDI   $1 -1
461 l_insertHead_1:          SWAPBR $2
462                         NEG    $2
463                         ADDI   $1 1
464                         EXCH   $3 $1
465                         ADDI   $1 1
466                         EXCH   $6 $1
467                         EXCH   $2 $1
468                         ADDI   $1 -1
469                         ADD    $8 $3
470                         ADDI   $8 2
471                         EXCH   $9 $8
472                         ADDI   $8 -2
473                         SUB    $8 $3
474 cmp_top_52:             BNE    $9 $0
    cmp_bot_53
475                         XORI   $10 1
476 cmp_bot_53:             BNE    $9 $0
    cmp_top_52
477                         EXCH   $11 $6
478 cmp_top_54:             BEQ    $11 $0
    cmp_bot_55
479                         XORI   $12 1
480 cmp_bot_55:             BEQ    $11 $0
    cmp_top_54
481                         ANDX   $13 $10 $12
482 f_top_56:               BEQ    $13 $0
    f_bot_57
483                         XORI   $14 1
484 f_bot_57:               BEQ    $13 $0
    f_top_56
485                         XOR    $7 $14
486 f_bot_57_i:             BEQ    $13 $0
    f_top_56_i
487                         XORI   $14 1
488 f_top_56_i:             BEQ    $13 $0
    f_bot_57_i
489                         ANDX   $13 $10 $12
490 cmp_bot_55_i:           BEQ    $11 $0
    cmp_top_54_i
491                         XORI   $12 1
492 cmp_top_54_i:           BEQ    $11 $0
    cmp_bot_55_i
493                         EXCH   $11 $6
494 cmp_bot_53_i:           BNE    $9 $0
    cmp_top_52_i
495                         XORI   $10 1
496 cmp_top_52_i:           BNE    $9 $0
    cmp_bot_53_i
497                         ADD    $8 $3
498                         ADDI   $8 2
499                         EXCH   $9 $8
500                         ADDI   $8 -2
501                         SUB    $8 $3
502 test_48:                 BEQ    $7 $0
    test_false_50
503                         XORI   $7 1
504                         ADD    $8 $3
505                         ADDI   $8 2
506                         EXCH   $9 $8
507                         ADDI   $8 -2
508                         SUB    $8 $3
509                         EXCH   $10 $6
510 swap_58:                XOR    $9 $10
511                         XOR    $10 $9
512                         XOR    $9 $10
513                         EXCH   $10 $6
514                         ADD    $8 $3
515                         ADDI   $8 2
516                         EXCH   $9 $8
517                         ADDI   $8 -2
518                         SUB    $8 $3
519                         XORI   $7 1
520 assert_true_49:         BRA    assert_51
521 test_false_50:          BRA    test_48
522 assert_51:              BNE    $7 $0
    assert_true_49
523                         ADD    $8 $3
524                         ADDI   $8 2
525                         EXCH   $9 $8
526                         ADDI   $8 -2
527                         SUB    $8 $3
528 cmp_top_59:             BEQ    $9 $0
    cmp_bot_60
529                         XORI   $10 1
530 cmp_bot_60:             BEQ    $9 $0
    cmp_top_59
531                         EXCH   $11 $6
532 cmp_top_61:             BNE    $11 $0
    cmp_bot_62
533                         XORI   $12 1
534 cmp_bot_62:             BNE    $11 $0
    cmp_top_61
535                         ANDX   $13 $10 $12
536 f_top_63:               BEQ    $13 $0
    f_bot_64
537                         XORI   $14 1
538 f_bot_64:               BEQ    $13 $0
    f_top_63
539                         XOR    $7 $14
540 f_bot_64_i:             BEQ    $13 $0
    f_top_63_i
541                         XORI   $14 1
542 f_top_63_i:             BEQ    $13 $0
    f_bot_64_i
543                         ANDX   $13 $10 $12
544 cmp_bot_62_i:           BNE    $11 $0
    cmp_top_61_i
545                         XORI   $12 1
546 cmp_top_61_i:           BNE    $11 $0
    cmp_bot_62_i
547                         EXCH   $11 $6
548 cmp_bot_60_i:           BEQ    $9 $0
    cmp_top_59_i
549                         XORI   $10 1
550 cmp_top_59_i:           BEQ    $9 $0
    cmp_bot_60_i
551                         ADD    $8 $3
552                         ADDI   $8 2
553                         EXCH   $9 $8
554                         ADDI   $8 -2
555                         SUB    $8 $3
556 l_insertHead_1_bot:     BRA
    l_insertHead_1_top
557 l_appendCell_2_top:     BRA
    l_appendCell_2_bot
```

```
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $3 $1
                        ADDI    $1 -1
l_appendCell_2:         SWAPBR  $2
                        NEG     $2
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $3 $1
                        ADDI    $1 -1
                        BRA     l_insertHead_6
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
cmp_top_69:             BEQ     $9 $0
    cmp_bot_70
                        XORI    $10 1
cmp_bot_70:             BEQ     $9 $0
    cmp_top_69
f_top_71:               BEQ     $10 $0
    f_bot_72
                        XORI    $11 1
f_bot_72:               BEQ     $10 $0
    f_top_71
                        XOR     $7 $11
f_bot_72_i:             BEQ     $10 $0
    f_top_71_i
                        XORI    $11 1
f_top_71_i:             BEQ     $10 $0
    f_bot_72_i
cmp_bot_70_i:           BEQ     $9 $0
    cmp_top_69_i
                        XORI    $10 1
cmp_top_69_i:           BEQ     $9 $0
    cmp_bot_70_i
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
test_65:                BEQ     $7 $0
    test_false_67
                        XORI    $7 1
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
                        XOR     $10 $9
loadMetAdd_73:          EXCH    $11 $10
                        ADDI    $11 1
                        EXCH    $12 $11
                        XOR     $13 $12
                        EXCH    $12 $11
                        ADDI    $11 -1
                        EXCH    $11 $10
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        ADDI    $13 -629
l_jmp_74:               SWAPBR  $13
                        NEG     $13
                        ADDI    $13 629
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
                        EXCH    $11 $10
                        ADDI    $11 1
                        EXCH    $12 $11
                        XOR     $13 $12
                        EXCH    $12 $11
                        ADDI    $11 -1
loadMetAdd_73_i:        EXCH    $11 $10
                        XOR     $10 $9
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
                        XORI    $7 1
assert_true_66:         BRA     assert_68
test_false_67:          BRA     test_65
assert_68:              BNE     $7 $0
    assert_true_66
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
cmp_top_75:             BEQ     $9 $0
    cmp_bot_76
                        XORI    $10 1
cmp_bot_76:             BEQ     $9 $0
    cmp_top_75
f_top_77:               BEQ     $10 $0
    f_bot_78
                        XORI    $11 1
f_bot_78:               BEQ     $10 $0
    f_top_77
                        XOR     $7 $11
f_bot_78_i:             BEQ     $10 $0
    f_top_77_i
```

```
675                         XORI    $11 1
676 f_top_77_i:             BEQ     $10 $0
        f_bot_78_i
677 cmp_bot_76_i:           BEQ     $9 $0
        cmp_top_75_i
678                         XORI    $10 1
679 cmp_top_75_i:           BEQ     $9 $0
        cmp_bot_76_i
680                         ADD     $8 $3
681                         ADDI    $8 2
682                         EXCH    $9 $8
683                         ADDI    $8 -2
684                         SUB     $8 $3
685                         ADD     $7 $3
686                         ADDI    $7 3
687                         EXCH    $8 $7
688                         ADDI    $7 -3
689                         SUB     $7 $3
690                         XORI    $9 1
691                         ADD     $8 $9
692                         XORI    $9 1
693                         ADD     $7 $3
694                         ADDI    $7 3
695                         EXCH    $8 $7
696                         ADDI    $7 -3
697                         SUB     $7 $3
698 l_appendCell_2_bot:     BRA
        l_appendCell_2_top
699 l_prependCell_3_top:    BRA
        l_prependCell_3_bot
700                         ADDI    $1 1
701                         EXCH    $2 $1
702                         EXCH    $6 $1
703                         ADDI    $1 -1
704                         EXCH    $3 $1
705                         ADDI    $1 -1
706 l_prependCell_3:         SWAPBR  $2
707                         NEG     $2
708                         ADDI    $1 1
709                         EXCH    $3 $1
710                         ADDI    $1 1
711                         EXCH    $6 $1
712                         EXCH    $2 $1
713                         ADDI    $1 -1
714                         EXCH    $6 $1
715                         ADDI    $1 -1
716                         EXCH    $3 $1
717                         ADDI    $1 -1
718                         BRA     l_insertHead_71
719                         ADDI    $1 1
720                         EXCH    $3 $1
721                         ADDI    $1 1
722                         EXCH    $6 $1
723                         EXCH    $8 $6
724 cmp_top_83:             BEQ     $8 $0
        cmp_bot_84
725                         XORI    $9 1
726 cmp_bot_84:             BEQ     $8 $0
        cmp_top_83
727                         ADD     $10 $3
728                         ADDI    $10 2
729                         EXCH    $11 $10
730                         ADDI    $10 -2
731                         SUB     $10 $3
732 cmp_top_85:             BEQ     $11 $0
        cmp_bot_86
733                         XORI    $12 1
734 cmp_bot_86:             BEQ     $11 $0
        cmp_top_85
735                         ANDX    $13 $9 $12
736 f_top_87:               BEQ     $13 $0
        f_bot_88
737                         XORI    $14 1
738 f_bot_88:               BEQ     $13 $0
        f_top_87
739                         XOR     $7 $14
740 f_bot_88_i:             BEQ     $13 $0
        f_top_87_i
741                         XORI    $14 1
742 f_top_87_i:             BEQ     $13 $0
        f_bot_88_i
743                         ANDX    $13 $9 $12
744 cmp_bot_86_i:           BEQ     $11 $0
        cmp_top_85_i
745                         XORI    $12 1
746 cmp_top_85_i:           BEQ     $11 $0
        cmp_bot_86_i
747                         ADD     $10 $3
748                         ADDI    $10 2
749                         EXCH    $11 $10
750                         ADDI    $10 -2
751                         SUB     $10 $3
752 cmp_bot_84_i:           BEQ     $8 $0
        cmp_top_83_i
753                         XORI    $9 1
754 cmp_top_83_i:           BEQ     $8 $0
        cmp_bot_84_i
755                         EXCH    $8 $6
756 test_79:                BEQ     $7 $0
        test_false_81
757                         XORI    $7 1
758                         EXCH    $8 $6
759                         XOR     $9 $8
760 loadMetAdd_89:          EXCH    $10 $9
761                         ADDI    $10 1
762                         EXCH    $11 $10
763                         XOR     $12 $11
764                         EXCH    $11 $10
765                         ADDI    $10 -1
766                         EXCH    $10 $9
767                         EXCH    $8 $6
768                         ADD     $13 $3
769                         ADDI    $13 2
770                         EXCH    $3 $1
771                         ADDI    $1 -1
772                         EXCH    $6 $1
773                         ADDI    $1 -1
774                         EXCH    $13 $1
775                         ADDI    $1 -1
776                         EXCH    $9 $1
777                         ADDI    $1 -1
778                         ADDI    $12 -777
779 l_jmp_90:               SWAPBR  $12
780                         NEG     $12
781                         ADDI    $12 777
782                         ADDI    $1 1
783                         EXCH    $9 $1
784                         ADDI    $1 1
785                         EXCH    $13 $1
786                         ADDI    $1 1
787                         EXCH    $6 $1
788                         ADDI    $1 1
```

```
789                             EXCH    $3 $1
790                             ADDI    $13 -2
791                             SUB     $13 $3
792                             EXCH    $8 $6
793                             EXCH    $10 $9
794                             ADDI    $10 1
795                             EXCH    $11 $10
796                             XOR     $12 $11
797                             EXCH    $11 $10
798                             ADDI    $10 -1
799 loadMetAdd_89_i:             EXCH    $10 $9
800                             XOR     $9 $8
801                             EXCH    $8 $6
802                             XORI    $7 1
803 assert_true_80:              BRA     assert_82
804 test_false_81:               BRA     test_79
805 assert_82:                   BNE     $7 $0
        assert_true_80
806                             EXCH    $8 $6
807 cmp_top_91:                  BEQ     $8 $0
        cmp_bot_92
808                             XORI    $9 1
809 cmp_bot_92:                  BEQ     $8 $0
        cmp_top_91
810                             ADD     $10 $3
811                             ADDI    $10 2
812                             EXCH    $11 $10
813                             ADDI    $10 -2
814                             SUB     $10 $3
815 cmp_top_93:                  BNE     $11 $0
        cmp_bot_94
816                             XORI    $12 1
817 cmp_bot_94:                  BNE     $11 $0
        cmp_top_93
818                             ANDX    $13 $9 $12
819 f_top_95:                   BEQ     $13 $0
        f_bot_96
820                             XORI    $14 1
821 f_bot_96:                   BEQ     $13 $0
        f_top_95
822                             XOR     $7 $14
823 f_bot_96_i:                 BEQ     $13 $0
        f_top_95_i
824                             XORI    $14 1
825 f_top_95_i:                 BEQ     $13 $0
        f_bot_96_i
826                             ANDX    $13 $9 $12
827 cmp_bot_94_i:               BNE     $11 $0
        cmp_top_93_i
828                             XORI    $12 1
829 cmp_top_93_i:               BNE     $11 $0
        cmp_bot_94_i
830                             ADD     $10 $3
831                             ADDI    $10 2
832                             EXCH    $11 $10
833                             ADDI    $10 -2
834                             SUB     $10 $3
835 cmp_bot_92_i:               BEQ     $8 $0
        cmp_top_91_i
836                             XORI    $9 1
837 cmp_top_91_i:               BEQ     $8 $0
        cmp_bot_92_i
838                             EXCH    $8 $6
839                             EXCH    $8 $6
840 cmp_top_101:                BEQ     $8 $0
        cmp_bot_102
841                             XORI    $9 1
842 cmp_bot_102:                BEQ     $8 $0
        cmp_top_101
843                             ADD     $10 $3
844                             ADDI    $10 2
845                             EXCH    $11 $10
846                             ADDI    $10 -2
847                             SUB     $10 $3
848 cmp_top_103:                BNE     $11 $0
        cmp_bot_104
849                             XORI    $12 1
850 cmp_bot_104:                BNE     $11 $0
        cmp_top_103
851                             ANDX    $13 $9 $12
852 f_top_105:                  BEQ     $13 $0
        f_bot_106
853                             XORI    $14 1
854 f_bot_106:                  BEQ     $13 $0
        f_top_105
855                             XOR     $7 $14
856 f_bot_106_i:                BEQ     $13 $0
        f_top_105_i
857                             XORI    $14 1
858 f_top_105_i:                BEQ     $13 $0
        f_bot_106_i
859                             ANDX    $13 $9 $12
860 cmp_bot_104_i:              BNE     $11 $0
        cmp_top_103_i
861                             XORI    $12 1
862 cmp_top_103_i:              BNE     $11 $0
        cmp_bot_104_i
863                             ADD     $10 $3
864                             ADDI    $10 2
865                             EXCH    $11 $10
866                             ADDI    $10 -2
867                             SUB     $10 $3
868 cmp_bot_102_i:              BEQ     $8 $0
        cmp_top_101_i
869                             XORI    $9 1
870 cmp_top_101_i:              BEQ     $8 $0
        cmp_bot_102_i
871                             EXCH    $8 $6
872 test_97:                    BEQ     $7 $0
        test_false_99
873                             XORI    $7 1
874                             EXCH    $8 $6
875                             ADD     $9 $3
876                             ADDI    $9 2
877                             EXCH    $10 $9
878                             ADDI    $9 -2
879                             SUB     $9 $3
880 swap_107:                    XOR     $8 $10
881                             XOR     $10 $8
882                             XOR     $8 $10
883                             ADD     $9 $3
884                             ADDI    $9 2
885                             EXCH    $10 $9
886                             ADDI    $9 -2
887                             SUB     $9 $3
888                             EXCH    $8 $6
889                             XORI    $7 1
890 assert_true_98:              BRA     assert_100
891 test_false_99:               BRA     test_97
892 assert_100:                  BNE     $7 $0
        assert_true_98
893                             EXCH    $8 $6
```

```
cmp_top_108:            BNE    $8 $0
    cmp_bot_109
                        XORI   $9 1
cmp_bot_109:            BNE    $8 $0
    cmp_top_108
                        ADD    $10 $3
                        ADDI   $10 2
                        EXCH   $11 $10
                        ADDI   $10 -2
                        SUB    $10 $3
cmp_top_110:            BEQ    $11 $0
    cmp_bot_111
                        XORI   $12 1
cmp_bot_111:            BEQ    $11 $0
    cmp_top_110
                        ANDX   $13 $9 $12
f_top_112:              BEQ    $13 $0
    f_bot_113
                        XORI   $14 1
f_bot_113:              BEQ    $13 $0
    f_top_112
                        XOR    $7 $14
f_bot_113_i:            BEQ    $13 $0
    f_top_112_i
                        XORI   $14 1
f_top_112_i:            BEQ    $13 $0
    f_bot_113_i
                        ANDX   $13 $9 $12
cmp_bot_111_i:          BEQ    $11 $0
    cmp_top_110_i
                        XORI   $12 1
cmp_top_110_i:          BEQ    $11 $0
    cmp_bot_111_i
                        ADD    $10 $3
                        ADDI   $10 2
                        EXCH   $11 $10
                        ADDI   $10 -2
                        SUB    $10 $3
cmp_bot_109_i:          BNE    $8 $0
    cmp_top_108_i
                        XORI   $9 1
cmp_top_108_i:          BNE    $8 $0
    cmp_bot_109_i
                        EXCH   $8 $6
                        ADD    $7 $3
                        ADDI   $7 3
                        EXCH   $8 $7
                        ADDI   $7 -3
                        SUB    $7 $3
                        XORI   $9 1
                        ADD    $8 $9
                        XORI   $9 1
                        ADD    $7 $3
                        ADDI   $7 3
                        EXCH   $8 $7
                        ADDI   $7 -3
                        SUB    $7 $3
l_prependCell_3_bot:    BRA
    l_prependCell_3_top
l_length_4_top:         BRA    l_length_4_bot
                        ADDI   $1 1
                        EXCH   $2 $1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $3 $1
                        ADDI   $1 -1
l_length_4:             SWAPBR $2
                        NEG    $2
                        ADDI   $1 1
                        EXCH   $3 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        EXCH   $2 $1
                        ADDI   $1 -1
                        EXCH   $7 $6
                        ADD    $8 $3
                        ADDI   $8 3
                        EXCH   $9 $8
                        ADDI   $8 -3
                        SUB    $8 $3
                        XOR    $7 $9
                        ADD    $8 $3
                        ADDI   $8 3
                        EXCH   $9 $8
                        ADDI   $8 -3
                        SUB    $8 $3
                        EXCH   $7 $6
l_length_4_bot:         BRA    l_length_4_top
l_main_0_top:           BRA    l_main_0_bot
                        ADDI   $1 1
                        EXCH   $2 $1
                        EXCH   $3 $1
                        ADDI   $1 -1
l_main_0:               SWAPBR $2
                        NEG    $2
                        ADDI   $1 1
                        EXCH   $3 $1
                        EXCH   $2 $1
                        ADDI   $1 -1
                        EXCH   $3 $1
                        ADDI   $1 -1
obj_con_114:            ADDI   $8 4
                        EXCH   $8 $1
                        ADDI   $1 -1
                        EXCH   $7 $1
                        ADDI   $1 -1
                        BRA    l_malloc
                        ADDI   $1 1
                        EXCH   $7 $1
                        ADDI   $1 1
                        EXCH   $8 $1
obj_con_114_i:          ADDI   $8 -4
                        ADDI   $1 1
                        EXCH   $3 $1
                        ADD    $6 $3
                        ADDI   $6 2
                        XORI   $8 5
                        EXCH   $8 $7
                        ADDI   $7 1
                        XORI   $8 1
                        EXCH   $8 $7
obj_con_114_bot:        ADDI   $7 -1
                        EXCH   $7 $6
                        ADDI   $6 -2
                        SUB    $6 $3
                        ADD    $6 $3
                        ADDI   $6 4
                        EXCH   $7 $6
                        ADDI   $6 -4
                        SUB    $6 $3
                        XORI   $8 10
                        ADD    $7 $8
```

```
                        XORI    $8 10
                        ADD     $6 $3
                        ADDI    $6 4
                        EXCH    $7 $6
                        ADDI    $6 -4
                        SUB     $6 $3
localBlock_133:         XOR     $6 $1
                        XOR     $7 $0
                        EXCH    $7 $1
                        ADDI    $1 -1
                        XORI    $7 1
entry_115:              BEQ     $7 $0
    assert_117
                        EXCH    $8 $6
cmp_top_119:            BNE     $8 $0
    cmp_bot_120
                        XORI    $9 1
cmp_bot_120:            BNE     $8 $0
    cmp_top_119
f_top_121:              BEQ     $9 $0
    f_bot_122
                        XORI    $10 1
f_bot_122:              BEQ     $9 $0
    f_top_121
                        XOR     $7 $10
f_bot_122_i:            BEQ     $9 $0
    f_top_121_i
                        XORI    $10 1
f_top_121_i:            BEQ     $9 $0
    f_bot_122_i
cmp_bot_120_i:          BNE     $8 $0
    cmp_top_119_i
                        XORI    $9 1
cmp_top_119_i:          BNE     $8 $0
    cmp_bot_120_i
                        EXCH    $8 $6
                        EXCH    $8 $6
                        ADD     $9 $3
                        ADDI    $9 4
                        EXCH    $10 $9
                        ADDI    $9 -4
                        SUB     $9 $3
cmp_top_129:            BNE     $8 $10
    cmp_bot_130
                        XORI    $11 1
cmp_bot_130:            BNE     $8 $10
    cmp_top_129
f_top_131:              BEQ     $11 $0
    f_bot_132
                        XORI    $12 1
f_bot_132:              BEQ     $11 $0
    f_top_131
                        XOR     $7 $12
f_bot_132_i:            BEQ     $11 $0
    f_top_131_i
                        XORI    $12 1
f_top_131_i:            BEQ     $11 $0
    f_bot_132_i
cmp_bot_130_i:          BNE     $8 $10
    cmp_top_129_i
                        XORI    $11 1
cmp_top_129_i:          BNE     $8 $10
    cmp_bot_130_i
                        ADD     $9 $3
                        ADDI    $9 4
                        EXCH    $10 $9
                        ADDI    $9 -4
                        SUB     $9 $3
                        EXCH    $8 $6
test_116:               BNE     $7 $0 exit_118
localBlock_128:         XOR     $8 $1
                        XOR     $9 $0
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
obj_con_123:            ADDI    $10 4
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        BRA     l_malloc
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
obj_con_123_i:          ADDI    $10 -4
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        XORI    $10 9
                        EXCH    $10 $9
                        ADDI    $9 1
                        XORI    $10 1
                        EXCH    $10 $9
obj_con_123_bot:        ADDI    $9 -1
                        EXCH    $9 $8
                        EXCH    $9 $8
                        XOR     $10 $9
loadMetAdd_124:         EXCH    $11 $10
                        ADDI    $11 0
                        EXCH    $12 $11
                        XOR     $13 $12
                        EXCH    $12 $11
                        ADDI    $11 0
                        EXCH    $11 $10
                        EXCH    $9 $8
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        ADDI    $13 -1117
l_jmp_125:              SWAPBR  $13
                        NEG     $13
                        ADDI    $13 1117
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $8 $1
```

```
                        ADDI   $1 1
                        EXCH   $3 $1
                        EXCH   $9 $8
                        EXCH   $11 $10
                        ADDI   $11 0
                        EXCH   $12 $11
                        XOR    $13 $12
                        EXCH   $12 $11
                        ADDI   $11 0
loadMetAdd_124_i:       EXCH   $11 $10
                        XOR    $10 $9
                        EXCH   $9 $8
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
                        XOR    $11 $10
loadMetAdd_126:         EXCH   $12 $11
                        ADDI   $12 1
                        EXCH   $13 $12
                        XOR    $14 $13
                        EXCH   $13 $12
                        ADDI   $12 -1
                        EXCH   $12 $11
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
                        EXCH   $3 $1
                        ADDI   $1 -1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $8 $1
                        ADDI   $1 -1
                        EXCH   $11 $1
                        ADDI   $1 -1
                        ADDI   $14 -1165
l_jmp_127:              SWAPBR $14
                        NEG    $14
                        ADDI   $14 1165
                        ADDI   $1 1
                        EXCH   $11 $1
                        ADDI   $1 1
                        EXCH   $8 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
                        EXCH   $12 $11
                        ADDI   $12 1
                        EXCH   $13 $12
                        XOR    $14 $13
                        EXCH   $13 $12
                        ADDI   $12 -1
loadMetAdd_126_i:       EXCH   $12 $11
                        XOR    $11 $10
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
                        ADDI   $1 1
                        EXCH   $9 $1
                        XOR    $9 $0
localBlock_128_i:       XOR    $8 $1
                        EXCH   $8 $6
                        XORI   $9 1
                        ADD    $8 $9
                        XORI   $9 1
                        EXCH   $8 $6
assert_117:             BRA    entry_115
exit_118:               BRA    test_116
                        XORI   $7 1
                        ADD    $7 $3
                        ADDI   $7 4
                        EXCH   $8 $7
                        ADDI   $7 -4
                        SUB    $7 $3
                        ADDI   $1 1
                        EXCH   $9 $1
                        XOR    $9 $8
localBlock_133_i:       XOR    $6 $1
                        ADD    $7 $3
                        ADDI   $7 4
                        EXCH   $8 $7
                        ADDI   $7 -4
                        SUB    $7 $3
l_main_0_bot:           BRA    l_main_0_top
start:                  BRA    top
                        START
                        ADDI   $4 1279
                        XOR    $5 $4
                        ADDI   $5 10
                        XOR    $7 $5
                        ADDI   $4 10
                        ADDI   $4 -1
                        EXCH   $7 $4
                        ADDI   $4 1
                        ADDI   $4 -10
                        XOR    $1 $5
                        ADDI   $1 2048
                        ADDI   $1 -8
                        XOR    $3 $1
                        XORI   $6 4
                        EXCH   $6 $3
                        ADDI   $1 -1
                        EXCH   $3 $1
                        ADDI   $1 -1
                        BRA    l_main_0
                        ADDI   $1 1
                        EXCH   $3 $1
                        ADDI   $3 1
                        ADDI   $3 1
                        EXCH   $6 $3
                        XORI   $7 1
                        EXCH   $6 $7
                        XORI   $7 1
                        ADDI   $3 -1
                        ADDI   $3 -1
                        ADDI   $3 1
                        ADDI   $3 2
                        EXCH   $6 $3
                        XORI   $7 2
                        EXCH   $6 $7
                        XORI   $7 2
```

```
1260                    ADDI    $3 -2
1261                    ADDI    $3 -1
1262                    ADDI    $3 1
1263                    ADDI    $3 3
1264                    EXCH    $6 $3
1265                    XORI    $7 3
1266                    EXCH    $6 $7
1267                    XORI    $7 3
1268                    ADDI    $3 -3
1269                    ADDI    $3 -1
1270                    ADDI    $1 1
1271                    EXCH    $6 $3
1272                    XORI    $6 4
1273                    XOR     $3 $1
1274                    ADDI    $1 8
1275                    ADDI    $1 -2048
1276                    XOR     $1 $5
1277                    ADDI    $5 -10
1278                    XOR     $5 $4
1279                    ADDI    $4 -1279
1280 finish:            FINISH
```

BinaryTree.rplpp

```
class Node
    Node left
    Node right
    int value

    method setValue(int newValue)
        value ^= newValue

    method insertNode(Node node, int nodeValue)
        if nodeValue < value then                          // Determine if we insert left or
            right
            if left = nil & node != nil then
                left <=> node                              // If open left node, store here
            else skip
            fi left != nil & node = nil

            if left != nil then
                call left::insertNode(node, nodeValue)     // If current node has left, continue
                    iterating
            else skip
            fi left != nil
        else
            if right = nil & node != nil then
                right <=> node                             // If open right node spot, store here
            else skip
            fi right != nil & node = nil

            if right != nil then
                call right::insertNode(node, nodeValue) // If current node has, continue
                    searching
            else skip
            fi right != nil
        fi nodeValue < value

    method getSum(int result)
        result += value                  // Add the value of this node to the sum

        if left != nil then
            call left::getSum(result)    // If we have a left child, follow that path
        else skip                        // Else, skip
        fi left != nil

        if right != nil then
            call right::getSum(result)   // If we have a right child, follow that path
        else skip                        // Else, skip
         fi right != nil

    method mirror()
        left <=> right                   // Swap left and right children

        if left = nil then skip
        else call left::mirror()         // Recursively swap children if left != nil
        fi left = nil

        if right = nil then skip
        else call right::mirror()        // Recursively swap children if right != nil
        fi right = nil

class Tree
    Node root

    method insertNode(Node node, int value)
        if root = nil & node != nil then
```

```
61              root <=> node
62          else skip
63          fi root != nil & node = nil
64
65          if root != nil then
66              call root::insertNode(node, value)
67          else skip
68          fi root != nil
69
70      method sum(int result)
71          if root != nil then
72              call root::getSum(result)
73          else skip
74          fi root != nil
75
76      method mirror()
77          if root != nil then
78              call root::mirror()
79          else skip
80          fi root != nil
81
82  class Program
83      int sumResult
84      Tree tree
85      int nodeCount
86
87      method main()
88          new Tree tree
89          nodeCount += 3
90
91          local int x = 0
92          from x = 0 do
93              skip
94          loop
95              local Node node = nil
96              new Node node                    // Init new node
97              call node::setValue(x)           // Set node value
98              call tree::insertNode(node, x)  // Insert node in tree
99              delocal Node node = nil
100             x += 1
101         until x = nodeCount
102         delocal int x = nodeCount
103
104         call tree::sum(sumResult)
105         call tree::mirror()
```

BinaryTree.pal

```
;; pendulum pal file
top:                BRA    start
l_r_sumResult:      DATA   0
l_r_tree:           DATA   0
l_r_nodeCount:      DATA   0
l_Program_vt:       DATA   1644
l_Tree_vt:          DATA   1201
                    DATA   1414
                    DATA   1532
l_Node_vt:          DATA   224
                    DATA   255
                    DATA   727
                    DATA   962
l_malloc_top:       BRA    l_malloc_bot
l_malloc:           SWAPBR $2
                    NEG    $2
                    ADDI   $9 2
                    XOR    $8 $0
                    ADDI   $1 1
                    EXCH   $6 $1
                    ADDI   $1 1
                    EXCH   $7 $1
                    EXCH   $2 $1
                    ADDI   $1 -1
                    BRA    l_malloc1
                    ADDI   $1 1
                    EXCH   $2 $1
                    EXCH   $7 $1
                    ADDI   $1 -1
                    EXCH   $6 $1
                    ADDI   $1 -1
                    XOR    $8 $0
                    ADDI   $9 -2
l_malloc_bot:       BRA    l_malloc_top
l_malloc1_top:      BRA    l_malloc1_bot
                    ADDI   $1 1
                    EXCH   $2 $1
                    SUB    $17 $8
                    XOR    $17 $4
l_malloc1:          SWAPBR $2
                    NEG    $2
                    EXCH   $2 $1
                    ADDI   $1 -1
                    XOR    $17 $4
                    ADD    $17 $8
                    EXCH   $19 $17
                    XOR    $18 $19
                    EXCH   $19 $17
                    XOR    $13 $9
                    SUB    $13 $7
cmp_top_8:          BGEZ   $13 cmp_bot_9
                    XORI   $14 1
cmp_bot_9:          BGEZ   $13 cmp_top_8
                    XOR    $10 $14
cmp_bot_9_i:        BGEZ   $13
    cmp_top_8_i
                    XORI   $14 1
cmp_top_8_i:        BGEZ   $13
    cmp_bot_9_i
                    ADD    $13 $7
                    XOR    $13 $9
l_o_test:           BEQ    $10 $0
    l_o_test_false
                    XORI   $10 1
                    ADDI   $8 1
                    EXCH   $19 $17
                    XOR    $18 $19
                    EXCH   $19 $17
                    RL     $9 1
                    EXCH   $10 $1
                    ADDI   $1 -1
                    EXCH   $11 $1
                    ADDI   $1 -1
                    EXCH   $12 $1
                    ADDI   $1 -1
                    EXCH   $14 $1
                    ADDI   $1 -1
                    EXCH   $16 $1
                    ADDI   $1 -1
                    EXCH   $17 $1
                    ADDI   $1 -1
                    EXCH   $18 $1
                    ADDI   $1 -1
                    EXCH   $20 $1
                    ADDI   $1 -1
                    EXCH   $21 $1
                    ADDI   $1 -1
                    EXCH   $22 $1
                    ADDI   $1 -1
                    EXCH   $23 $1
                    ADDI   $1 -1
                    BRA    l_malloc1
                    ADDI   $1 1
                    EXCH   $23 $1
                    ADDI   $1 1
                    EXCH   $22 $1
                    ADDI   $1 1
                    EXCH   $21 $1
                    ADDI   $1 1
                    EXCH   $20 $1
                    ADDI   $1 1
                    EXCH   $18 $1
                    ADDI   $1 1
                    EXCH   $17 $1
                    ADDI   $1 1
                    EXCH   $16 $1
                    ADDI   $1 1
                    EXCH   $14 $1
                    ADDI   $1 1
                    EXCH   $12 $1
                    ADDI   $1 1
                    EXCH   $11 $1
                    ADDI   $1 1
                    EXCH   $10 $1
                    RR     $9 1
                    ADDI   $8 -1
                    XORI   $10 1
l_o_assert_true:    BRA    l_o_assert
l_o_test_false:     BRA    l_o_test
cmp_top_12:         BEQ    $18 $0
    cmp_bot_13
                    XORI   $20 1
cmp_bot_13:         BEQ    $18 $0
    cmp_top_12
```

```
120                             XOR     $11 $20
121 cmp_bot_13_i:               BEQ     $18 $0
        cmp_top_12_i
122                             XORI    $20 1
123 cmp_top_12_i:               BEQ     $18 $0
        cmp_bot_13_i
124 l_i_test:                   BEQ     $11 $0
        l_i_test_false
125                             XORI    $11 1
126                             ADD     $6 $18
127                             SUB     $18 $6
128                             EXCH    $12 $6
129                             EXCH    $12 $17
130                             XOR     $12 $6
131                             XORI    $11 1
132 l_i_assert_true:            BRA     l_i_assert
133 l_i_test_false:             BRA     l_i_test
134                             ADDI    $8 1
135                             RL      $9 1
136                             EXCH    $10 $1
137                             ADDI    $1 -1
138                             EXCH    $11 $1
139                             ADDI    $1 -1
140                             EXCH    $12 $1
141                             ADDI    $1 -1
142                             EXCH    $14 $1
143                             ADDI    $1 -1
144                             EXCH    $16 $1
145                             ADDI    $1 -1
146                             EXCH    $17 $1
147                             ADDI    $1 -1
148                             EXCH    $18 $1
149                             ADDI    $1 -1
150                             EXCH    $20 $1
151                             ADDI    $1 -1
152                             EXCH    $21 $1
153                             ADDI    $1 -1
154                             EXCH    $22 $1
155                             ADDI    $1 -1
156                             EXCH    $23 $1
157                             ADDI    $1 -1
158                             BRA     l_malloc1
159                             ADDI    $1 1
160                             EXCH    $23 $1
161                             ADDI    $1 1
162                             EXCH    $22 $1
163                             ADDI    $1 1
164                             EXCH    $21 $1
165                             ADDI    $1 1
166                             EXCH    $20 $1
167                             ADDI    $1 1
168                             EXCH    $18 $1
169                             ADDI    $1 1
170                             EXCH    $17 $1
171                             ADDI    $1 1
172                             EXCH    $16 $1
173                             ADDI    $1 1
174                             EXCH    $14 $1
175                             ADDI    $1 1
176                             EXCH    $12 $1
177                             ADDI    $1 1
178                             EXCH    $11 $1
179                             ADDI    $1 1
180                             EXCH    $10 $1
181                             RR      $9 1
182                             ADDI    $8 -1
183                             XOR     $12 $6
184                             EXCH    $12 $17
185                             ADD     $6 $9
186 l_i_assert:                 BNE     $11 $0
        l_i_assert_true
187                             EXCH    $12 $17
188                             SUB     $6 $9
189 cmp_top_14:                 BEQ     $6 $12
        cmp_bot_15
190                             XORI    $21 1
191 cmp_bot_15:                 BEQ     $6 $12
        cmp_top_14
192 cmp_top_16:                 BNE     $12 $0
        cmp_bot_17
193                             XORI    $22 1
194 cmp_bot_17:                 BNE     $12 $0
        cmp_top_16
195                             ORX     $23 $21 $22
196                             XOR     $11 $23
197                             ORX     $23 $21 $22
198 cmp_bot_17_i:               BNE     $12 $0
        cmp_top_16_i
199                             XORI    $22 1
200 cmp_top_16_i:               BNE     $12 $0
        cmp_bot_17_i
201 cmp_bot_15_i:               BEQ     $6 $12
        cmp_top_14_i
202                             XORI    $21 1
203 cmp_top_14_i:               BEQ     $6 $12
        cmp_bot_15_i
204                             ADD     $6 $9
205                             EXCH    $12 $17
206 l_o_assert:                 BNE     $10 $0
        l_o_assert_true
207                             XOR     $15 $9
208                             SUB     $15 $7
209 cmp_top_10:                 BGEZ    $15 cmp_bot_11
210                             XORI    $16 1
211 cmp_bot_11:                 BGEZ    $15 cmp_top_10
212                             XOR     $10 $16
213 cmp_bot_11_i:               BGEZ    $15
        cmp_top_10_i
214                             XORI    $16 1
215 cmp_top_10_i:               BGEZ    $15
        cmp_bot_11_i
216                             ADD     $15 $7
217                             XOR     $15 $9
218 l_malloc1_bot:              BRA     l_malloc1_top
219 l_setValue_4_top:           BRA
        l_setValue_4_bot
220                             ADDI    $1 1
221                             EXCH    $2 $1
222                             EXCH    $6 $1
223                             ADDI    $1 -1
224                             EXCH    $3 $1
225                             ADDI    $1 -1
226 l_setValue_4:               SWAPBR  $2
227                             NEG     $2
228                             ADDI    $1 1
229                             EXCH    $3 $1
230                             ADDI    $1 1
231                             EXCH    $6 $1
232                             EXCH    $2 $1
233                             ADDI    $1 -1
234                             ADD     $7 $3
235                             ADDI    $7 4
```

```
                        EXCH    $8 $7
                        ADDI    $7 -4
                        SUB     $7 $3
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADD     $7 $3
                        ADDI    $7 4
                        EXCH    $8 $7
                        ADDI    $7 -4
                        SUB     $7 $3
l_setValue_4_bot:       BRA
    l_setValue_4_top
l_insertNode_5_top:     BRA
    l_insertNode_5_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $3 $1
                        ADDI    $1 -1
l_insertNode_5:         SWAPBR  $2
                        NEG     $2
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
                        EXCH    $9 $7
                        ADD     $10 $3
                        ADDI    $10 4
                        EXCH    $11 $10
                        ADDI    $10 -4
                        SUB     $10 $3
                        XOR     $12 $9
                        SUB     $12 $11
cmp_top_22:             BGEZ    $12 cmp_bot_23
                        XORI    $13 1
cmp_bot_23:             BGEZ    $12 cmp_top_22
f_top_24:               BEQ     $13 $0
    f_bot_25
                        XORI    $14 1
f_bot_25:               BEQ     $13 $0
    f_top_24
                        XOR     $8 $14
f_bot_25_i:             BEQ     $13 $0
    f_top_24_i
                        XORI    $14 1
f_top_24_i:             BEQ     $13 $0
    f_bot_25_i
cmp_bot_23_i:           BGEZ    $12
    cmp_top_22_i
                        XORI    $13 1
cmp_top_22_i:           BGEZ    $12
    cmp_bot_23_i
                        ADD     $12 $11
                        XOR     $12 $9
                        ADD     $10 $3
                        ADDI    $10 4
                        EXCH    $11 $10
                        ADDI    $10 -4
                        SUB     $10 $3
                        EXCH    $9 $7
test_18:                BEQ     $8 $0
    test_false_20
                        XORI    $8 1
                        ADD     $10 $3
                        ADDI    $10 2
                        EXCH    $11 $10
                        ADDI    $10 -2
                        SUB     $10 $3
cmp_top_30:             BNE     $11 $0
    cmp_bot_31
                        XORI    $12 1
cmp_bot_31:             BNE     $11 $0
    cmp_top_30
                        EXCH    $13 $6
cmp_top_32:             BEQ     $13 $0
    cmp_bot_33
                        XORI    $14 1
cmp_bot_33:             BEQ     $13 $0
    cmp_top_32
                        ANDX    $15 $12 $14
f_top_34:               BEQ     $15 $0
    f_bot_35
                        XORI    $16 1
f_bot_35:               BEQ     $15 $0
    f_top_34
                        XOR     $9 $16
f_bot_35_i:             BEQ     $15 $0
    f_top_34_i
                        XORI    $16 1
f_top_34_i:             BEQ     $15 $0
    f_bot_35_i
                        ANDX    $15 $12 $14
cmp_bot_33_i:           BEQ     $13 $0
    cmp_top_32_i
                        XORI    $14 1
cmp_top_32_i:           BEQ     $13 $0
    cmp_bot_33_i
                        EXCH    $13 $6
cmp_bot_31_i:           BNE     $11 $0
    cmp_top_30_i
                        XORI    $12 1
cmp_top_30_i:           BNE     $11 $0
    cmp_bot_31_i
                        ADD     $10 $3
                        ADDI    $10 2
                        EXCH    $11 $10
                        ADDI    $10 -2
                        SUB     $10 $3
test_26:                BEQ     $9 $0
    test_false_28
                        XORI    $9 1
                        ADD     $10 $3
                        ADDI    $10 2
                        EXCH    $11 $10
                        ADDI    $10 -2
                        SUB     $10 $3
                        EXCH    $12 $6
swap_36:                XOR     $11 $12
                        XOR     $12 $11
                        XOR     $11 $12
                        EXCH    $12 $6
                        ADD     $10 $3
                        ADDI    $10 2
                        EXCH    $11 $10
```

```
346                             ADDI    $10 -2
347                             SUB     $10 $3
348                             XORI    $9 1
349 assert_true_27:             BRA     assert_29
350 test_false_28:              BRA     test_26
351 assert_29:                   BNE     $9 $0
        assert_true_27
352                             ADD     $10 $3
353                             ADDI    $10 2
354                             EXCH    $11 $10
355                             ADDI    $10 -2
356                             SUB     $10 $3
357 cmp_top_37:                 BEQ     $11 $0
        cmp_bot_38
358                             XORI    $12 1
359 cmp_bot_38:                 BEQ     $11 $0
        cmp_top_37
360                             EXCH    $13 $6
361 cmp_top_39:                 BNE     $13 $0
        cmp_bot_40
362                             XORI    $14 1
363 cmp_bot_40:                 BNE     $13 $0
        cmp_top_39
364                             ANDX    $15 $12 $14
365 f_top_41:                   BEQ     $15 $0
        f_bot_42
366                             XORI    $16 1
367 f_bot_42:                   BEQ     $15 $0
        f_top_41
368                             XOR     $9 $16
369 f_bot_42_i:                 BEQ     $15 $0
        f_top_41_i
370                             XORI    $16 1
371 f_top_41_i:                 BEQ     $15 $0
        f_bot_42_i
372                             ANDX    $15 $12 $14
373 cmp_bot_40_i:               BNE     $13 $0
        cmp_top_39_i
374                             XORI    $14 1
375 cmp_top_39_i:               BNE     $13 $0
        cmp_bot_40_i
376                             EXCH    $13 $6
377 cmp_bot_38_i:               BEQ     $11 $0
        cmp_top_37_i
378                             XORI    $12 1
379 cmp_top_37_i:               BEQ     $11 $0
        cmp_bot_38_i
380                             ADD     $10 $3
381                             ADDI    $10 2
382                             EXCH    $11 $10
383                             ADDI    $10 -2
384                             SUB     $10 $3
385                             ADD     $10 $3
386                             ADDI    $10 2
387                             EXCH    $11 $10
388                             ADDI    $10 -2
389                             SUB     $10 $3
390 cmp_top_47:                 BEQ     $11 $0
        cmp_bot_48
391                             XORI    $12 1
392 cmp_bot_48:                 BEQ     $11 $0
        cmp_top_47
393 f_top_49:                   BEQ     $12 $0
        f_bot_50
394                             XORI    $13 1
395 f_bot_50:                   BEQ     $12 $0
        f_top_49
396                             XOR     $9 $13
397 f_bot_50_i:                 BEQ     $12 $0
        f_top_49_i
398                             XORI    $13 1
399 f_top_49_i:                 BEQ     $12 $0
        f_bot_50_i
400 cmp_bot_48_i:               BEQ     $11 $0
        cmp_top_47_i
401                             XORI    $12 1
402 cmp_top_47_i:               BEQ     $11 $0
        cmp_bot_48_i
403                             ADD     $10 $3
404                             ADDI    $10 2
405                             EXCH    $11 $10
406                             ADDI    $10 -2
407                             SUB     $10 $3
408 test_43:                    BEQ     $9 $0
        test_false_45
409                             XORI    $9 1
410                             ADD     $10 $3
411                             ADDI    $10 2
412                             EXCH    $11 $10
413                             ADDI    $10 -2
414                             SUB     $10 $3
415                             XOR     $12 $11
416 loadMetAdd_51:              EXCH    $13 $12
417                             ADDI    $13 1
418                             EXCH    $14 $13
419                             XOR     $15 $14
420                             EXCH    $14 $13
421                             ADDI    $13 -1
422                             EXCH    $13 $12
423                             ADD     $10 $3
424                             ADDI    $10 2
425                             EXCH    $11 $10
426                             ADDI    $10 -2
427                             SUB     $10 $3
428                             EXCH    $3 $1
429                             ADDI    $1 -1
430                             EXCH    $6 $1
431                             ADDI    $1 -1
432                             EXCH    $7 $1
433                             ADDI    $1 -1
434                             EXCH    $12 $1
435                             ADDI    $1 -1
436                             ADDI    $15 -435
437 l_jmp_52:                   SWAPBR  $15
438                             NEG     $15
439                             ADDI    $15 435
440                             ADDI    $1 1
441                             EXCH    $12 $1
442                             ADDI    $1 1
443                             EXCH    $7 $1
444                             ADDI    $1 1
445                             EXCH    $6 $1
446                             ADDI    $1 1
447                             EXCH    $3 $1
448                             ADD     $10 $3
449                             ADDI    $10 2
450                             EXCH    $11 $10
451                             ADDI    $10 -2
452                             SUB     $10 $3
453                             EXCH    $13 $12
454                             ADDI    $13 1
455                             EXCH    $14 $13
```

```
456                         XOR    $15 $14
457                         EXCH   $14 $13
458                         ADDI   $13 -1
459 loadMetAdd_51_i:        EXCH   $13 $12
460                         XOR    $12 $11
461                         ADD    $10 $3
462                         ADDI   $10 2
463                         EXCH   $11 $10
464                         ADDI   $10 -2
465                         SUB    $10 $3
466                         XORI   $9 1
467 assert_true_44:         BRA    assert_46
468 test_false_45:          BRA    test_43
469 assert_46:               BNE    $9 $0
        assert_true_44
470                         ADD    $10 $3
471                         ADDI   $10 2
472                         EXCH   $11 $10
473                         ADDI   $10 -2
474                         SUB    $10 $3
475 cmp_top_53:             BEQ    $11 $0
        cmp_bot_54
476                         XORI   $12 1
477 cmp_bot_54:             BEQ    $11 $0
        cmp_top_53
478 f_top_55:               BEQ    $12 $0
        f_bot_56
479                         XORI   $13 1
480 f_bot_56:               BEQ    $12 $0
        f_top_55
481                         XOR    $9 $13
482 f_bot_56_i:             BEQ    $12 $0
        f_top_55_i
483                         XORI   $13 1
484 f_top_55_i:             BEQ    $12 $0
        f_bot_56_i
485 cmp_bot_54_i:           BEQ    $11 $0
        cmp_top_53_i
486                         XORI   $12 1
487 cmp_top_53_i:           BEQ    $11 $0
        cmp_bot_54_i
488                         ADD    $10 $3
489                         ADDI   $10 2
490                         EXCH   $11 $10
491                         ADDI   $10 -2
492                         SUB    $10 $3
493                         XORI   $8 1
494 assert_true_19:         BRA    assert_21
495 test_false_20:          BRA    test_18
496                         ADD    $10 $3
497                         ADDI   $10 3
498                         EXCH   $11 $10
499                         ADDI   $10 -3
500                         SUB    $10 $3
501 cmp_top_61:             BNE    $11 $0
        cmp_bot_62
502                         XORI   $12 1
503 cmp_bot_62:             BNE    $11 $0
        cmp_top_61
504                         EXCH   $13 $6
505 cmp_top_63:             BEQ    $13 $0
        cmp_bot_64
506                         XORI   $14 1
507 cmp_bot_64:             BEQ    $13 $0
        cmp_top_63
508                         ANDX   $15 $12 $14
509 f_top_65:               BEQ    $15 $0
        f_bot_66
510                         XORI   $16 1
511 f_bot_66:               BEQ    $15 $0
        f_top_65
512                         XOR    $9 $16
513 f_bot_66_i:             BEQ    $15 $0
        f_top_65_i
514                         XORI   $16 1
515 f_top_65_i:             BEQ    $15 $0
        f_bot_66_i
516                         ANDX   $15 $12 $14
517 cmp_bot_64_i:           BEQ    $13 $0
        cmp_top_63_i
518                         XORI   $14 1
519 cmp_top_63_i:           BEQ    $13 $0
        cmp_bot_64_i
520                         EXCH   $13 $6
521 cmp_bot_62_i:           BNE    $11 $0
        cmp_top_61_i
522                         XORI   $12 1
523 cmp_top_61_i:           BNE    $11 $0
        cmp_bot_62_i
524                         ADD    $10 $3
525                         ADDI   $10 3
526                         EXCH   $11 $10
527                         ADDI   $10 -3
528                         SUB    $10 $3
529 test_57:                 BEQ    $9 $0
        test_false_59
530                         XORI   $9 1
531                         ADD    $10 $3
532                         ADDI   $10 3
533                         EXCH   $11 $10
534                         ADDI   $10 -3
535                         SUB    $10 $3
536                         EXCH   $12 $6
537 swap_67:                 XOR    $11 $12
538                         XOR    $12 $11
539                         XOR    $11 $12
540                         EXCH   $12 $6
541                         ADD    $10 $3
542                         ADDI   $10 3
543                         EXCH   $11 $10
544                         ADDI   $10 -3
545                         SUB    $10 $3
546                         XORI   $9 1
547 assert_true_58:         BRA    assert_60
548 test_false_59:          BRA    test_57
549 assert_60:               BNE    $9 $0
        assert_true_58
550                         ADD    $10 $3
551                         ADDI   $10 3
552                         EXCH   $11 $10
553                         ADDI   $10 -3
554                         SUB    $10 $3
555 cmp_top_68:             BEQ    $11 $0
        cmp_bot_69
556                         XORI   $12 1
557 cmp_bot_69:             BEQ    $11 $0
        cmp_top_68
558                         EXCH   $13 $6
559 cmp_top_70:             BNE    $13 $0
        cmp_bot_71
560                         XORI   $14 1
561 cmp_bot_71:             BNE    $13 $0
```

```
    cmp_top_70
                        ANDX    $15 $12 $14
f_top_72:               BEQ     $15 $0
    f_bot_73
                        XORI    $16 1
f_bot_73:               BEQ     $15 $0
    f_top_72
                        XOR     $9 $16
f_bot_73_i:             BEQ     $15 $0
    f_top_72_i
                        XORI    $16 1
f_top_72_i:             BEQ     $15 $0
    f_bot_73_i
                        ANDX    $15 $12 $14
cmp_bot_71_i:           BNE     $13 $0
    cmp_top_70_i
                        XORI    $14 1
cmp_top_70_i:           BNE     $13 $0
    cmp_bot_71_i
                        EXCH    $13 $6
cmp_bot_69_i:           BEQ     $11 $0
    cmp_top_68_i
                        XORI    $12 1
cmp_top_68_i:           BEQ     $11 $0
    cmp_bot_69_i
                        ADD     $10 $3
                        ADDI    $10 3
                        EXCH    $11 $10
                        ADDI    $10 -3
                        SUB     $10 $3
                        ADD     $10 $3
                        ADDI    $10 3
                        EXCH    $11 $10
                        ADDI    $10 -3
                        SUB     $10 $3
cmp_top_78:             BEQ     $11 $0
    cmp_bot_79
                        XORI    $12 1
cmp_bot_79:             BEQ     $11 $0
    cmp_top_78
f_top_80:               BEQ     $12 $0
    f_bot_81
                        XORI    $13 1
f_bot_81:               BEQ     $12 $0
    f_top_80
                        XOR     $9 $13
f_bot_81_i:             BEQ     $12 $0
    f_top_80_i
                        XORI    $13 1
f_top_80_i:             BEQ     $12 $0
    f_bot_81_i
cmp_bot_79_i:           BEQ     $11 $0
    cmp_top_78_i
                        XORI    $12 1
cmp_top_78_i:           BEQ     $11 $0
    cmp_bot_79_i
                        ADD     $10 $3
                        ADDI    $10 3
                        EXCH    $11 $10
                        ADDI    $10 -3
                        SUB     $10 $3
test_74:                BEQ     $9 $0
    test_false_76
                        XORI    $9 1
                        ADD     $10 $3
                        ADDI    $10 3
                        EXCH    $11 $10
                        ADDI    $10 -3
                        SUB     $10 $3
                        XORI    $9 1
loadMetAdd_82:          EXCH    $13 $12
                        ADDI    $13 1
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -1
                        EXCH    $13 $12
                        ADD     $10 $3
                        ADDI    $10 3
                        EXCH    $11 $10
                        ADDI    $10 -3
                        SUB     $10 $3
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $12 $1
                        ADDI    $1 -1
                        ADDI    $15 -633
l_jmp_83:               SWAPBR  $15
                        NEG     $15
                        ADDI    $15 633
                        ADDI    $1 1
                        EXCH    $12 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADD     $10 $3
                        ADDI    $10 3
                        EXCH    $11 $10
                        ADDI    $10 -3
                        SUB     $10 $3
                        EXCH    $13 $12
                        ADDI    $13 1
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -1
loadMetAdd_82_i:        EXCH    $13 $12
                        XOR     $12 $11
                        ADD     $10 $3
                        ADDI    $10 3
                        EXCH    $11 $10
                        ADDI    $10 -3
                        SUB     $10 $3
                        XORI    $9 1
assert_true_75:         BRA     assert_77
test_false_76:          BRA     test_74
assert_77:              BNE     $9 $0
    assert_true_75
                        ADD     $10 $3
                        ADDI    $10 3
                        EXCH    $11 $10
                        ADDI    $10 -3
                        SUB     $10 $3
cmp_top_84:             BEQ     $11 $0
    cmp_bot_85
```

```
674                         XORI    $12 1
675 cmp_bot_85:             BEQ     $11 $0
        cmp_top_84
676 f_top_86:               BEQ     $12 $0
        f_bot_87
677                         XORI    $13 1
678 f_bot_87:               BEQ     $12 $0
        f_top_86
679                         XOR     $9 $13
680 f_bot_87_i:             BEQ     $12 $0
        f_top_86_i
681                         XORI    $13 1
682 f_top_86_i:             BEQ     $12 $0
        f_bot_87_i
683 cmp_bot_85_i:           BEQ     $11 $0
        cmp_top_84_i
684                         XORI    $12 1
685 cmp_top_84_i:           BEQ     $11 $0
        cmp_bot_85_i
686                         ADD     $10 $3
687                         ADDI    $10 3
688                         EXCH    $11 $10
689                         ADDI    $10 -3
690                         SUB     $10 $3
691 assert_21:              BNE     $8 $0
        assert_true_19
692                         EXCH    $9 $7
693                         ADD     $10 $3
694                         ADDI    $10 4
695                         EXCH    $11 $10
696                         ADDI    $10 -4
697                         SUB     $10 $3
698                         XOR     $12 $9
699                         SUB     $12 $11
700 cmp_top_88:             BGEZ    $12 cmp_bot_89
701                         XORI    $13 1
702 cmp_bot_89:             BGEZ    $12 cmp_top_88
703 f_top_90:               BEQ     $13 $0
        f_bot_91
704                         XORI    $14 1
705 f_bot_91:               BEQ     $13 $0
        f_top_90
706                         XOR     $8 $14
707 f_bot_91_i:             BEQ     $13 $0
        f_top_90_i
708                         XORI    $14 1
709 f_top_90_i:             BEQ     $13 $0
        f_bot_91_i
710 cmp_bot_89_i:           BGEZ    $12
        cmp_top_88_i
711                         XORI    $13 1
712 cmp_top_88_i:           BGEZ    $12
        cmp_bot_89_i
713                         ADD     $12 $11
714                         XOR     $12 $9
715                         ADD     $10 $3
716                         ADDI    $10 4
717                         EXCH    $11 $10
718                         ADDI    $10 -4
719                         SUB     $10 $3
720                         EXCH    $9 $7
721 l_insertNode_5_bot:     BRA
        l_insertNode_5_top
722 l_getSum_6_top:         BRA     l_getSum_6_bot
723                         ADDI    $1 1
724                         EXCH    $2 $1
725                         EXCH    $6 $1
726                         ADDI    $1 -1
727                         EXCH    $3 $1
728                         ADDI    $1 -1
729 l_getSum_6:             SWAPBR  $2
730                         NEG     $2
731                         ADDI    $1 1
732                         EXCH    $3 $1
733                         ADDI    $1 1
734                         EXCH    $6 $1
735                         EXCH    $2 $1
736                         ADDI    $1 -1
737                         EXCH    $7 $6
738                         ADD     $8 $3
739                         ADDI    $8 4
740                         EXCH    $9 $8
741                         ADDI    $8 -4
742                         SUB     $8 $3
743                         ADD     $7 $9
744                         ADD     $8 $3
745                         ADDI    $8 4
746                         EXCH    $9 $8
747                         ADDI    $8 -4
748                         SUB     $8 $3
749                         EXCH    $7 $6
750                         ADD     $8 $3
751                         ADDI    $8 2
752                         EXCH    $9 $8
753                         ADDI    $8 -2
754                         SUB     $8 $3
755 cmp_top_96:             BEQ     $9 $0
        cmp_bot_97
756                         XORI    $10 1
757 cmp_bot_97:             BEQ     $9 $0
        cmp_top_96
758 f_top_98:               BEQ     $10 $0
        f_bot_99
759                         XORI    $11 1
760 f_bot_99:               BEQ     $10 $0
        f_top_98
761                         XOR     $7 $11
762 f_bot_99_i:             BEQ     $10 $0
        f_top_98_i
763                         XORI    $11 1
764 f_top_98_i:             BEQ     $10 $0
        f_bot_99_i
765 cmp_bot_97_i:           BEQ     $9 $0
        cmp_top_96_i
766                         XORI    $10 1
767 cmp_top_96_i:           BEQ     $9 $0
        cmp_bot_97_i
768                         ADD     $8 $3
769                         ADDI    $8 2
770                         EXCH    $9 $8
771                         ADDI    $8 -2
772                         SUB     $8 $3
773 test_92:                BEQ     $7 $0
        test_false_94
774                         XORI    $7 1
775                         ADD     $8 $3
776                         ADDI    $8 2
777                         EXCH    $9 $8
778                         ADDI    $8 -2
779                         SUB     $8 $3
780                         XOR     $10 $9
781 loadMetAdd_100:         EXCH    $11 $10
```

```
                        ADDI    $11 2
                        EXCH    $12 $11
                        XOR     $13 $12
                        EXCH    $12 $11
                        ADDI    $11 -2
                        EXCH    $11 $10
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        ADDI    $13 -798
l_jmp_101:              SWAPBR  $13
                        NEG     $13
                        ADDI    $13 798
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
                        EXCH    $11 $10
                        ADDI    $11 2
                        EXCH    $12 $11
                        XOR     $13 $12
                        EXCH    $12 $11
                        ADDI    $11 -2
loadMetAdd_100_i:       EXCH    $11 $10
                        XOR     $10 $9
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
                        XORI    $7 1
assert_true_93:         BRA     assert_95
test_false_94:          BRA     test_92
assert_95:              BNE     $7 $0
    assert_true_93
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
cmp_top_102:            BEQ     $9 $0
    cmp_bot_103
                        XORI    $10 1
cmp_bot_103:            BEQ     $9 $0
    cmp_top_102
f_top_104:              BEQ     $10 $0
    f_bot_105
                        XORI    $11 1
f_bot_105:              BEQ     $10 $0
    f_top_104
                        XOR     $7 $11
f_bot_105_i:            BEQ     $10 $0
    f_top_104_i
                        XORI    $11 1
f_top_104_i:            BEQ     $10 $0
    f_bot_105_i
cmp_bot_103_i:          BEQ     $9 $0
    cmp_top_102_i
                        XORI    $10 1
cmp_top_102_i:          BEQ     $9 $0
    cmp_bot_103_i
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
                        ADD     $8 $3
                        ADDI    $8 3
                        EXCH    $9 $8
                        ADDI    $8 -3
                        SUB     $8 $3
cmp_top_110:            BEQ     $9 $0
    cmp_bot_111
                        XORI    $10 1
cmp_bot_111:            BEQ     $9 $0
    cmp_top_110
f_top_112:              BEQ     $10 $0
    f_bot_113
                        XORI    $11 1
f_bot_113:              BEQ     $10 $0
    f_top_112
                        XOR     $7 $11
f_bot_113_i:            BEQ     $10 $0
    f_top_112_i
                        XORI    $11 1
f_top_112_i:            BEQ     $10 $0
    f_bot_113_i
cmp_bot_111_i:          BEQ     $9 $0
    cmp_top_110_i
                        XORI    $10 1
cmp_top_110_i:          BEQ     $9 $0
    cmp_bot_111_i
                        ADD     $8 $3
                        ADDI    $8 3
                        EXCH    $9 $8
                        ADDI    $8 -3
                        SUB     $8 $3
test_106:               BEQ     $7 $0
    test_false_108
                        XORI    $7 1
                        ADD     $8 $3
                        ADDI    $8 3
                        EXCH    $9 $8
                        ADDI    $8 -3
                        SUB     $8 $3
                        XOR     $10 $9
loadMetAdd_114:         EXCH    $11 $10
                        ADDI    $11 2
                        EXCH    $12 $11
                        XOR     $13 $12
                        EXCH    $12 $11
                        ADDI    $11 -2
                        EXCH    $11 $10
                        ADD     $8 $3
                        ADDI    $8 3
                        EXCH    $9 $8
                        ADDI    $8 -3
```

```
                        SUB     $8 $3
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        ADDI    $13 -902
l_jmp_115:              SWAPBR  $13
                        NEG     $13
                        ADDI    $13 902
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADD     $8 $3
                        ADDI    $8 3
                        EXCH    $9 $8
                        ADDI    $8 -3
                        SUB     $8 $3
                        EXCH    $11 $10
                        ADDI    $11 2
                        EXCH    $12 $11
                        XOR     $13 $12
                        EXCH    $12 $11
                        ADDI    $11 -2
loadMetAdd_114_i:       EXCH    $11 $10
                        XOR     $10 $9
                        ADD     $8 $3
                        ADDI    $8 3
                        EXCH    $9 $8
                        ADDI    $8 -3
                        SUB     $8 $3
                        XORI    $7 1
assert_true_107:        BRA     assert_109
test_false_108:         BRA     test_106
assert_109:             BNE     $7 $0
    assert_true_107
                        ADD     $8 $3
                        ADDI    $8 3
                        EXCH    $9 $8
                        ADDI    $8 -3
                        SUB     $8 $3
cmp_top_116:            BEQ     $9 $0
    cmp_bot_117
                        XORI    $10 1
cmp_bot_117:            BEQ     $9 $0
    cmp_top_116
f_top_118:              BEQ     $10 $0
    f_bot_119
                        XORI    $11 1
f_bot_119:              BEQ     $10 $0
    f_top_118
                        XOR     $7 $11
f_bot_119_i:            BEQ     $10 $0
    f_top_118_i
                        XORI    $11 1
f_top_118_i:            BEQ     $10 $0
    f_bot_119_i
cmp_bot_117_i:          BEQ     $9 $0
    cmp_top_116_i
                        XORI    $10 1
cmp_top_116_i:          BEQ     $9 $0
    cmp_bot_117_i
                        ADD     $8 $3
                        ADDI    $8 3
                        EXCH    $9 $8
                        ADDI    $8 -3
                        SUB     $8 $3
l_getSum_6_bot:         BRA     l_getSum_6_top
l_mirror_7_top:         BRA     l_mirror_7_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $3 $1
                        ADDI    $1 -1
l_mirror_7:             SWAPBR  $2
                        NEG     $2
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
                        ADD     $6 $3
                        ADDI    $6 2
                        EXCH    $7 $6
                        ADDI    $6 -2
                        SUB     $6 $3
                        ADD     $8 $3
                        ADDI    $8 3
                        EXCH    $9 $8
                        ADDI    $8 -3
                        SUB     $8 $3
swap_120:               XOR     $7 $9
                        XOR     $9 $7
                        XOR     $7 $9
                        ADD     $8 $3
                        ADDI    $8 3
                        EXCH    $9 $8
                        ADDI    $8 -3
                        SUB     $8 $3
                        ADD     $6 $3
                        ADDI    $6 2
                        EXCH    $7 $6
                        ADDI    $6 -2
                        SUB     $6 $3
                        ADD     $7 $3
                        ADDI    $7 2
                        EXCH    $8 $7
                        ADDI    $7 -2
                        SUB     $7 $3
cmp_top_125:            BNE     $8 $0
    cmp_bot_126
                        XORI    $9 1
cmp_bot_126:            BNE     $8 $0
    cmp_top_125
f_top_127:              BEQ     $9 $0
    f_bot_128
                        XORI    $10 1
f_bot_128:              BEQ     $9 $0
    f_top_127
                        XOR     $6 $10
f_bot_128_i:            BEQ     $9 $0
    f_top_127_i
                        XORI    $10 1
f_top_127_i:            BEQ     $9 $0
    f_bot_128_i
cmp_bot_126_i:          BNE     $8 $0
    cmp_top_125_i
                        XORI    $9 1
cmp_top_125_i:          BNE     $8 $0
    cmp_bot_126_i
```

```
                        ADD     $7 $3
                        ADDI    $7 2
                        EXCH    $8 $7
                        ADDI    $7 -2
                        SUB     $7 $3
test_121:               BEQ     $6 $0
    test_false_123
                        XORI    $6 1
                        XORI    $6 1
assert_true_122:        BRA     assert_124
test_false_123:         BRA     test_121
                        ADD     $7 $3
                        ADDI    $7 2
                        EXCH    $8 $7
                        ADDI    $7 -2
                        SUB     $7 $3
                        XOR     $9 $8
loadMetAdd_129:         EXCH    $10 $9
                        ADDI    $10 3
                        EXCH    $11 $10
                        XOR     $12 $11
                        EXCH    $11 $10
                        ADDI    $10 -3
                        EXCH    $10 $9
                        ADD     $7 $3
                        ADDI    $7 2
                        EXCH    $8 $7
                        ADDI    $7 -2
                        SUB     $7 $3
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        ADDI    $12 -1042
l_jmp_130:              SWAPBR  $12
                        NEG     $12
                        ADDI    $12 1042
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADD     $7 $3
                        ADDI    $7 2
                        EXCH    $8 $7
                        ADDI    $7 -2
                        SUB     $7 $3
                        EXCH    $10 $9
                        ADDI    $10 3
                        EXCH    $11 $10
                        XOR     $12 $11
                        EXCH    $11 $10
                        ADDI    $10 -3
loadMetAdd_129_i:       EXCH    $10 $9
                        XOR     $9 $8
                        ADD     $7 $3
                        ADDI    $7 2
                        EXCH    $8 $7
                        ADDI    $7 -2
                        SUB     $7 $3
assert_124:             BNE     $6 $0
    assert_true_122
                        ADD     $7 $3
                        ADDI    $7 2
                        EXCH    $8 $7
                        ADDI    $7 -2
                        SUB     $7 $3
cmp_top_131:            BNE     $8 $0
    cmp_bot_132
                        XORI    $9 1
cmp_bot_132:            BNE     $8 $0
    cmp_top_131
f_top_133:              BEQ     $9 $0
    f_bot_134
                        XORI    $10 1
f_bot_134:              BEQ     $9 $0
    f_top_133
                        XOR     $6 $10
f_bot_134_i:            BEQ     $9 $0
    f_top_133_i
                        XORI    $10 1
f_top_133_i:            BEQ     $9 $0
    f_bot_134_i
cmp_bot_132_i:          BNE     $8 $0
    cmp_top_131_i
                        XORI    $9 1
cmp_top_131_i:          BNE     $8 $0
    cmp_bot_132_i
                        ADD     $7 $3
                        ADDI    $7 2
                        EXCH    $8 $7
                        ADDI    $7 -2
                        SUB     $7 $3
                        ADD     $7 $3
                        ADDI    $7 3
                        EXCH    $8 $7
                        ADDI    $7 -3
                        SUB     $7 $3
cmp_top_139:            BNE     $8 $0
    cmp_bot_140
                        XORI    $9 1
cmp_bot_140:            BNE     $8 $0
    cmp_top_139
f_top_141:              BEQ     $9 $0
    f_bot_142
                        XORI    $10 1
f_bot_142:              BEQ     $9 $0
    f_top_141
                        XOR     $6 $10
f_bot_142_i:            BEQ     $9 $0
    f_top_141_i
                        XORI    $10 1
f_top_141_i:            BEQ     $9 $0
    f_bot_142_i
cmp_bot_140_i:          BNE     $8 $0
    cmp_top_139_i
                        XORI    $9 1
cmp_top_139_i:          BNE     $8 $0
    cmp_bot_140_i
                        ADD     $7 $3
                        ADDI    $7 3
                        EXCH    $8 $7
                        ADDI    $7 -3
                        SUB     $7 $3
test_135:               BEQ     $6 $0
    test_false_137
                        XORI    $6 1
                        XORI    $6 1
assert_true_136:        BRA     assert_138
test_false_137:         BRA     test_135
                        ADD     $7 $3
                        ADDI    $7 3
                        EXCH    $8 $7
```

```
1124                            ADDI    $7 -3
1125                            SUB     $7 $3
1126                            XOR     $9 $8
1127 loadMetAdd_143:           EXCH    $10 $9
1128                            ADDI    $10 3
1129                            EXCH    $11 $10
1130                            XOR     $12 $11
1131                            EXCH    $11 $10
1132                            ADDI    $10 -3
1133                            EXCH    $10 $9
1134                            ADD     $7 $3
1135                            ADDI    $7 3
1136                            EXCH    $8 $7
1137                            ADDI    $7 -3
1138                            SUB     $7 $3
1139                            EXCH    $3 $1
1140                            ADDI    $1 -1
1141                            EXCH    $9 $1
1142                            ADDI    $1 -1
1143                            ADDI    $12 -1142
1144 l_jmp_144:                SWAPBR  $12
1145                            NEG     $12
1146                            ADDI    $12 1142
1147                            ADDI    $1 1
1148                            EXCH    $9 $1
1149                            ADDI    $1 1
1150                            EXCH    $3 $1
1151                            ADD     $7 $3
1152                            ADDI    $7 3
1153                            EXCH    $8 $7
1154                            ADDI    $7 -3
1155                            SUB     $7 $3
1156                            EXCH    $10 $9
1157                            ADDI    $10 3
1158                            EXCH    $11 $10
1159                            XOR     $12 $11
1160                            EXCH    $11 $10
1161                            ADDI    $10 -3
1162 loadMetAdd_143_i:         EXCH    $10 $9
1163                            XOR     $9 $8
1164                            ADD     $7 $3
1165                            ADDI    $7 3
1166                            EXCH    $8 $7
1167                            ADDI    $7 -3
1168                            SUB     $7 $3
1169 assert_138:                BNE     $6 $0
         assert_true_136
1170                            ADD     $7 $3
1171                            ADDI    $7 3
1172                            EXCH    $8 $7
1173                            ADDI    $7 -3
1174                            SUB     $7 $3
1175 cmp_top_145:               BNE     $8 $0
         cmp_bot_146
1176                            XORI    $9 1
1177 cmp_bot_146:               BNE     $8 $0
         cmp_top_145
1178 f_top_147:                 BEQ     $9 $0
         f_bot_148
1179                            XORI    $10 1
1180 f_bot_148:                 BEQ     $9 $0
         f_top_147
1181                            XOR     $6 $10
1182 f_bot_148_i:               BEQ     $9 $0
         f_top_147_i
1183                            XORI    $10 1
1184 f_top_147_i:               BEQ     $9 $0
         f_bot_148_i
1185 cmp_bot_146_i:             BNE     $8 $0
         cmp_top_145_i
1186                            XORI    $9 1
1187 cmp_top_145_i:             BNE     $8 $0
         cmp_bot_146_i
1188                            ADD     $7 $3
1189                            ADDI    $7 3
1190                            EXCH    $8 $7
1191                            ADDI    $7 -3
1192                            SUB     $7 $3
1193 l_mirror_7_bot:            BRA     l_mirror_7_top
1194 l_insertNode_1_top:        BRA
         l_insertNode_1_bot
1195                            ADDI    $1 1
1196                            EXCH    $2 $1
1197                            EXCH    $6 $1
1198                            ADDI    $1 -1
1199                            EXCH    $7 $1
1200                            ADDI    $1 -1
1201                            EXCH    $3 $1
1202                            ADDI    $1 -1
1203 l_insertNode_1:            SWAPBR  $2
1204                            NEG     $2
1205                            ADDI    $1 1
1206                            EXCH    $3 $1
1207                            ADDI    $1 1
1208                            EXCH    $7 $1
1209                            ADDI    $1 1
1210                            EXCH    $6 $1
1211                            EXCH    $2 $1
1212                            ADDI    $1 -1
1213                            ADD     $9 $3
1214                            ADDI    $9 2
1215                            EXCH    $10 $9
1216                            ADDI    $9 -2
1217                            SUB     $9 $3
1218 cmp_top_153:               BNE     $10 $0
         cmp_bot_154
1219                            XORI    $11 1
1220 cmp_bot_154:               BNE     $10 $0
         cmp_top_153
1221                            EXCH    $12 $6
1222 cmp_top_155:               BEQ     $12 $0
         cmp_bot_156
1223                            XORI    $13 1
1224 cmp_bot_156:               BEQ     $12 $0
         cmp_top_155
1225                            ANDX    $14 $11 $13
1226 f_top_157:                 BEQ     $14 $0
         f_bot_158
1227                            XORI    $15 1
1228 f_bot_158:                 BEQ     $14 $0
         f_top_157
1229                            XOR     $8 $15
1230 f_bot_158_i:               BEQ     $14 $0
         f_top_157_i
1231                            XORI    $15 1
1232 f_top_157_i:               BEQ     $14 $0
         f_bot_158_i
1233                            ANDX    $14 $11 $13
1234 cmp_bot_156_i:             BEQ     $12 $0
         cmp_top_155_i
1235                            XORI    $13 1
1236 cmp_top_155_i:             BEQ     $12 $0
```

```
        cmp_bot_156_i
1237                    EXCH    $12 $6
1238 cmp_bot_154_i:    BNE     $10 $0
        cmp_top_153_i
1239                    XORI    $11 1
1240 cmp_top_153_i:    BNE     $10 $0
        cmp_bot_154_i
1241                    ADD     $9 $3
1242                    ADDI    $9 2
1243                    EXCH    $10 $9
1244                    ADDI    $9 -2
1245                    SUB     $9 $3
1246 test_149:         BEQ     $8 $0
        test_false_151
1247                    XORI    $8 1
1248                    ADD     $9 $3
1249                    ADDI    $9 2
1250                    EXCH    $10 $9
1251                    ADDI    $9 -2
1252                    SUB     $9 $3
1253                    EXCH    $11 $6
1254 swap_159:         XOR     $10 $11
1255                    XOR     $11 $10
1256                    XOR     $10 $11
1257                    EXCH    $11 $6
1258                    ADD     $9 $3
1259                    ADDI    $9 2
1260                    EXCH    $10 $9
1261                    ADDI    $9 -2
1262                    SUB     $9 $3
1263                    XORI    $8 1
1264 assert_true_150:   BRA     assert_152
1265 test_false_151:    BRA     test_149
1266 assert_152:        BNE     $8 $0
        assert_true_150
1267                    ADD     $9 $3
1268                    ADDI    $9 2
1269                    EXCH    $10 $9
1270                    ADDI    $9 -2
1271                    SUB     $9 $3
1272 cmp_top_160:       BEQ     $10 $0
        cmp_bot_161
1273                    XORI    $11 1
1274 cmp_bot_161:       BEQ     $10 $0
        cmp_top_160
1275                    EXCH    $12 $6
1276 cmp_top_162:       BNE     $12 $0
        cmp_bot_163
1277                    XORI    $13 1
1278 cmp_bot_163:       BNE     $12 $0
        cmp_top_162
1279                    ANDX    $14 $11 $13
1280 f_top_164:        BEQ     $14 $0
        f_bot_165
1281                    XORI    $15 1
1282 f_bot_165:        BEQ     $14 $0
        f_top_164
1283                    XOR     $8 $15
1284 f_bot_165_i:      BEQ     $14 $0
        f_top_164_i
1285                    XORI    $15 1
1286 f_top_164_i:      BEQ     $14 $0
        f_bot_165_i
1287                    ANDX    $14 $11 $13
1288 cmp_bot_163_i:    BNE     $12 $0
        cmp_top_162_i
1289                    XORI    $13 1
1290 cmp_top_162_i:    BNE     $12 $0
        cmp_bot_163_i
1291                    EXCH    $12 $6
1292 cmp_bot_161_i:    BEQ     $10 $0
        cmp_top_160_i
1293                    XORI    $11 1
1294 cmp_top_160_i:    BEQ     $10 $0
        cmp_bot_161_i
1295                    ADD     $9 $3
1296                    ADDI    $9 2
1297                    EXCH    $10 $9
1298                    ADDI    $9 -2
1299                    SUB     $9 $3
1300                    ADD     $9 $3
1301                    ADDI    $9 2
1302                    EXCH    $10 $9
1303                    ADDI    $9 -2
1304                    SUB     $9 $3
1305 cmp_top_170:       BEQ     $10 $0
        cmp_bot_171
1306                    XORI    $11 1
1307 cmp_bot_171:       BEQ     $10 $0
        cmp_top_170
1308 f_top_172:        BEQ     $11 $0
        f_bot_173
1309                    XORI    $12 1
1310 f_bot_173:        BEQ     $11 $0
        f_top_172
1311                    XOR     $8 $12
1312 f_bot_173_i:      BEQ     $11 $0
        f_top_172_i
1313                    XORI    $12 1
1314 f_top_172_i:      BEQ     $11 $0
        f_bot_173_i
1315 cmp_bot_171_i:    BEQ     $10 $0
        cmp_top_170_i
1316                    XORI    $11 1
1317 cmp_top_170_i:    BEQ     $10 $0
        cmp_bot_171_i
1318                    ADD     $9 $3
1319                    ADDI    $9 2
1320                    EXCH    $10 $9
1321                    ADDI    $9 -2
1322                    SUB     $9 $3
1323 test_166:         BEQ     $8 $0
        test_false_168
1324                    XORI    $8 1
1325                    ADD     $9 $3
1326                    ADDI    $9 2
1327                    EXCH    $10 $9
1328                    ADDI    $9 -2
1329                    SUB     $9 $3
1330                    XOR     $11 $10
1331 loadMetAdd_174:    EXCH    $12 $11
1332                    ADDI    $12 1
1333                    EXCH    $13 $12
1334                    XOR     $14 $13
1335                    EXCH    $13 $12
1336                    ADDI    $12 -1
1337                    EXCH    $12 $11
1338                    ADD     $9 $3
1339                    ADDI    $9 2
1340                    EXCH    $10 $9
1341                    ADDI    $9 -2
1342                    SUB     $9 $3
```

```
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $11 $1
                        ADDI    $1 -1
                        ADDI    $14 -1350
l_jmp_175:              SWAPBR  $14
                        NEG     $14
                        ADDI    $14 1350
                        ADDI    $1 1
                        EXCH    $11 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADD     $9 $3
                        ADDI    $9 2
                        EXCH    $10 $9
                        ADDI    $9 -2
                        SUB     $9 $3
                        EXCH    $12 $11
                        ADDI    $12 1
                        EXCH    $13 $12
                        XOR     $14 $13
                        EXCH    $13 $12
                        ADDI    $12 -1
loadMetAdd_174_i:       EXCH    $12 $11
                        XOR     $11 $10
                        ADD     $9 $3
                        ADDI    $9 2
                        EXCH    $10 $9
                        ADDI    $9 -2
                        SUB     $9 $3
                        XORI    $8 1
assert_true_167:        BRA     assert_169
test_false_168:         BRA     test_166
assert_169:             BNE     $8 $0
    assert_true_167
                        ADD     $9 $3
                        ADDI    $9 2
                        EXCH    $10 $9
                        ADDI    $9 -2
                        SUB     $9 $3
cmp_top_176:            BEQ     $10 $0
    cmp_bot_177
                        XORI    $11 1
cmp_bot_177:            BEQ     $10 $0
    cmp_top_176
f_top_178:              BEQ     $11 $0
    f_bot_179
                        XORI    $12 1
f_bot_179:              BEQ     $11 $0
    f_top_178
                        XOR     $8 $12
f_bot_179_i:            BEQ     $11 $0
    f_top_178_i
                        XORI    $12 1
f_top_178_i:            BEQ     $11 $0
    f_bot_179_i
cmp_bot_177_i:          BEQ     $10 $0
    cmp_top_176_i
                        XORI    $11 1
cmp_top_176_i:          BEQ     $10 $0
    cmp_bot_177_i
                        ADD     $9 $3
                        ADDI    $9 2
                        EXCH    $10 $9
                        ADDI    $9 -2
                        SUB     $9 $3
l_insertNode_1_bot:     BRA
    l_insertNode_1_top
l_sum_2_top:            BRA     l_sum_2_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $3 $1
                        ADDI    $1 -1
l_sum_2:                SWAPBR  $2
                        NEG     $2
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
cmp_top_184:            BEQ     $9 $0
    cmp_bot_185
                        XORI    $10 1
cmp_bot_185:            BEQ     $9 $0
    cmp_top_184
f_top_186:              BEQ     $10 $0
    f_bot_187
                        XORI    $11 1
f_bot_187:              BEQ     $10 $0
    f_top_186
                        XOR     $7 $11
f_bot_187_i:            BEQ     $10 $0
    f_top_186_i
                        XORI    $11 1
f_top_186_i:            BEQ     $10 $0
    f_bot_187_i
cmp_bot_185_i:          BEQ     $9 $0
    cmp_top_184_i
                        XORI    $10 1
cmp_top_184_i:          BEQ     $9 $0
    cmp_bot_185_i
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
test_180:               BEQ     $7 $0
    test_false_182
                        XORI    $7 1
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
                        XOR     $10 $9
loadMetAdd_188:         EXCH    $11 $10
```

```
1456                            ADDI    $11 2
1457                            EXCH    $12 $11
1458                            XOR     $13 $12
1459                            EXCH    $12 $11
1460                            ADDI    $11 -2
1461                            EXCH    $11 $10
1462                            ADD     $8 $3
1463                            ADDI    $8 2
1464                            EXCH    $9 $8
1465                            ADDI    $8 -2
1466                            SUB     $8 $3
1467                            EXCH    $3 $1
1468                            ADDI    $1 -1
1469                            EXCH    $6 $1
1470                            ADDI    $1 -1
1471                            EXCH    $10 $1
1472                            ADDI    $1 -1
1473                            ADDI    $13 -1472
1474 l_jmp_189:                 SWAPBR  $13
1475                            NEG     $13
1476                            ADDI    $13 1472
1477                            ADDI    $1 1
1478                            EXCH    $10 $1
1479                            ADDI    $1 1
1480                            EXCH    $6 $1
1481                            ADDI    $1 1
1482                            EXCH    $3 $1
1483                            ADD     $8 $3
1484                            ADDI    $8 2
1485                            EXCH    $9 $8
1486                            ADDI    $8 -2
1487                            SUB     $8 $3
1488                            EXCH    $11 $10
1489                            ADDI    $11 2
1490                            EXCH    $12 $11
1491                            XOR     $13 $12
1492                            EXCH    $12 $11
1493                            ADDI    $11 -2
1494 loadMetAdd_188_i:          EXCH    $11 $10
1495                            XOR     $10 $9
1496                            ADD     $8 $3
1497                            ADDI    $8 2
1498                            EXCH    $9 $8
1499                            ADDI    $8 -2
1500                            SUB     $8 $3
1501                            XORI    $7 1
1502 assert_true_181:           BRA     assert_183
1503 test_false_182:            BRA     test_180
1504 assert_183:                BNE     $7 $0
        assert_true_181
1505                            ADD     $8 $3
1506                            ADDI    $8 2
1507                            EXCH    $9 $8
1508                            ADDI    $8 -2
1509                            SUB     $8 $3
1510 cmp_top_190:               BEQ     $9 $0
        cmp_bot_191
1511                            XORI    $10 1
1512 cmp_bot_191:               BEQ     $9 $0
        cmp_top_190
1513 f_top_192:                 BEQ     $10 $0
        f_bot_193
1514                            XORI    $11 1
1515 f_bot_193:                 BEQ     $10 $0
        f_top_192
1516                            XOR     $7 $11
1517 f_bot_193_i:               BEQ     $10 $0
        f_top_192_i
1518                            XORI    $11 1
1519 f_top_192_i:               BEQ     $10 $0
        f_bot_193_i
1520 cmp_bot_191_i:             BEQ     $9 $0
        cmp_top_190_i
1521                            XORI    $10 1
1522 cmp_top_190_i:             BEQ     $9 $0
        cmp_bot_191_i
1523                            ADD     $8 $3
1524                            ADDI    $8 2
1525                            EXCH    $9 $8
1526                            ADDI    $8 -2
1527                            SUB     $8 $3
1528 l_sum_2_bot:               BRA     l_sum_2_top
1529 l_mirror_3_top:            BRA     l_mirror_3_bot
1530                            ADDI    $1 1
1531                            EXCH    $2 $1
1532                            EXCH    $3 $1
1533                            ADDI    $1 -1
1534 l_mirror_3:                SWAPBR  $2
1535                            NEG     $2
1536                            ADDI    $1 1
1537                            EXCH    $3 $1
1538                            EXCH    $2 $1
1539                            ADDI    $1 -1
1540                            ADD     $7 $3
1541                            ADDI    $7 2
1542                            EXCH    $8 $7
1543                            ADDI    $7 -2
1544                            SUB     $7 $3
1545 cmp_top_198:               BEQ     $8 $0
        cmp_bot_199
1546                            XORI    $9 1
1547 cmp_bot_199:               BEQ     $8 $0
        cmp_top_198
1548 f_top_200:                 BEQ     $9 $0
        f_bot_201
1549                            XORI    $10 1
1550 f_bot_201:                 BEQ     $9 $0
        f_top_200
1551                            XOR     $6 $10
1552 f_bot_201_i:               BEQ     $9 $0
        f_top_200_i
1553                            XORI    $10 1
1554 f_top_200_i:               BEQ     $9 $0
        f_bot_201_i
1555 cmp_bot_199_i:             BEQ     $8 $0
        cmp_top_198_i
1556                            XORI    $9 1
1557 cmp_top_198_i:             BEQ     $8 $0
        cmp_bot_199_i
1558                            ADD     $7 $3
1559                            ADDI    $7 2
1560                            EXCH    $8 $7
1561                            ADDI    $7 -2
1562                            SUB     $7 $3
1563 test_194:                   BEQ     $6 $0
        test_false_196
1564                            XORI    $6 1
1565                            ADD     $7 $3
1566                            ADDI    $7 2
1567                            EXCH    $8 $7
1568                            ADDI    $7 -2
1569                            SUB     $7 $3
```

```
1570                            XOR     $9 $8
1571 loadMetAdd_202:           EXCH    $10 $9
1572                            ADDI    $10 3
1573                            EXCH    $11 $10
1574                            XOR     $12 $11
1575                            EXCH    $11 $10
1576                            ADDI    $10 -3
1577                            EXCH    $10 $9
1578                            ADD     $7 $3
1579                            ADDI    $7 2
1580                            EXCH    $8 $7
1581                            ADDI    $7 -2
1582                            SUB     $7 $3
1583                            EXCH    $3 $1
1584                            ADDI    $1 -1
1585                            EXCH    $9 $1
1586                            ADDI    $1 -1
1587                            ADDI    $12 -1586
1588 l_jmp_203:                SWAPBR  $12
1589                            NEG     $12
1590                            ADDI    $12 1586
1591                            ADDI    $1 1
1592                            EXCH    $9 $1
1593                            ADDI    $1 1
1594                            EXCH    $3 $1
1595                            ADD     $7 $3
1596                            ADDI    $7 2
1597                            EXCH    $8 $7
1598                            ADDI    $7 -2
1599                            SUB     $7 $3
1600                            EXCH    $10 $9
1601                            ADDI    $10 3
1602                            EXCH    $11 $10
1603                            XOR     $12 $11
1604                            EXCH    $11 $10
1605                            ADDI    $10 -3
1606 loadMetAdd_202_i:         EXCH    $10 $9
1607                            XOR     $9 $8
1608                            ADD     $7 $3
1609                            ADDI    $7 2
1610                            EXCH    $8 $7
1611                            ADDI    $7 -2
1612                            SUB     $7 $3
1613                            XORI    $6 1
1614 assert_true_195:          BRA     assert_197
1615 test_false_196:           BRA     test_194
1616 assert_197:                BNE     $6 $0
         assert_true_195
1617                            ADD     $7 $3
1618                            ADDI    $7 2
1619                            EXCH    $8 $7
1620                            ADDI    $7 -2
1621                            SUB     $7 $3
1622 cmp_top_204:               BEQ     $8 $0
         cmp_bot_205
1623                            XORI    $9 1
1624 cmp_bot_205:               BEQ     $8 $0
         cmp_top_204
1625 f_top_206:                 BEQ     $9 $0
         f_bot_207
1626                            XORI    $10 1
1627 f_bot_207:                 BEQ     $9 $0
         f_top_206
1628                            XOR     $6 $10
1629 f_bot_207_i:               BEQ     $9 $0
         f_top_206_i
1630                            XORI    $10 1
1631 f_top_206_i:               BEQ     $9 $0
         f_bot_207_i
1632 cmp_bot_205_i:            BEQ     $8 $0
         cmp_top_204_i
1633                            XORI    $9 1
1634 cmp_top_204_i:            BEQ     $8 $0
         cmp_bot_205_i
1635                            ADD     $7 $3
1636                            ADDI    $7 2
1637                            EXCH    $8 $7
1638                            ADDI    $7 -2
1639                            SUB     $7 $3
1640 l_mirror_3_bot:           BRA     l_mirror_3_top
1641 l_main_0_top:             BRA     l_main_0_bot
1642                            ADDI    $1 1
1643                            EXCH    $2 $1
1644                            EXCH    $3 $1
1645                            ADDI    $1 -1
1646 l_main_0:                  SWAPBR  $2
1647                            NEG     $2
1648                            ADDI    $1 1
1649                            EXCH    $3 $1
1650                            EXCH    $2 $1
1651                            ADDI    $1 -1
1652                            EXCH    $3 $1
1653                            ADDI    $1 -1
1654 obj_con_208:               ADDI    $8 4
1655                            EXCH    $8 $1
1656                            ADDI    $1 -1
1657                            EXCH    $7 $1
1658                            ADDI    $1 -1
1659                            BRA     l_malloc
1660                            ADDI    $1 1
1661                            EXCH    $7 $1
1662                            ADDI    $1 1
1663                            EXCH    $8 $1
1664 obj_con_208_i:             ADDI    $8 -4
1665                            ADDI    $1 1
1666                            EXCH    $3 $1
1667                            ADD     $6 $3
1668                            ADDI    $6 3
1669                            XORI    $8 5
1670                            EXCH    $8 $7
1671                            ADDI    $7 1
1672                            XORI    $8 1
1673                            EXCH    $8 $7
1674 obj_con_208_bot:           ADDI    $7 -1
1675                            EXCH    $7 $6
1676                            ADDI    $6 -3
1677                            SUB     $6 $3
1678                            ADD     $6 $3
1679                            ADDI    $6 4
1680                            EXCH    $7 $6
1681                            ADDI    $6 -4
1682                            SUB     $6 $3
1683                            XORI    $8 3
1684                            ADD     $7 $8
1685                            XORI    $8 3
1686                            ADD     $6 $3
1687                            ADDI    $6 4
1688                            EXCH    $7 $6
1689                            ADDI    $6 -4
1690                            SUB     $6 $3
1691 localBlock_227:            XOR     $6 $1
1692                            XOR     $7 $0
```

```
                        EXCH    $7 $1
                        ADDI    $1 -1
                        XORI    $7 1
entry_209:              BEQ     $7 $0
    assert_211
                        EXCH    $8 $6
cmp_top_213:            BNE     $8 $0
    cmp_bot_214
                        XORI    $9 1
cmp_bot_214:            BNE     $8 $0
    cmp_top_213
f_top_215:              BEQ     $9 $0
    f_bot_216
                        XORI    $10 1
f_bot_216:              BEQ     $9 $0
    f_top_215
                        XOR     $7 $10
f_bot_216_i:            BEQ     $9 $0
    f_top_215_i
                        XORI    $10 1
f_top_215_i:            BEQ     $9 $0
    f_bot_216_i
cmp_bot_214_i:          BNE     $8 $0
    cmp_top_213_i
                        XORI    $9 1
cmp_top_213_i:          BNE     $8 $0
    cmp_bot_214_i
                        EXCH    $8 $6
                        EXCH    $8 $6
                        ADD     $9 $3
                        ADDI    $9 4
                        EXCH    $10 $9
                        ADDI    $9 -4
                        SUB     $9 $3
cmp_top_223:            BNE     $8 $10
    cmp_bot_224
                        XORI    $11 1
cmp_bot_224:            BNE     $8 $10
    cmp_top_223
f_top_225:              BEQ     $11 $0
    f_bot_226
                        XORI    $12 1
f_bot_226:              BEQ     $11 $0
    f_top_225
                        XOR     $7 $12
f_bot_226_i:            BEQ     $11 $0
    f_top_225_i
                        XORI    $12 1
f_top_225_i:            BEQ     $11 $0
    f_bot_226_i
cmp_bot_224_i:          BNE     $8 $10
    cmp_top_223_i
                        XORI    $11 1
cmp_top_223_i:          BNE     $8 $10
    cmp_bot_224_i
                        ADD     $9 $3
                        ADDI    $9 4
                        EXCH    $10 $9
                        ADDI    $9 -4
                        SUB     $9 $3
                        EXCH    $8 $6
test_210:               BNE     $7 $0 exit_218
localBlock_222:         XOR     $8 $1
                        XOR     $9 $0
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
obj_con_217:            ADDI    $10 8
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        BRA     l_malloc
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
obj_con_217_i:          ADDI    $10 -8
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        XORI    $10 8
                        EXCH    $10 $9
                        ADDI    $9 1
                        XORI    $10 1
                        EXCH    $10 $9
obj_con_217_bot:        ADDI    $9 -1
                        EXCH    $9 $8
                        EXCH    $9 $8
                        XOR     $10 $9
loadMetAdd_218:         EXCH    $11 $10
                        ADDI    $11 0
                        EXCH    $12 $11
                        XOR     $13 $12
                        EXCH    $12 $11
                        ADDI    $11 0
                        EXCH    $11 $10
                        EXCH    $9 $8
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        ADDI    $13 -1789
l_jmp_219:              SWAPBR  $13
                        NEG     $13
                        ADDI    $13 1789
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $9 $8
                        EXCH    $11 $10
                        ADDI    $11 0
                        EXCH    $12 $11
                        XOR     $13 $12
                        EXCH    $12 $11
```

```
                        ADDI   $11 0
loadMetAdd_218_i:       EXCH   $11 $10
                        XOR    $10 $9
                        EXCH   $9 $8
                        ADD    $9 $3
                        ADDI   $9 3
                        EXCH   $10 $9
                        ADDI   $9 -3
                        SUB    $9 $3
                        XOR    $11 $10
loadMetAdd_220:         EXCH   $12 $11
                        ADDI   $12 0
                        EXCH   $13 $12
                        XOR    $14 $13
                        EXCH   $13 $12
                        ADDI   $12 0
                        EXCH   $12 $11
                        ADD    $9 $3
                        ADDI   $9 3
                        EXCH   $10 $9
                        ADDI   $9 -3
                        SUB    $9 $3
                        EXCH   $3 $1
                        ADDI   $1 -1
                        EXCH   $8 $1
                        ADDI   $1 -1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $11 $1
                        ADDI   $1 -1
                        ADDI   $14 -1837
l_jmp_221:              SWAPBR $14
                        NEG    $14
                        ADDI   $14 1837
                        ADDI   $1 1
                        EXCH   $11 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        ADDI   $1 1
                        EXCH   $8 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        ADD    $9 $3
                        ADDI   $9 3
                        EXCH   $10 $9
                        ADDI   $9 -3
                        SUB    $9 $3
                        EXCH   $12 $11
                        ADDI   $12 0
                        EXCH   $13 $12
                        XOR    $14 $13
                        EXCH   $13 $12
                        ADDI   $12 0
loadMetAdd_220_i:       EXCH   $12 $11
                        XOR    $11 $10
                        ADD    $9 $3
                        ADDI   $9 3
                        EXCH   $10 $9
                        ADDI   $9 -3
                        SUB    $9 $3
                        ADDI   $1 1
                        EXCH   $9 $1
                        XOR    $9 $0
localBlock_222_i:       XOR    $8 $1
                        EXCH   $8 $6
                        XORI   $9 1
                        ADD    $8 $9
                        XORI   $9 1
                        EXCH   $8 $6
assert_211:             BRA    entry_209
exit_212:               BRA    test_210
                        XORI   $7 1
                        ADD    $7 $3
                        ADDI   $7 4
                        EXCH   $8 $7
                        ADDI   $7 -4
                        SUB    $7 $3
                        ADDI   $1 1
                        EXCH   $9 $1
                        XOR    $9 $8
localBlock_227_i:       XOR    $6 $1
                        ADD    $7 $3
                        ADDI   $7 4
                        EXCH   $8 $7
                        ADDI   $7 -4
                        SUB    $7 $3
                        ADD    $6 $3
                        ADDI   $6 3
                        EXCH   $7 $6
                        ADDI   $6 -3
                        SUB    $6 $3
                        XOR    $8 $7
loadMetAdd_228:         EXCH   $9 $8
                        ADDI   $9 1
                        EXCH   $10 $9
                        XOR    $11 $10
                        EXCH   $10 $9
                        ADDI   $9 -1
                        EXCH   $9 $8
                        ADD    $6 $3
                        ADDI   $6 3
                        EXCH   $7 $6
                        ADDI   $6 -3
                        SUB    $6 $3
                        ADD    $12 $3
                        ADDI   $12 2
                        EXCH   $3 $1
                        ADDI   $1 -1
                        EXCH   $12 $1
                        ADDI   $1 -1
                        EXCH   $8 $1
                        ADDI   $1 -1
                        ADDI   $11 -1919
l_jmp_229:              SWAPBR $11
                        NEG    $11
                        ADDI   $11 1919
                        ADDI   $1 1
                        EXCH   $8 $1
                        ADDI   $1 1
                        EXCH   $12 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        ADDI   $12 -2
                        SUB    $12 $3
                        ADD    $6 $3
                        ADDI   $6 3
                        EXCH   $7 $6
                        ADDI   $6 -3
                        SUB    $6 $3
                        EXCH   $9 $8
                        ADDI   $9 1
                        EXCH   $10 $9
```

```
                        XOR     $11 $10
                        EXCH    $10 $9
                        ADDI    $9 -1
loadMetAdd_228_i:       EXCH    $9 $8
                        XOR     $8 $7
                        ADD     $6 $3
                        ADDI    $6 3
                        EXCH    $7 $6
                        ADDI    $6 -3
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 3
                        EXCH    $7 $6
                        ADDI    $6 -3
                        SUB     $6 $3
                        XOR     $8 $7
loadMetAdd_230:         EXCH    $9 $8
                        ADDI    $9 2
                        EXCH    $10 $9
                        XOR     $11 $10
                        EXCH    $10 $9
                        ADDI    $9 -2
                        EXCH    $9 $8
                        ADD     $6 $3
                        ADDI    $6 3
                        EXCH    $7 $6
                        ADDI    $6 -3
                        SUB     $6 $3
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        ADDI    $11 -1971
l_jmp_231:              SWAPBR  $11
                        NEG     $11
                        ADDI    $11 1971
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADD     $6 $3
                        ADDI    $6 3
                        EXCH    $7 $6
                        ADDI    $6 -3
                        SUB     $6 $3
                        EXCH    $9 $8
                        ADDI    $9 2
                        EXCH    $10 $9
                        XOR     $11 $10
                        EXCH    $10 $9
                        ADDI    $9 -2
loadMetAdd_230_i:       EXCH    $9 $8
                        XOR     $8 $7
                        ADD     $6 $3
                        ADDI    $6 3
                        EXCH    $7 $6
                        ADDI    $6 -3
                        SUB     $6 $3
l_main_0_bot:           BRA     l_main_0_top
start:                  BRA     top
                        START
                        ADDI    $4 2055
                        XOR     $5 $4
                        ADDI    $5 10
                        XOR     $7 $5
                        ADDI    $4 10
                        ADDI    $4 -1
                        EXCH    $7 $4
                        ADDI    $4 1
                        ADDI    $4 -10
                        XOR     $1 $5
                        ADDI    $1 2048
                        ADDI    $1 -8
                        XOR     $3 $1
                        XORI    $6 4
                        EXCH    $6 $3
                        ADDI    $1 -1
                        EXCH    $3 $1
                        ADDI    $1 -1
                        BRA     l_main_0
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $3 1
                        ADDI    $3 1
                        EXCH    $6 $3
                        XORI    $7 1
                        EXCH    $6 $7
                        XORI    $7 1
                        ADDI    $3 -1
                        ADDI    $3 -1
                        ADDI    $3 1
                        ADDI    $3 2
                        EXCH    $6 $3
                        XORI    $7 2
                        EXCH    $6 $7
                        XORI    $7 2
                        ADDI    $3 -2
                        ADDI    $3 -1
                        ADDI    $3 1
                        ADDI    $3 3
                        EXCH    $6 $3
                        XORI    $7 3
                        EXCH    $6 $7
                        XORI    $7 3
                        ADDI    $3 -3
                        ADDI    $3 -1
                        ADDI    $1 1
                        EXCH    $6 $3
                        XORI    $6 4
                        XOR     $3 $1
                        ADDI    $1 8
                        ADDI    $1 -2048
                        XOR     $1 $5
                        ADDI    $5 -10
                        XOR     $5 $4
                        ADDI    $4 -2055
finish:                 FINISH
```

DoublyLinkedList.rplpp

```
class Cell
    int data
    int index
    Cell left
    Cell right
    Cell self

    method setData(int value)
        data ^= value

    method setIndex(int i)
        index ^= i

    method setLeft(Cell cell)
        left <=> cell

    method setRight(Cell cell)
        right <=> cell

    method setSelf(Cell cell)
        self <=> cell

    method append(Cell cell)
        if right = nil & cell != nil then   // If current cell does not have a right neighbour
            right <=> cell                  // Set new cell as right neighbour of current cell

            local Cell selfCopy = nil
            copy Cell self selfCopy         // Copy reference to current cell
            call right::setLeft(selfCopy)   // Set current cell as left neighbour of newly
                added right neighbour
            delocal Cell selfCopy = nil

            local int cellIndex = index + 1
            call right::setIndex(cellIndex) // Set cell index in newly added right neightbour
                of current cell
            delocal int cellIndex = index + 1
        else skip
        fi right != nil & cell = nil

        if right != nil then
            call right::append(cell)        // Keep searching for empty right neighbour
        else skip
        fi right != nil

class DoublyLinkedList
    Cell head
    int length

    method appendCell(Cell cell)
        if head = nil & cell != nil then
            head <=> cell
        else skip
        fi head != nil & cell = nil

        if head != nil then
            call head::append(cell)
        else skip
        fi head != nil

        length += 1

class Program
    DoublyLinkedList list
```

```
62      int listLength
63
64      method main()
65          new DoublyLinkedList list
66          listLength += 10
67
68          local int x = 0
69          from x = 0 do skip
70          loop
71              local Cell cell = nil
72                  new Cell cell
73
74                  local Cell cellCopy = nil
75                  copy Cell cell cellCopy
76                  call cell::setSelf(cellCopy)
77                  delocal Cell cellCopy = nil
78
79                  call cell::setData(x)
80                  call list::appendCell(cell)
81              delocal Cell cell = nil
82              x += 1
83          until x = listLength
84          delocal int x = listLength
```

DoublyLinkedList.pal

```
;; pendulum pal file
top:                    BRA     start
l_r_list:               DATA    0
l_r_listLength:         DATA    0
l_Program_vt:           DATA    977
l_DoublyLinkedList_vt:  DATA    759
l_Cell_vt:              DATA    223
                        DATA    252
                        DATA    281
                        DATA    312
                        DATA    343
                        DATA    374
l_malloc_top:           BRA     l_malloc_bot
l_malloc:               SWAPBR  $2
                        NEG     $2
                        ADDI    $9 2
                        XOR     $8 $0
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
                        BRA     l_malloc1
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        XOR     $8 $0
                        ADDI    $9 -2
l_malloc_bot:           BRA     l_malloc_top
l_malloc1_top:          BRA     l_malloc1_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        SUB     $17 $8
                        XOR     $17 $4
l_malloc1:              SWAPBR  $2
                        NEG     $2
                        EXCH    $2 $1
                        ADDI    $1 -1
                        XOR     $17 $4
                        ADD     $17 $8
                        EXCH    $19 $17
                        XOR     $18 $19
                        EXCH    $19 $17
                        XOR     $13 $9
                        SUB     $13 $7
cmp_top_8:              BGEZ    $13 cmp_bot_9
                        XORI    $14 1
cmp_bot_9:              BGEZ    $13 cmp_top_8
                        XOR     $10 $14
cmp_bot_9_i:            BGEZ    $13
    cmp_top_8_i
                        XORI    $14 1
cmp_top_8_i:            BGEZ    $13
    cmp_bot_9_i
                        ADD     $13 $7
                        XOR     $13 $9
l_o_test:               BEQ     $10 $0
    l_o_test_false
                        XORI    $10 1
                        ADDI    $8 1
                        EXCH    $19 $17
                        XOR     $18 $19
                        EXCH    $19 $17
                        RL      $9 1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $11 $1
                        ADDI    $1 -1
                        EXCH    $12 $1
                        ADDI    $1 -1
                        EXCH    $14 $1
                        ADDI    $1 -1
                        EXCH    $16 $1
                        ADDI    $1 -1
                        EXCH    $17 $1
                        ADDI    $1 -1
                        EXCH    $18 $1
                        ADDI    $1 -1
                        EXCH    $20 $1
                        ADDI    $1 -1
                        EXCH    $21 $1
                        ADDI    $1 -1
                        EXCH    $22 $1
                        ADDI    $1 -1
                        EXCH    $23 $1
                        ADDI    $1 -1
                        BRA     l_malloc1
                        ADDI    $1 1
                        EXCH    $23 $1
                        ADDI    $1 1
                        EXCH    $22 $1
                        ADDI    $1 1
                        EXCH    $21 $1
                        ADDI    $1 1
                        EXCH    $20 $1
                        ADDI    $1 1
                        EXCH    $18 $1
                        ADDI    $1 1
                        EXCH    $17 $1
                        ADDI    $1 1
                        EXCH    $16 $1
                        ADDI    $1 1
                        EXCH    $14 $1
                        ADDI    $1 1
                        EXCH    $12 $1
                        ADDI    $1 1
                        EXCH    $11 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        RR      $9 1
                        ADDI    $8 -1
                        XORI    $10 1
l_o_assert_true:        BRA     l_o_assert
l_o_test_false:         BRA     l_o_test
cmp_top_12:             BEQ     $18 $0
    cmp_bot_13
                        XORI    $20 1
cmp_bot_13:             BEQ     $18 $0
    cmp_top_12
                        XOR     $11 $20
```

```
cmp_bot_13_i:           BEQ    $18 $0
    cmp_top_12_i
                        XORI   $20 1
cmp_top_12_i:           BEQ    $18 $0
    cmp_bot_13_i
l_i_test:               BEQ    $11 $0
    l_i_test_false
                        XORI   $11 1
                        ADD    $6 $18
                        SUB    $18 $6
                        EXCH   $12 $6
                        EXCH   $12 $17
                        XOR    $12 $6
                        XORI   $11 1
l_i_assert_true:        BRA    l_i_assert
l_i_test_false:         BRA    l_i_test
                        ADDI   $8 1
                        RL     $9 1
                        EXCH   $10 $1
                        ADDI   $1 -1
                        EXCH   $11 $1
                        ADDI   $1 -1
                        EXCH   $12 $1
                        ADDI   $1 -1
                        EXCH   $14 $1
                        ADDI   $1 -1
                        EXCH   $16 $1
                        ADDI   $1 -1
                        EXCH   $17 $1
                        ADDI   $1 -1
                        EXCH   $18 $1
                        ADDI   $1 -1
                        EXCH   $20 $1
                        ADDI   $1 -1
                        EXCH   $21 $1
                        ADDI   $1 -1
                        EXCH   $22 $1
                        ADDI   $1 -1
                        EXCH   $23 $1
                        ADDI   $1 -1
                        BRA    l_malloc1
                        ADDI   $1 1
                        EXCH   $23 $1
                        ADDI   $1 1
                        EXCH   $22 $1
                        ADDI   $1 1
                        EXCH   $21 $1
                        ADDI   $1 1
                        EXCH   $20 $1
                        ADDI   $1 1
                        EXCH   $18 $1
                        ADDI   $1 1
                        EXCH   $17 $1
                        ADDI   $1 1
                        EXCH   $16 $1
                        ADDI   $1 1
                        EXCH   $14 $1
                        ADDI   $1 1
                        EXCH   $12 $1
                        ADDI   $1 1
                        EXCH   $11 $1
                        ADDI   $1 1
                        EXCH   $10 $1
                        RR     $9 1
                        ADDI   $8 -1
                        XOR    $12 $6
                        EXCH   $12 $17
                        ADD    $6 $9
l_i_assert:             BNE    $11 $0
    l_i_assert_true
                        EXCH   $12 $17
                        SUB    $6 $9
cmp_top_14:             BEQ    $6 $12
    cmp_bot_15
                        XORI   $21 1
cmp_bot_15:             BEQ    $6 $12
    cmp_top_14
cmp_top_16:             BNE    $12 $0
    cmp_bot_17
                        XORI   $22 1
cmp_bot_17:             BNE    $12 $0
    cmp_top_16
                        ORX    $23 $21 $22
                        XOR    $11 $23
                        ORX    $23 $21 $22
cmp_bot_17_i:           BNE    $12 $0
    cmp_top_16_i
                        XORI   $22 1
cmp_top_16_i:           BNE    $12 $0
    cmp_bot_17_i
cmp_bot_15_i:           BEQ    $6 $12
    cmp_top_14_i
                        XORI   $21 1
cmp_top_14_i:           BEQ    $6 $12
    cmp_bot_15_i
                        ADD    $6 $9
                        EXCH   $12 $17
l_o_assert:             BNE    $10 $0
    l_o_assert_true
                        XOR    $15 $9
                        SUB    $15 $7
cmp_top_10:             BGEZ   $15 cmp_bot_11
                        XORI   $16 1
cmp_bot_11:             BGEZ   $15 cmp_top_10
                        XOR    $10 $16
cmp_bot_11_i:           BGEZ   $15
    cmp_top_10_i
                        XORI   $16 1
cmp_top_10_i:           BGEZ   $15
    cmp_bot_11_i
                        ADD    $15 $7
                        XOR    $15 $9
l_malloc1_bot:          BRA    l_malloc1_top
l_setData_2_top:        BRA
    l_setData_2_bot
                        ADDI   $1 1
                        EXCH   $2 $1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $3 $1
                        ADDI   $1 -1
l_setData_2:            SWAPBR $2
                        NEG    $2
                        ADDI   $1 1
                        EXCH   $3 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        EXCH   $2 $1
                        ADDI   $1 -1
                        ADD    $7 $3
                        ADDI   $7 2
                        EXCH   $8 $7
```

```
236                         ADDI    $7 -2
237                         SUB     $7 $3
238                         EXCH    $9 $6
239                         XOR     $8 $9
240                         EXCH    $9 $6
241                         ADD     $7 $3
242                         ADDI    $7 2
243                         EXCH    $8 $7
244                         ADDI    $7 -2
245                         SUB     $7 $3
246 l_setData_2_bot:        BRA
        l_setData_2_top
247 l_setIndex_3_top:       BRA
        l_setIndex_3_bot
248                         ADDI    $1 1
249                         EXCH    $2 $1
250                         EXCH    $6 $1
251                         ADDI    $1 -1
252                         EXCH    $3 $1
253                         ADDI    $1 -1
254 l_setIndex_3:           SWAPBR  $2
255                         NEG     $2
256                         ADDI    $1 1
257                         EXCH    $3 $1
258                         ADDI    $1 1
259                         EXCH    $6 $1
260                         EXCH    $2 $1
261                         ADDI    $1 -1
262                         ADD     $7 $3
263                         ADDI    $7 3
264                         EXCH    $8 $7
265                         ADDI    $7 -3
266                         SUB     $7 $3
267                         EXCH    $9 $6
268                         XOR     $8 $9
269                         EXCH    $9 $6
270                         ADD     $7 $3
271                         ADDI    $7 3
272                         EXCH    $8 $7
273                         ADDI    $7 -3
274                         SUB     $7 $3
275 l_setIndex_3_bot:       BRA
        l_setIndex_3_top
276 l_setLeft_4_top:        BRA
        l_setLeft_4_bot
277                         ADDI    $1 1
278                         EXCH    $2 $1
279                         EXCH    $6 $1
280                         ADDI    $1 -1
281                         EXCH    $3 $1
282                         ADDI    $1 -1
283 l_setLeft_4:            SWAPBR  $2
284                         NEG     $2
285                         ADDI    $1 1
286                         EXCH    $3 $1
287                         ADDI    $1 1
288                         EXCH    $6 $1
289                         EXCH    $2 $1
290                         ADDI    $1 -1
291                         ADD     $7 $3
292                         ADDI    $7 4
293                         EXCH    $8 $7
294                         ADDI    $7 -4
295                         SUB     $7 $3
296                         EXCH    $9 $6
297 swap_18:                XOR     $8 $9
298                         XOR     $9 $8
299                         XOR     $8 $9
300                         EXCH    $9 $6
301                         ADD     $7 $3
302                         ADDI    $7 4
303                         EXCH    $8 $7
304                         ADDI    $7 -4
305                         SUB     $7 $3
306 l_setLeft_4_bot:        BRA
        l_setLeft_4_top
307 l_setRight_5_top:       BRA
        l_setRight_5_bot
308                         ADDI    $1 1
309                         EXCH    $2 $1
310                         EXCH    $6 $1
311                         ADDI    $1 -1
312                         EXCH    $3 $1
313                         ADDI    $1 -1
314 l_setRight_5:           SWAPBR  $2
315                         NEG     $2
316                         ADDI    $1 1
317                         EXCH    $3 $1
318                         ADDI    $1 1
319                         EXCH    $6 $1
320                         EXCH    $2 $1
321                         ADDI    $1 -1
322                         ADD     $7 $3
323                         ADDI    $7 5
324                         EXCH    $8 $7
325                         ADDI    $7 -5
326                         SUB     $7 $3
327                         EXCH    $9 $6
328 swap_19:                XOR     $8 $9
329                         XOR     $9 $8
330                         XOR     $8 $9
331                         EXCH    $9 $6
332                         ADD     $7 $3
333                         ADDI    $7 5
334                         EXCH    $8 $7
335                         ADDI    $7 -5
336                         SUB     $7 $3
337 l_setRight_5_bot:       BRA
        l_setRight_5_top
338 l_setSelf_6_top:        BRA
        l_setSelf_6_bot
339                         ADDI    $1 1
340                         EXCH    $2 $1
341                         EXCH    $6 $1
342                         ADDI    $1 -1
343                         EXCH    $3 $1
344                         ADDI    $1 -1
345 l_setSelf_6:            SWAPBR  $2
346                         NEG     $2
347                         ADDI    $1 1
348                         EXCH    $3 $1
349                         ADDI    $1 1
350                         EXCH    $6 $1
351                         EXCH    $2 $1
352                         ADDI    $1 -1
353                         ADD     $7 $3
354                         ADDI    $7 6
355                         EXCH    $8 $7
356                         ADDI    $7 -6
357                         SUB     $7 $3
358                         EXCH    $9 $6
359 swap_20:                XOR     $8 $9
```

```
                        XOR     $9 $8
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADD     $7 $3
                        ADDI    $7 6
                        EXCH    $8 $7
                        ADDI    $7 -6
                        SUB     $7 $3
l_setSelf_6_bot:        BRA
    l_setSelf_6_top
l_append_7_top:         BRA     l_append_7_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $3 $1
                        ADDI    $1 -1
l_append_7:             SWAPBR  $2
                        NEG     $2
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
                        ADD     $8 $3
                        ADDI    $8 5
                        EXCH    $9 $8
                        ADDI    $8 -5
                        SUB     $8 $3
cmp_top_25:             BNE     $9 $0
    cmp_bot_26
                        XORI    $10 1
cmp_bot_26:             BNE     $9 $0
    cmp_top_25
                        EXCH    $11 $6
cmp_top_27:             BEQ     $11 $0
    cmp_bot_28
                        XORI    $12 1
cmp_bot_28:             BEQ     $11 $0
    cmp_top_27
                        ANDX    $13 $10 $12
f_top_29:               BEQ     $13 $0
    f_bot_30
                        XORI    $14 1
f_bot_30:               BEQ     $13 $0
    f_top_29
                        XOR     $7 $14
f_bot_30_i:             BEQ     $13 $0
    f_top_29_i
                        XORI    $14 1
f_top_29_i:             BEQ     $13 $0
    f_bot_30_i
                        ANDX    $13 $10 $12
cmp_bot_28_i:           BEQ     $11 $0
    cmp_top_27_i
                        XORI    $12 1
cmp_top_27_i:           BEQ     $11 $0
    cmp_bot_28_i
                        EXCH    $11 $6
cmp_bot_26_i:           BNE     $9 $0
    cmp_top_25_i
                        XORI    $10 1
cmp_top_25_i:           BNE     $9 $0
    cmp_bot_26_i
                        ADD     $8 $3
                        ADDI    $8 5
                        EXCH    $9 $8
                        ADDI    $8 -5
                        SUB     $8 $3
test_21:                BEQ     $7 $0
    test_false_23
                        XORI    $7 1
                        ADD     $8 $3
                        ADDI    $8 5
                        EXCH    $9 $8
                        ADDI    $8 -5
                        SUB     $8 $3
                        EXCH    $10 $6
swap_31:                XOR     $9 $10
                        XOR     $10 $9
                        XOR     $9 $10
                        EXCH    $10 $6
                        ADD     $8 $3
                        ADDI    $8 5
                        EXCH    $9 $8
                        ADDI    $8 -5
                        SUB     $8 $3
localBlock_35:          XOR     $8 $1
                        XOR     $9 $0
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $9 $8
                        ADD     $10 $3
                        ADDI    $10 6
                        EXCH    $11 $10
                        ADDI    $10 -6
                        SUB     $10 $3
copy_32:                XOR     $9 $11
                        ADDI    $11 1
                        EXCH    $12 $11
                        ADDI    $12 1
                        EXCH    $12 $11
                        ADDI    $11 -1
                        ADD     $10 $3
                        ADDI    $10 6
                        EXCH    $11 $10
                        ADDI    $10 -6
                        SUB     $10 $3
                        EXCH    $9 $8
                        ADD     $9 $3
                        ADDI    $9 5
                        EXCH    $10 $9
                        ADDI    $9 -5
                        SUB     $9 $3
                        XOR     $11 $10
loadMetAdd_33:          EXCH    $12 $11
                        ADDI    $12 2
                        EXCH    $13 $12
                        XOR     $14 $13
                        EXCH    $13 $12
                        ADDI    $12 -2
                        EXCH    $12 $11
                        ADD     $9 $3
                        ADDI    $9 5
                        EXCH    $10 $9
                        ADDI    $9 -5
                        SUB     $9 $3
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
```

```
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $11 $1
                        ADDI    $1 -1
                        ADDI    $14 -481
l_jmp_34:               SWAPBR  $14
                        NEG     $14
                        ADDI    $14 481
                        ADDI    $1 1
                        EXCH    $11 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADD     $9 $3
                        ADDI    $9 5
                        EXCH    $10 $9
                        ADDI    $9 -5
                        SUB     $9 $3
                        EXCH    $12 $11
                        ADDI    $12 2
                        EXCH    $13 $12
                        XOR     $14 $13
                        EXCH    $13 $12
                        ADDI    $12 -2
loadMetAdd_33_i:        EXCH    $12 $11
                        XOR     $11 $10
                        ADD     $9 $3
                        ADDI    $9 5
                        EXCH    $10 $9
                        ADDI    $9 -5
                        SUB     $9 $3
                        ADDI    $1 1
                        EXCH    $9 $1
                        XOR     $9 $0
localBlock_35_i:        XOR     $8 $1
                        ADD     $9 $3
                        ADDI    $9 3
                        EXCH    $10 $9
                        ADDI    $9 -3
                        SUB     $9 $3
                        XORI    $11 1
                        XOR     $12 $10
                        ADD     $12 $11
localBlock_38:          XOR     $8 $1
                        XOR     $13 $12
                        EXCH    $13 $1
                        ADDI    $1 -1
                        SUB     $12 $11
                        XOR     $12 $10
                        XORI    $11 1
                        ADD     $9 $3
                        ADDI    $9 3
                        EXCH    $10 $9
                        ADDI    $9 -3
                        SUB     $9 $3
                        ADD     $9 $3
                        ADDI    $9 5
                        EXCH    $10 $9
                        ADDI    $9 -5
                        SUB     $9 $3
                        XOR     $11 $10
loadMetAdd_36:          EXCH    $12 $11
                        ADDI    $12 1
                        EXCH    $13 $12
                        XOR     $14 $13
                        EXCH    $13 $12
                        ADDI    $12 -1
                        EXCH    $12 $11
                        ADD     $9 $3
                        ADDI    $9 5
                        EXCH    $10 $9
                        ADDI    $9 -5
                        SUB     $9 $3
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $11 $1
                        ADDI    $1 -1
                        ADDI    $14 -561
l_jmp_37:               SWAPBR  $14
                        NEG     $14
                        ADDI    $14 561
                        ADDI    $1 1
                        EXCH    $11 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADD     $9 $3
                        ADDI    $9 5
                        EXCH    $10 $9
                        ADDI    $9 -5
                        SUB     $9 $3
                        EXCH    $12 $11
                        ADDI    $12 1
                        EXCH    $13 $12
                        XOR     $14 $13
                        EXCH    $13 $12
                        ADDI    $12 -1
loadMetAdd_36_i:        EXCH    $12 $11
                        XOR     $11 $10
                        ADD     $9 $3
                        ADDI    $9 5
                        EXCH    $10 $9
                        ADDI    $9 -5
                        SUB     $9 $3
                        ADD     $9 $3
                        ADDI    $9 3
                        EXCH    $10 $9
                        ADDI    $9 -3
                        SUB     $9 $3
                        XORI    $11 1
                        XOR     $12 $10
                        ADD     $12 $11
                        ADDI    $1 1
                        EXCH    $13 $1
                        XOR     $13 $12
localBlock_38_i:        XOR     $8 $1
                        SUB     $12 $11
                        XOR     $12 $10
                        XORI    $11 1
                        ADD     $9 $3
                        ADDI    $9 3
                        EXCH    $10 $9
```

```
610                         ADDI    $9 -3
611                         SUB     $9 $3
612                         XORI    $7 1
613 assert_true_22:         BRA     assert_24
614 test_false_23:          BRA     test_21
615 assert_24:              BNE     $7 $0
        assert_true_22
616                         ADD     $8 $3
617                         ADDI    $8 5
618                         EXCH    $9 $8
619                         ADDI    $8 -5
620                         SUB     $8 $3
621 cmp_top_39:             BEQ     $9 $0
        cmp_bot_40
622                         XORI    $10 1
623 cmp_bot_40:             BEQ     $9 $0
        cmp_top_39
624                         EXCH    $11 $6
625 cmp_top_41:             BNE     $11 $0
        cmp_bot_42
626                         XORI    $12 1
627 cmp_bot_42:             BNE     $11 $0
        cmp_top_41
628                         ANDX    $13 $10 $12
629 f_top_43:               BEQ     $13 $0
        f_bot_44
630                         XORI    $14 1
631 f_bot_44:               BEQ     $13 $0
        f_top_43
632                         XOR     $7 $14
633 f_bot_44_i:             BEQ     $13 $0
        f_top_43_i
634                         XORI    $14 1
635 f_top_43_i:             BEQ     $13 $0
        f_bot_44_i
636                         ANDX    $13 $10 $12
637 cmp_bot_42_i:           BNE     $11 $0
        cmp_top_41_i
638                         XORI    $12 1
639 cmp_top_41_i:           BNE     $11 $0
        cmp_bot_42_i
640                         EXCH    $11 $6
641 cmp_bot_40_i:           BEQ     $9 $0
        cmp_top_39_i
642                         XORI    $10 1
643 cmp_top_39_i:           BEQ     $9 $0
        cmp_bot_40_i
644                         ADD     $8 $3
645                         ADDI    $8 5
646                         EXCH    $9 $8
647                         ADDI    $8 -5
648                         SUB     $8 $3
649                         ADD     $8 $3
650                         ADDI    $8 5
651                         EXCH    $9 $8
652                         ADDI    $8 -5
653                         SUB     $8 $3
654 cmp_top_49:             BEQ     $9 $0
        cmp_bot_50
655                         XORI    $10 1
656 cmp_bot_50:             BEQ     $9 $0
        cmp_top_49
657 f_top_51:               BEQ     $10 $0
        f_bot_52
658                         XORI    $11 1
659 f_bot_52:               BEQ     $10 $0
        f_top_51
660                         XOR     $7 $11
661 f_bot_52_i:             BEQ     $10 $0
        f_top_51_i
662                         XORI    $11 1
663 f_top_51_i:             BEQ     $10 $0
        f_bot_52_i
664 cmp_bot_50_i:           BEQ     $9 $0
        cmp_top_49_i
665                         XORI    $10 1
666 cmp_top_49_i:           BEQ     $9 $0
        cmp_bot_50_i
667                         ADD     $8 $3
668                         ADDI    $8 5
669                         EXCH    $9 $8
670                         ADDI    $8 -5
671                         SUB     $8 $3
672 test_45:                 BEQ     $7 $0
        test_false_47
673                         XORI    $7 1
674                         ADD     $8 $3
675                         ADDI    $8 5
676                         EXCH    $9 $8
677                         ADDI    $8 -5
678                         SUB     $8 $3
679                         XOR     $10 $9
680 loadMetAdd_53:          EXCH    $11 $10
681                         ADDI    $11 5
682                         EXCH    $12 $11
683                         XOR     $13 $12
684                         EXCH    $12 $11
685                         ADDI    $11 -5
686                         EXCH    $11 $10
687                         ADD     $8 $3
688                         ADDI    $8 5
689                         EXCH    $9 $8
690                         ADDI    $8 -5
691                         SUB     $8 $3
692                         EXCH    $3 $1
693                         ADDI    $1 -1
694                         EXCH    $6 $1
695                         ADDI    $1 -1
696                         EXCH    $10 $1
697                         ADDI    $1 -1
698                         ADDI    $13 -697
699 l_jmp_54:               SWAPBR  $13
700                         NEG     $13
701                         ADDI    $13 697
702                         ADDI    $1 1
703                         EXCH    $10 $1
704                         ADDI    $1 1
705                         EXCH    $6 $1
706                         ADDI    $1 1
707                         EXCH    $3 $1
708                         ADD     $8 $3
709                         ADDI    $8 5
710                         EXCH    $9 $8
711                         ADDI    $8 -5
712                         SUB     $8 $3
713                         EXCH    $11 $10
714                         ADDI    $11 5
715                         EXCH    $12 $11
716                         XOR     $13 $12
717                         EXCH    $12 $11
718                         ADDI    $11 -5
719 loadMetAdd_53_i:        EXCH    $11 $10
```

```
                        XOR     $10 $9
                        ADD     $8 $3
                        ADDI    $8 5
                        EXCH    $9 $8
                        ADDI    $8 -5
                        SUB     $8 $3
                        XORI    $7 1
assert_true_46:         BRA     assert_48
test_false_47:          BRA     test_45
assert_48:              BNE     $7 $0
    assert_true_46
                        ADD     $8 $3
                        ADDI    $8 5
                        EXCH    $9 $8
                        ADDI    $8 -5
                        SUB     $8 $3
cmp_top_55:             BEQ     $9 $0
    cmp_bot_56
                        XORI    $10 1
cmp_bot_56:             BEQ     $9 $0
    cmp_top_55
f_top_57:               BEQ     $10 $0
    f_bot_58
                        XORI    $11 1
f_bot_58:               BEQ     $10 $0
    f_top_57
                        XOR     $7 $11
f_bot_58_i:             BEQ     $10 $0
    f_top_57_i
                        XORI    $11 1
f_top_57_i:             BEQ     $10 $0
    f_bot_58_i
cmp_bot_56_i:           BEQ     $9 $0
    cmp_top_55_i
                        XORI    $10 1
cmp_top_55_i:           BEQ     $9 $0
    cmp_bot_56_i
                        ADD     $8 $3
                        ADDI    $8 5
                        EXCH    $9 $8
                        ADDI    $8 -5
                        SUB     $8 $3
l_append_7_bot:         BRA     l_append_7_top
l_appendCell_1_top:     BRA
    l_appendCell_1_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $3 $1
                        ADDI    $1 -1
l_appendCell_1:         SWAPBR  $2
                        NEG     $2
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
cmp_top_63:             BNE     $9 $0
    cmp_bot_64
                        XORI    $10 1
cmp_bot_64:             BNE     $9 $0
    cmp_top_63
                        EXCH    $11 $6
cmp_top_65:             BEQ     $11 $0
    cmp_bot_66
                        XORI    $12 1
cmp_bot_66:             BEQ     $11 $0
    cmp_top_65
                        ANDX    $13 $10 $12
f_top_67:               BEQ     $13 $0
    f_bot_68
                        XORI    $14 1
f_bot_68:               BEQ     $13 $0
    f_top_67
                        XOR     $7 $14
f_bot_68_i:             BEQ     $13 $0
    f_top_67_i
                        XORI    $14 1
f_top_67_i:             BEQ     $13 $0
    f_bot_68_i
                        ANDX    $13 $10 $12
cmp_bot_66_i:           BEQ     $11 $0
    cmp_top_65_i
                        XORI    $12 1
cmp_top_65_i:           BEQ     $11 $0
    cmp_bot_66_i
                        EXCH    $11 $6
cmp_bot_64_i:           BNE     $9 $0
    cmp_top_63_i
                        XORI    $10 1
cmp_top_63_i:           BNE     $9 $0
    cmp_bot_64_i
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
test_59:                BEQ     $7 $0
    test_false_61
                        XORI    $7 1
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
                        EXCH    $10 $6
swap_69:                XOR     $9 $10
                        XOR     $10 $9
                        XOR     $9 $10
                        EXCH    $10 $6
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
                        XORI    $7 1
assert_true_60:         BRA     assert_62
test_false_61:          BRA     test_59
assert_62:              BNE     $7 $0
    assert_true_60
                        ADD     $8 $3
                        ADDI    $8 2
                        EXCH    $9 $8
                        ADDI    $8 -2
                        SUB     $8 $3
```

```
828 cmp_top_70:             BEQ    $9 $0
        cmp_bot_71
829                         XORI   $10 1
830 cmp_bot_71:             BEQ    $9 $0
        cmp_top_70
831                         EXCH   $11 $6
832 cmp_top_72:             BNE    $11 $0
        cmp_bot_73
833                         XORI   $12 1
834 cmp_bot_73:             BNE    $11 $0
        cmp_top_72
835                         ANDX   $13 $10 $12
836 f_top_74:               BEQ    $13 $0
        f_bot_75
837                         XORI   $14 1
838 f_bot_75:               BEQ    $13 $0
        f_top_74
839                         XOR    $7 $14
840 f_bot_75_i:             BEQ    $13 $0
        f_top_74_i
841                         XORI   $14 1
842 f_top_74_i:             BEQ    $13 $0
        f_bot_75_i
843                         ANDX   $13 $10 $12
844 cmp_bot_73_i:           BNE    $11 $0
        cmp_top_72_i
845                         XORI   $12 1
846 cmp_top_72_i:           BNE    $11 $0
        cmp_bot_73_i
847                         EXCH   $11 $6
848 cmp_bot_71_i:           BEQ    $9 $0
        cmp_top_70_i
849                         XORI   $10 1
850 cmp_top_70_i:           BEQ    $9 $0
        cmp_bot_71_i
851                         ADD    $8 $3
852                         ADDI   $8 2
853                         EXCH   $9 $8
854                         ADDI   $8 -2
855                         SUB    $8 $3
856                         ADD    $8 $3
857                         ADDI   $8 2
858                         EXCH   $9 $8
859                         ADDI   $8 -2
860                         SUB    $8 $3
861 cmp_top_80:             BEQ    $9 $0
        cmp_bot_81
862                         XORI   $10 1
863 cmp_bot_81:             BEQ    $9 $0
        cmp_top_80
864 f_top_82:               BEQ    $10 $0
        f_bot_83
865                         XORI   $11 1
866 f_bot_83:               BEQ    $10 $0
        f_top_82
867                         XOR    $7 $11
868 f_bot_83_i:             BEQ    $10 $0
        f_top_82_i
869                         XORI   $11 1
870 f_top_82_i:             BEQ    $10 $0
        f_bot_83_i
871 cmp_bot_81_i:           BEQ    $9 $0
        cmp_top_80_i
872                         XORI   $10 1
873 cmp_top_80_i:           BEQ    $9 $0
        cmp_bot_81_i
874                         ADD    $8 $3
875                         ADDI   $8 2
876                         EXCH   $9 $8
877                         ADDI   $8 -2
878                         SUB    $8 $3
879 test_76:                 BEQ    $7 $0
        test_false_78
880                         XORI   $7 1
881                         ADD    $8 $3
882                         ADDI   $8 2
883                         EXCH   $9 $8
884                         ADDI   $8 -2
885                         SUB    $8 $3
886                         XOR    $10 $9
887 loadMetAdd_84:          EXCH   $11 $10
888                         ADDI   $11 5
889                         EXCH   $12 $11
890                         XOR    $13 $12
891                         EXCH   $12 $11
892                         ADDI   $11 -5
893                         EXCH   $11 $10
894                         ADD    $8 $3
895                         ADDI   $8 2
896                         EXCH   $9 $8
897                         ADDI   $8 -2
898                         SUB    $8 $3
899                         EXCH   $3 $1
900                         ADDI   $1 -1
901                         EXCH   $6 $1
902                         ADDI   $1 -1
903                         EXCH   $10 $1
904                         ADDI   $1 -1
905                         ADDI   $13 -904
906 l_jmp_85:               SWAPBR $13
907                         NEG    $13
908                         ADDI   $13 904
909                         ADDI   $1 1
910                         EXCH   $10 $1
911                         ADDI   $1 1
912                         EXCH   $6 $1
913                         ADDI   $1 1
914                         EXCH   $3 $1
915                         ADD    $8 $3
916                         ADDI   $8 2
917                         EXCH   $9 $8
918                         ADDI   $8 -2
919                         SUB    $8 $3
920                         EXCH   $11 $10
921                         ADDI   $11 5
922                         EXCH   $12 $11
923                         XOR    $13 $12
924                         EXCH   $12 $11
925                         ADDI   $11 -5
926 loadMetAdd_84_i:        EXCH   $11 $10
927                         XOR    $10 $9
928                         ADD    $8 $3
929                         ADDI   $8 2
930                         EXCH   $9 $8
931                         ADDI   $8 -2
932                         SUB    $8 $3
933                         XORI   $7 1
934 assert_true_77:         BRA    assert_79
935 test_false_78:          BRA    test_76
936 assert_79:               BNE    $7 $0
        assert_true_77
937                         ADD    $8 $3
```

```
938                             ADDI    $8 2
939                             EXCH    $9 $8
940                             ADDI    $8 -2
941                             SUB     $8 $3
942 cmp_top_86:                 BEQ     $9 $0
        cmp_bot_87
943                             XORI    $10 1
944 cmp_bot_87:                 BEQ     $9 $0
        cmp_top_86
945 f_top_88:                   BEQ     $10 $0
        f_bot_89
946                             XORI    $11 1
947 f_bot_89:                   BEQ     $10 $0
        f_top_88
948                             XOR     $7 $11
949 f_bot_89_i:                 BEQ     $10 $0
        f_top_88_i
950                             XORI    $11 1
951 f_top_88_i:                 BEQ     $10 $0
        f_bot_89_i
952 cmp_bot_87_i:               BEQ     $9 $0
        cmp_top_86_i
953                             XORI    $10 1
954 cmp_top_86_i:               BEQ     $9 $0
        cmp_bot_87_i
955                             ADD     $8 $3
956                             ADDI    $8 2
957                             EXCH    $9 $8
958                             ADDI    $8 -2
959                             SUB     $8 $3
960                             ADD     $7 $3
961                             ADDI    $7 3
962                             EXCH    $8 $7
963                             ADDI    $7 -3
964                             SUB     $7 $3
965                             XORI    $9 1
966                             ADD     $8 $9
967                             XORI    $9 1
968                             ADD     $7 $3
969                             ADDI    $7 3
970                             EXCH    $8 $7
971                             ADDI    $7 -3
972                             SUB     $7 $3
973 l_appendCell_1_bot:         BRA
        l_appendCell_1_top
974 l_main_0_top:               BRA     l_main_0_bot
975                             ADDI    $1 1
976                             EXCH    $2 $1
977                             EXCH    $3 $1
978                             ADDI    $1 -1
979 l_main_0:                   SWAPBR  $2
980                             NEG     $2
981                             ADDI    $1 1
982                             EXCH    $3 $1
983                             EXCH    $2 $1
984                             ADDI    $1 -1
985                             EXCH    $3 $1
986                             ADDI    $1 -1
987 obj_con_90:                 ADDI    $8 4
988                             EXCH    $8 $1
989                             ADDI    $1 -1
990                             EXCH    $7 $1
991                             ADDI    $1 -1
992                             BRA     l_malloc
993                             ADDI    $1 1
994                             EXCH    $7 $1
995                             ADDI    $1 1
996                             EXCH    $8 $1
997 obj_con_90_i:               ADDI    $8 -4
998                             ADDI    $1 1
999                             EXCH    $3 $1
1000                            ADD     $6 $3
1001                            ADDI    $6 2
1002                            XORI    $8 4
1003                            EXCH    $8 $7
1004                            ADDI    $7 1
1005                            XORI    $8 1
1006                            EXCH    $8 $7
1007 obj_con_90_bot:            ADDI    $7 -1
1008                            EXCH    $7 $6
1009                            ADDI    $6 -2
1010                            SUB     $6 $3
1011                            ADD     $6 $3
1012                            ADDI    $6 3
1013                            EXCH    $7 $6
1014                            ADDI    $6 -3
1015                            SUB     $6 $3
1016                            XORI    $8 10
1017                            ADD     $7 $8
1018                            XORI    $8 10
1019                            ADD     $6 $3
1020                            ADDI    $6 3
1021                            EXCH    $7 $6
1022                            ADDI    $6 -3
1023                            SUB     $6 $3
1024 localBlock_113:            XOR     $6 $1
1025                            XOR     $7 $0
1026                            EXCH    $7 $1
1027                            ADDI    $1 -1
1028                            XORI    $7 1
1029 entry_91:                  BEQ     $7 $0
        assert_93
1030                            EXCH    $8 $6
1031 cmp_top_95:                BNE     $8 $0
        cmp_bot_96
1032                            XORI    $9 1
1033 cmp_bot_96:                BNE     $8 $0
        cmp_top_95
1034 f_top_97:                  BEQ     $9 $0 f_bot_98
1035                            XORI    $10 1
1036 f_bot_98:                  BEQ     $9 $0 f_top_97
1037                            XOR     $7 $10
1038 f_bot_98_i:                BEQ     $9 $0
        f_top_97_i
1039                            XORI    $10 1
1040 f_top_97_i:                BEQ     $9 $0
        f_bot_98_i
1041 cmp_bot_96_i:              BNE     $8 $0
        cmp_top_95_i
1042                            XORI    $9 1
1043 cmp_top_95_i:              BNE     $8 $0
        cmp_bot_96_i
1044                            EXCH    $8 $6
1045                            EXCH    $8 $6
1046                            ADD     $9 $3
1047                            ADDI    $9 3
1048                            EXCH    $10 $9
1049                            ADDI    $9 -3
1050                            SUB     $9 $3
1051 cmp_top_109:               BNE     $8 $10
        cmp_bot_110
1052                            XORI    $11 1
```

```
1053 cmp_bot_110:            BNE    $8 $10
         cmp_top_109
1054 f_top_111:              BEQ    $11 $0
         f_bot_112
1055                         XORI   $12 1
1056 f_bot_112:              BEQ    $11 $0
         f_top_111
1057                         XOR    $7 $12
1058 f_bot_112_i:            BEQ    $11 $0
         f_top_111_i
1059                         XORI   $12 1
1060 f_top_111_i:            BEQ    $11 $0
         f_bot_112_i
1061 cmp_bot_110_i:          BNE    $8 $10
         cmp_top_109_i
1062                         XORI   $11 1
1063 cmp_top_109_i:          BNE    $8 $10
         cmp_bot_110_i
1064                         ADD    $9 $3
1065                         ADDI   $9 3
1066                         EXCH   $10 $9
1067                         ADDI   $9 -3
1068                         SUB    $9 $3
1069                         EXCH   $8 $6
1070 test_92:                BNE    $7 $0 exit_94
1071 localBlock_108:         XOR    $8 $1
1072                         XOR    $9 $0
1073                         EXCH   $9 $1
1074                         ADDI   $1 -1
1075                         EXCH   $3 $1
1076                         ADDI   $1 -1
1077                         EXCH   $8 $1
1078                         ADDI   $1 -1
1079                         EXCH   $6 $1
1080                         ADDI   $1 -1
1081 obj_con_99:             ADDI   $10 8
1082                         EXCH   $10 $1
1083                         ADDI   $1 -1
1084                         EXCH   $9 $1
1085                         ADDI   $1 -1
1086                         BRA    l_malloc
1087                         ADDI   $1 1
1088                         EXCH   $9 $1
1089                         ADDI   $1 1
1090                         EXCH   $10 $1
1091 obj_con_99_i:           ADDI   $10 -8
1092                         ADDI   $1 1
1093                         EXCH   $6 $1
1094                         ADDI   $1 1
1095                         EXCH   $8 $1
1096                         ADDI   $1 1
1097                         EXCH   $3 $1
1098                         XORI   $10 5
1099                         EXCH   $10 $9
1100                         ADDI   $9 1
1101                         XORI   $10 1
1102                         EXCH   $10 $9
1103 obj_con_99_bot:         ADDI   $9 -1
1104                         EXCH   $9 $8
1105 localBlock_103:         XOR    $9 $1
1106                         XOR    $10 $0
1107                         EXCH   $10 $1
1108                         ADDI   $1 -1
1109                         EXCH   $10 $9
1110                         EXCH   $11 $8
1111 copy_100:               XOR    $10 $11
1112                         ADDI   $11 1
1113                         EXCH   $12 $11
1114                         ADDI   $12 1
1115                         EXCH   $12 $11
1116                         ADDI   $11 -1
1117                         EXCH   $11 $8
1118                         EXCH   $10 $9
1119                         EXCH   $10 $8
1120                         XOR    $11 $10
1121 loadMetAdd_101:         EXCH   $12 $11
1122                         ADDI   $12 4
1123                         EXCH   $13 $12
1124                         XOR    $14 $13
1125                         EXCH   $13 $12
1126                         ADDI   $12 -4
1127                         EXCH   $12 $11
1128                         EXCH   $10 $8
1129                         EXCH   $3 $1
1130                         ADDI   $1 -1
1131                         EXCH   $8 $1
1132                         ADDI   $1 -1
1133                         EXCH   $6 $1
1134                         ADDI   $1 -1
1135                         EXCH   $9 $1
1136                         ADDI   $1 -1
1137                         EXCH   $11 $1
1138                         ADDI   $1 -1
1139                         ADDI   $14 -1138
1140 l_jmp_102:              SWAPBR $14
1141                         NEG    $14
1142                         ADDI   $14 1138
1143                         ADDI   $1 1
1144                         EXCH   $11 $1
1145                         ADDI   $1 1
1146                         EXCH   $9 $1
1147                         ADDI   $1 1
1148                         EXCH   $6 $1
1149                         ADDI   $1 1
1150                         EXCH   $8 $1
1151                         ADDI   $1 1
1152                         EXCH   $3 $1
1153                         EXCH   $10 $8
1154                         EXCH   $12 $11
1155                         ADDI   $12 4
1156                         EXCH   $13 $12
1157                         XOR    $14 $13
1158                         EXCH   $13 $12
1159                         ADDI   $12 -4
1160 loadMetAdd_101_i:       EXCH   $12 $11
1161                         XOR    $11 $10
1162                         EXCH   $10 $8
1163                         ADDI   $1 1
1164                         EXCH   $10 $1
1165                         XOR    $10 $0
1166 localBlock_103_i:       XOR    $9 $1
1167                         EXCH   $9 $8
1168                         XOR    $10 $9
1169 loadMetAdd_104:         EXCH   $11 $10
1170                         ADDI   $11 0
1171                         EXCH   $12 $11
1172                         XOR    $13 $12
1173                         EXCH   $12 $11
1174                         ADDI   $11 0
1175                         EXCH   $11 $10
1176                         EXCH   $9 $8
1177                         EXCH   $3 $1
```

```
                        ADDI   $1 -1
                        EXCH   $8 $1
                        ADDI   $1 -1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $10 $1
                        ADDI   $1 -1
                        ADDI   $13 -1184
l_jmp_105:              SWAPBR $13
                        NEG    $13
                        ADDI   $13 1184
                        ADDI   $1 1
                        EXCH   $10 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        ADDI   $1 1
                        EXCH   $8 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        EXCH   $9 $8
                        EXCH   $11 $10
                        ADDI   $11 0
                        EXCH   $12 $11
                        XOR    $13 $12
                        EXCH   $12 $11
                        ADDI   $11 0
loadMetAdd_104_i:       EXCH   $11 $10
                        XOR    $10 $9
                        EXCH   $9 $8
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
                        XOR    $11 $10
loadMetAdd_106:         EXCH   $12 $11
                        ADDI   $12 0
                        EXCH   $13 $12
                        XOR    $14 $13
                        EXCH   $13 $12
                        ADDI   $12 0
                        EXCH   $12 $11
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
                        EXCH   $3 $1
                        ADDI   $1 -1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $8 $1
                        ADDI   $1 -1
                        EXCH   $11 $1
                        ADDI   $1 -1
                        ADDI   $14 -1232
l_jmp_107:              SWAPBR $14
                        NEG    $14
                        ADDI   $14 1232
                        ADDI   $1 1
                        EXCH   $11 $1
                        ADDI   $1 1
                        EXCH   $8 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
                        EXCH   $12 $11
                        ADDI   $12 0
                        EXCH   $13 $12
                        XOR    $14 $13
                        EXCH   $13 $12
                        ADDI   $12 0
                        EXCH   $12 $11
loadMetAdd_106_i:       XOR    $11 $10
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
                        ADDI   $1 1
                        EXCH   $9 $1
                        XOR    $9 $0
localBlock_108_i:       XOR    $8 $1
                        EXCH   $8 $6
                        XORI   $9 1
                        ADD    $8 $9
                        XORI   $9 1
                        EXCH   $8 $6
assert_93:              BRA    entry_91
exit_94:                BRA    test_92
                        XORI   $7 1
                        ADD    $7 $3
                        ADDI   $7 3
                        EXCH   $8 $7
                        ADDI   $7 -3
                        SUB    $7 $3
                        ADDI   $1 1
                        EXCH   $9 $1
                        XOR    $9 $8
localBlock_113_i:       XOR    $6 $1
                        ADD    $7 $3
                        ADDI   $7 3
                        EXCH   $8 $7
                        ADDI   $7 -3
                        SUB    $7 $3
l_main_0_bot:           BRA    l_main_0_top
start:                  BRA    top
                        START
                        ADDI   $4 1338
                        XOR    $5 $4
                        ADDI   $5 10
                        XOR    $7 $5
                        ADDI   $4 10
                        ADDI   $4 -1
                        EXCH   $7 $4
                        ADDI   $4 1
                        ADDI   $4 -10
                        XOR    $1 $5
                        ADDI   $1 2048
                        ADDI   $1 -4
                        XOR    $3 $1
                        XORI   $6 3
                        EXCH   $6 $3
                        ADDI   $1 -1
                        EXCH   $3 $1
                        ADDI   $1 -1
```

```
1310                    BRA     l_main_0
1311                    ADDI    $1 1
1312                    EXCH    $3 $1
1313                    ADDI    $3 1
1314                    ADDI    $3 1
1315                    EXCH    $6 $3
1316                    XORI    $7 1
1317                    EXCH    $6 $7
1318                    XORI    $7 1
1319                    ADDI    $3 -1
1320                    ADDI    $3 -1
1321                    ADDI    $3 1
1322                    ADDI    $3 2
1323                    EXCH    $6 $3
1324                    XORI    $7 2
1325                    EXCH    $6 $7
1326                    XORI    $7 2
1327                    ADDI    $3 -2
1328                    ADDI    $3 -1
1329                    ADDI    $1 1
1330                    EXCH    $6 $3
1331                    XORI    $6 3
1332                    XOR     $3 $1
1333                    ADDI    $1 4
1334                    ADDI    $1 -2048
1335                    XOR     $1 $5
1336                    ADDI    $5 -10
1337                    XOR     $5 $4
1338                    ADDI    $4 -1338
1339    finish:         FINISH
```

RTM.rplpp

```
class Cell
    Cell self
    Cell right
    Cell left
    int data

    method getLeft(Cell cell)
        right <=> cell

    method getRight(Cell cell)
        left <=> cell

    method getSelf(Cell cell)
        self <=> cell

    method getSymbol(int symbol)
        symbol <=> data

class RTM
    Cell tapeHead
    int[] q1
    int[] q2
    int[] s1
    int[] s2
    int SLASH
    int LEFT
    int RIGHT
    int BLANK
    int state
    int Qs
    int Qf
    int symbol
    int PC_MAX
    int pc

    method initLiterals()
        // Initialize string literals
        SLASH += 9999
        LEFT  += 9998
        RIGHT += 9997
        BLANK += 9996

        // Set max program counter
        PC_MAX += 7

    method initRules()
        // Initialize transition rule arrays
        new int[8] q1
        new int[8] q2
        new int[8] s1
        new int[8] s2

        // Define transition rules for binary number incrementation
        q1[0] += 1
        s1[0] += BLANK
        s2[0] += BLANK
        q2[0] += 2

        q1[1] += 2
        s1[1] += SLASH
        s2[1] += RIGHT
        q2[1] += 3

```

```
64          q1[2] += 3
65          s1[2] += 0
66          s2[2] += 1
67          q2[2] += 4
68
69          q1[3] += 3
70          s1[3] += 1
71          s2[3] += 0
72          q2[3] += 2
73
74          q1[4] += 3
75          s1[4] += BLANK
76          s2[4] += BLANK
77          q2[4] += 4
78
79          q1[5] += 4
80          s1[5] += SLASH
81          s2[5] += LEFT
82          q2[5] += 5
83
84          q1[6] += 5
85          s1[6] += 0
86          s2[6] += 0
87          q2[6] += 4
88
89          q1[7] += 5
90          s1[7] += BLANK
91          s2[7] += BLANK
92          q2[7] += 6
93
94      method initTape()
95          local Cell cell0 = nil
96          local Cell cell1 = nil
97          local Cell cell2 = nil
98          local Cell cell3 = nil
99          local Cell cell4 = nil
100
101         // Init cells
102         new Cell cell0
103         new Cell cell1
104         new Cell cell2
105         new Cell cell3
106         new Cell cell4
107
108         // Write 1 1 0 1 on tape
109         symbol += BLANK
110         uncall cell0::getSymbol(symbol)
111         symbol += 1
112         uncall cell1::getSymbol(symbol)
113         symbol += 1
114         uncall cell2::getSymbol(symbol)
115         symbol += 1
116         uncall cell4::getSymbol(symbol)
117
118         // Set tape head
119         tapeHead <=> cell0
120
121         // Set self pointers
122         copy Cell tapeHead cell0
123         uncall tapeHead::getSelf(cell0)
124         copy Cell cell1 cell0
125         uncall cell1::getSelf(cell0)
126         copy Cell cell2 cell0
127         uncall cell2::getSelf(cell0)
128         copy Cell cell3 cell0
129         uncall cell3::getSelf(cell0)
```

```
130         copy Cell cell4 cell0
131         uncall cell4::getSelf(cell0)
132
133         // Link cell 3 and 4
134         copy Cell cell3 cell0
135         uncall cell4::getLeft(cell0)
136         uncall cell3::getRight(cell4)
137
138         // Link cell 2 and 3
139         copy Cell cell2 cell0
140         uncall cell3::getLeft(cell0)
141         uncall cell2::getRight(cell3)
142
143         // Link cell1 and cell 2
144         copy Cell cell1 cell0
145         uncall cell2::getLeft(cell0)
146         uncall cell1::getRight(cell2)
147
148         // Link tapeHead and cell 1
149         copy Cell tapeHead cell0
150         uncall cell1::getLeft(cell0)
151         uncall tapeHead::getRight(cell1)
152
153         delocal Cell cell4 = nil
154         delocal Cell cell3 = nil
155         delocal Cell cell2 = nil
156         delocal Cell cell1 = nil
157         delocal Cell cell0 = nil
158
159     method init()
160         // Prepare for simulation
161         call initLiterals()
162         call initRules()
163         call initTape()
164
165         // Init pc, start and finishing state
166         state += 1
167         Qs += 1
168         Qf += 6
169
170         // Start simulation
171         call simulate()
172
173     method simulate()
174         from state = Qs do
175             call tapeHead::getSymbol(symbol)    // Fetch current symbol
176             call inst()
177             uncall tapeHead::getSymbol(symbol)  // Zero-clear symbol
178             pc += 1                             // Increment pc
179
180             if pc = PC_MAX then                 // Reset pc
181                 pc ^= PC_MAX
182             else skip
183             fi pc = 0
184         loop skip
185         until state = Qf
186
187     method inst()
188         if state = q1[pc] & symbol = s1[pc] then    // Symbol rule:
189             state += q2[pc]-q1[pc]                  // set state to q2[pc]
190             symbol += s2[pc]-s1[pc]                 // set symbol to s2[pc]
191         else skip
192         fi state = q2[pc] & symbol = s2[pc]
193         if state = q1[pc] & s1[pc] = SLASH then     // Move rule:
194             state += q2[pc]-q1[pc]                  // set state to q2[pc]
195             if s2[pc] = RIGHT then
```

```
196                 call moveRight()                    // Move tape head right
197             else skip
198             fi s2[pc] = RIGHT
199             if s2[pc] = LEFT then
200                 uncall moveRight()                  // Move tape head left
201             else skip
202             fi s2[pc] = LEFT
203         else skip
204         fi state = q2[pc] & s1[pc] = SLASH
205
206     method moveRight()
207         local Cell right = nil
208         local Cell tmp = nil
209         uncall tapeHead::getSymbol(symbol)   // Put symbol back in current cell
210         call tapeHead::getRight(right)       // Get right neighbour
211
212         if right = nil & symbol = BLANK then
213             symbol ^= BLANK                  // Zero clear symbol
214             new Cell right                   // Init new neighbour
215             copy Cell right tmp              // Copy reference to self
216             uncall right::getSelf(tmp)       // Store self reference
217             uncall right::getLeft(tapeHead)  // Set tape head as left of new cell
218             right <=> tapeHead
219         else
220             call right::getLeft(tmp)         // Get copy of tape head reference
221             uncopy Cell tmp tapeHead         // Clear reference to tape head
222
223             if tapeHead = nil & symbol = BLANK then
224                 call tmp::getSelf(tapeHead)  // rev: set self pointer
225                 uncopy Cell tmp tapeHead     // rev: new self pointer
226                 delete Cell tmp              // rev: new left neighbour
227                 symbol ^= BLANK
228             else skip                        // In reverse:
229             fi tmp = nil                     // Allocate new left if current is nil
230
231             uncall right::getLeft(tmp)       // Put tape head reference back
232             tapeHead <=> right
233             call tapeHead::getRight(right)   // Get right of new tape head
234             call tapeHead::getSymbol(symbol) // Get symbol of new tape head
235         fi right = nil
236
237         uncall tapeHead::getRight(right)     // Set right neighbour
238         delocal Cell right = nil
239         delocal Cell tmp = nil
240
241 class Program
242     RTM bni
243
244     method main()
245         // This program contains a RTM implementing
246         // incrementation of a non-negative n-bit binary number by 1 (modulo 2n).
247         // The tape is initialized with | b | 1Â | 1 | 0 | 0 | and after execution,
248         // the tape is left with | bÂ | 0 | 0Â | 1 | 1 |
249         new RTM bni
250         call bni::init()
```

RTM.pal

```
;; pendulum pal file
top:                BRA    start
l_r_bni:            DATA   0
l_Program_vt:       DATA   6592
l_RTM_vt:           DATA   348
                    DATA   425
                    DATA   2181
                    DATA   3606
                    DATA   3677
                    DATA   3976
                    DATA   5727
l_Cell_vt:          DATA   226
                    DATA   257
                    DATA   288
                    DATA   319
l_malloc_top:       BRA    l_malloc_bot
l_malloc:           SWAPBR $2
                    NEG    $2
                    ADDI   $9 2
                    XOR    $8 $0
                    ADDI   $1 1
                    EXCH   $6 $1
                    ADDI   $1 1
                    EXCH   $7 $1
                    EXCH   $2 $1
                    ADDI   $1 -1
                    BRA    l_malloc1
                    ADDI   $1 1
                    EXCH   $2 $1
                    EXCH   $7 $1
                    ADDI   $1 -1
                    EXCH   $6 $1
                    ADDI   $1 -1
                    XOR    $8 $0
                    ADDI   $9 -2
l_malloc_bot:       BRA    l_malloc_top
l_malloc1_top:      BRA    l_malloc1_bot
                    ADDI   $1 1
                    EXCH   $2 $1
                    SUB    $17 $8
                    XOR    $17 $4
l_malloc1:          SWAPBR $2
                    NEG    $2
                    EXCH   $2 $1
                    ADDI   $1 -1
                    XOR    $17 $4
                    ADD    $17 $8
                    EXCH   $19 $17
                    XOR    $18 $19
                    EXCH   $19 $17
                    XOR    $13 $9
                    SUB    $13 $7
cmp_top_12:         BGEZ   $13 cmp_bot_13
                    XORI   $14 1
cmp_bot_13:         BGEZ   $13 cmp_top_12
                    XOR    $10 $14
cmp_bot_13_i:       BGEZ   $13
    cmp_top_12_i
                    XORI   $14 1
cmp_top_12_i:       BGEZ   $13
    cmp_bot_13_i
                    ADD    $13 $7
                    XOR    $13 $9
l_o_test:           BEQ    $10 $0
    l_o_test_false
                    XORI   $10 1
                    ADDI   $8 1
                    EXCH   $19 $17
                    XOR    $18 $19
                    EXCH   $19 $17
                    RL     $9 1
                    EXCH   $10 $1
                    ADDI   $1 -1
                    EXCH   $11 $1
                    ADDI   $1 -1
                    EXCH   $12 $1
                    ADDI   $1 -1
                    EXCH   $14 $1
                    ADDI   $1 -1
                    EXCH   $16 $1
                    ADDI   $1 -1
                    EXCH   $17 $1
                    ADDI   $1 -1
                    EXCH   $18 $1
                    ADDI   $1 -1
                    EXCH   $20 $1
                    ADDI   $1 -1
                    EXCH   $21 $1
                    ADDI   $1 -1
                    EXCH   $22 $1
                    ADDI   $1 -1
                    EXCH   $23 $1
                    ADDI   $1 -1
                    BRA    l_malloc1
                    ADDI   $1 1
                    EXCH   $23 $1
                    ADDI   $1 1
                    EXCH   $22 $1
                    ADDI   $1 1
                    EXCH   $21 $1
                    ADDI   $1 1
                    EXCH   $20 $1
                    ADDI   $1 1
                    EXCH   $18 $1
                    ADDI   $1 1
                    EXCH   $17 $1
                    ADDI   $1 1
                    EXCH   $16 $1
                    ADDI   $1 1
                    EXCH   $14 $1
                    ADDI   $1 1
                    EXCH   $12 $1
                    ADDI   $1 1
                    EXCH   $11 $1
                    ADDI   $1 1
                    EXCH   $10 $1
                    RR     $9 1
                    ADDI   $8 -1
                    XORI   $10 1
l_o_assert_true:    BRA    l_o_assert
l_o_test_false:     BRA    l_o_test
cmp_top_16:         BEQ    $18 $0
    cmp_bot_17
                    XORI   $20 1
```

```
cmp_bot_17:             BEQ     $18 $0
    cmp_top_16
                        XOR     $11 $20
cmp_bot_17_i:           BEQ     $18 $0
    cmp_top_16_i
                        XORI    $20 1
cmp_top_16_i:           BEQ     $18 $0
    cmp_bot_17_i
l_i_test:               BEQ     $11 $0
    l_i_test_false
                        XORI    $11 1
                        ADD     $6 $18
                        SUB     $18 $6
                        EXCH    $12 $6
                        EXCH    $12 $17
                        XOR     $12 $6
                        XORI    $11 1
l_i_assert_true:        BRA     l_i_assert
l_i_test_false:         BRA     l_i_test
                        ADDI    $8 1
                        RL      $9 1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $11 $1
                        ADDI    $1 -1
                        EXCH    $12 $1
                        ADDI    $1 -1
                        EXCH    $14 $1
                        ADDI    $1 -1
                        EXCH    $16 $1
                        ADDI    $1 -1
                        EXCH    $17 $1
                        ADDI    $1 -1
                        EXCH    $18 $1
                        ADDI    $1 -1
                        EXCH    $20 $1
                        ADDI    $1 -1
                        EXCH    $21 $1
                        ADDI    $1 -1
                        EXCH    $22 $1
                        ADDI    $1 -1
                        EXCH    $23 $1
                        ADDI    $1 -1
                        BRA     l_malloc1
                        ADDI    $1 1
                        EXCH    $23 $1
                        ADDI    $1 1
                        EXCH    $22 $1
                        ADDI    $1 1
                        EXCH    $21 $1
                        ADDI    $1 1
                        EXCH    $20 $1
                        ADDI    $1 1
                        EXCH    $18 $1
                        ADDI    $1 1
                        EXCH    $17 $1
                        ADDI    $1 1
                        EXCH    $16 $1
                        ADDI    $1 1
                        EXCH    $14 $1
                        ADDI    $1 1
                        EXCH    $12 $1
                        ADDI    $1 1
                        EXCH    $11 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        RR      $9 1
                        ADDI    $8 -1
                        XOR     $12 $6
                        EXCH    $12 $17
                        ADD     $6 $9
l_i_assert:             BNE     $11 $0
    l_i_assert_true
                        EXCH    $12 $17
                        SUB     $6 $9
cmp_top_18:             BEQ     $6 $12
    cmp_bot_19
                        XORI    $21 1
cmp_bot_19:             BEQ     $6 $12
    cmp_top_18
cmp_top_20:             BNE     $12 $0
    cmp_bot_21
                        XORI    $22 1
cmp_bot_21:             BNE     $12 $0
    cmp_top_20
                        ORX     $23 $21 $22
                        XOR     $11 $23
                        ORX     $23 $21 $22
cmp_bot_21_i:           BNE     $12 $0
    cmp_top_20_i
                        XORI    $22 1
cmp_top_20_i:           BNE     $12 $0
    cmp_bot_21_i
cmp_bot_19_i:           BEQ     $6 $12
    cmp_top_18_i
                        XORI    $21 1
cmp_top_18_i:           BEQ     $6 $12
    cmp_bot_19_i
                        ADD     $6 $9
                        EXCH    $12 $17
l_o_assert:             BNE     $10 $0
    l_o_assert_true
                        XOR     $15 $9
                        SUB     $15 $7
cmp_top_14:             BGEZ    $15 cmp_bot_15
                        XORI    $16 1
cmp_bot_15:             BGEZ    $15 cmp_top_14
                        XOR     $10 $16
cmp_bot_15_i:           BGEZ    $15
    cmp_top_14_i
                        XORI    $16 1
cmp_top_14_i:           BGEZ    $15
    cmp_bot_15_i
                        ADD     $15 $7
                        XOR     $15 $9
l_malloc1_bot:          BRA     l_malloc1_top
l_getLeft_8_top:        BRA
    l_getLeft_8_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $3 $1
                        ADDI    $1 -1
l_getLeft_8:            SWAPBR  $2
                        NEG     $2
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
```

```
236                             ADD     $7 $3
237                             ADDI    $7 3
238                             EXCH    $8 $7
239                             ADDI    $7 -3
240                             SUB     $7 $3
241                             EXCH    $9 $6
242 swap_22:                    XOR     $8 $9
243                             XOR     $9 $8
244                             XOR     $8 $9
245                             EXCH    $9 $6
246                             ADD     $7 $3
247                             ADDI    $7 3
248                             EXCH    $8 $7
249                             ADDI    $7 -3
250                             SUB     $7 $3
251 l_getLeft_8_bot:            BRA
        l_getLeft_8_top
252 l_getRight_9_top:           BRA
        l_getRight_9_bot
253                             ADDI    $1 1
254                             EXCH    $2 $1
255                             EXCH    $6 $1
256                             ADDI    $1 -1
257                             EXCH    $3 $1
258                             ADDI    $1 -1
259 l_getRight_9:               SWAPBR  $2
260                             NEG     $2
261                             ADDI    $1 1
262                             EXCH    $3 $1
263                             ADDI    $1 1
264                             EXCH    $6 $1
265                             EXCH    $2 $1
266                             ADDI    $1 -1
267                             ADD     $7 $3
268                             ADDI    $7 4
269                             EXCH    $8 $7
270                             ADDI    $7 -4
271                             SUB     $7 $3
272                             EXCH    $9 $6
273 swap_23:                    XOR     $8 $9
274                             XOR     $9 $8
275                             XOR     $8 $9
276                             EXCH    $9 $6
277                             ADD     $7 $3
278                             ADDI    $7 4
279                             EXCH    $8 $7
280                             ADDI    $7 -4
281                             SUB     $7 $3
282 l_getRight_9_bot:           BRA
        l_getRight_9_top
283 l_getSelf_10_top:           BRA
        l_getSelf_10_bot
284                             ADDI    $1 1
285                             EXCH    $2 $1
286                             EXCH    $6 $1
287                             ADDI    $1 -1
288                             EXCH    $3 $1
289                             ADDI    $1 -1
290 l_getSelf_10:               SWAPBR  $2
291                             NEG     $2
292                             ADDI    $1 1
293                             EXCH    $3 $1
294                             ADDI    $1 1
295                             EXCH    $6 $1
296                             EXCH    $2 $1
297                             ADDI    $1 -1
298                             ADD     $7 $3
299                             ADDI    $7 2
300                             EXCH    $8 $7
301                             ADDI    $7 -2
302                             SUB     $7 $3
303                             EXCH    $9 $6
304 swap_24:                    XOR     $8 $9
305                             XOR     $9 $8
306                             XOR     $8 $9
307                             EXCH    $9 $6
308                             ADD     $7 $3
309                             ADDI    $7 2
310                             EXCH    $8 $7
311                             ADDI    $7 -2
312                             SUB     $7 $3
313 l_getSelf_10_bot:           BRA
        l_getSelf_10_top
314 l_getSymbol_11_top:         BRA
        l_getSymbol_11_bot
315                             ADDI    $1 1
316                             EXCH    $2 $1
317                             EXCH    $6 $1
318                             ADDI    $1 -1
319                             EXCH    $3 $1
320                             ADDI    $1 -1
321 l_getSymbol_11:             SWAPBR  $2
322                             NEG     $2
323                             ADDI    $1 1
324                             EXCH    $3 $1
325                             ADDI    $1 1
326                             EXCH    $6 $1
327                             EXCH    $2 $1
328                             ADDI    $1 -1
329                             EXCH    $7 $6
330                             ADD     $8 $3
331                             ADDI    $8 5
332                             EXCH    $9 $8
333                             ADDI    $8 -5
334                             SUB     $8 $3
335 swap_25:                    XOR     $7 $9
336                             XOR     $9 $7
337                             XOR     $7 $9
338                             ADD     $8 $3
339                             ADDI    $8 5
340                             EXCH    $9 $8
341                             ADDI    $8 -5
342                             SUB     $8 $3
343                             EXCH    $7 $6
344 l_getSymbol_11_bot:         BRA
        l_getSymbol_11_top
345 l_initLiterals_1_top:       BRA
        l_initLiterals_1_bot
346                             ADDI    $1 1
347                             EXCH    $2 $1
348                             EXCH    $3 $1
349                             ADDI    $1 -1
350 l_initLiterals_1:           SWAPBR  $2
351                             NEG     $2
352                             ADDI    $1 1
353                             EXCH    $3 $1
354                             EXCH    $2 $1
355                             ADDI    $1 -1
356                             ADD     $6 $3
357                             ADDI    $6 7
358                             EXCH    $7 $6
359                             ADDI    $6 -7
```

```
                        SUB   $6 $3
                        XORI  $8 9999
                        ADD   $7 $8
                        XORI  $8 9999
                        ADD   $6 $3
                        ADDI  $6 7
                        EXCH  $7 $6
                        ADDI  $6 -7
                        SUB   $6 $3
                        ADD   $6 $3
                        ADDI  $6 8
                        EXCH  $7 $6
                        ADDI  $6 -8
                        SUB   $6 $3
                        XORI  $8 9998
                        ADD   $7 $8
                        XORI  $8 9998
                        ADD   $6 $3
                        ADDI  $6 8
                        EXCH  $7 $6
                        ADDI  $6 -8
                        SUB   $6 $3
                        ADD   $6 $3
                        ADDI  $6 9
                        EXCH  $7 $6
                        ADDI  $6 -9
                        SUB   $6 $3
                        XORI  $8 9997
                        ADD   $7 $8
                        XORI  $8 9997
                        ADD   $6 $3
                        ADDI  $6 9
                        EXCH  $7 $6
                        ADDI  $6 -9
                        SUB   $6 $3
                        ADD   $6 $3
                        ADDI  $6 10
                        EXCH  $7 $6
                        ADDI  $6 -10
                        SUB   $6 $3
                        XORI  $8 9996
                        ADD   $7 $8
                        XORI  $8 9996
                        ADD   $6 $3
                        ADDI  $6 10
                        EXCH  $7 $6
                        ADDI  $6 -10
                        SUB   $6 $3
                        ADD   $6 $3
                        ADDI  $6 15
                        EXCH  $7 $6
                        ADDI  $6 -15
                        SUB   $6 $3
                        XORI  $8 7
                        ADD   $7 $8
                        XORI  $8 7
                        ADD   $6 $3
                        ADDI  $6 15
                        EXCH  $7 $6
                        ADDI  $6 -15
                        SUB   $6 $3
l_initLiterals_1_bot:   BRA
    l_initLiterals_1_top
l_initRules_2_top:      BRA
    l_initRules_2_bot
                        ADDI  $1 1
                        EXCH  $2 $1
                        EXCH  $3 $1
                        ADDI  $1 -1
l_initRules_2:          SWAPBR $2
                        NEG   $2
                        ADDI  $1 1
                        EXCH  $3 $1
                        EXCH  $2 $1
                        ADDI  $1 -1
                        XORI  $7 8
arr_con_26:             ADDI  $9 2
                        ADD   $9 $7
                        XORI  $7 8
                        EXCH  $3 $1
                        ADDI  $1 -1
                        EXCH  $9 $1
                        ADDI  $1 -1
                        EXCH  $8 $1
                        ADDI  $1 -1
                        BRA   l_malloc
                        ADDI  $1 1
                        EXCH  $8 $1
                        ADDI  $1 1
                        EXCH  $9 $1
                        ADDI  $1 1
                        EXCH  $3 $1
                        XORI  $7 8
                        SUB   $9 $7
arr_con_26_i:           ADDI  $9 -2
                        XORI  $7 8
                        ADD   $6 $3
                        ADDI  $6 3
                        XORI  $7 8
                        XOR   $9 $7
                        EXCH  $9 $8
                        ADDI  $8 1
                        XORI  $9 1
                        EXCH  $9 $8
                        ADDI  $8 -1
arr_con_26_bot:         EXCH  $8 $6
                        XORI  $7 8
                        ADDI  $6 -3
                        SUB   $6 $3
                        XORI  $7 8
arr_con_27:             ADDI  $9 2
                        ADD   $9 $7
                        XORI  $7 8
                        EXCH  $3 $1
                        ADDI  $1 -1
                        EXCH  $9 $1
                        ADDI  $1 -1
                        EXCH  $8 $1
                        ADDI  $1 -1
                        BRA   l_malloc
                        ADDI  $1 1
                        EXCH  $8 $1
                        ADDI  $1 1
                        EXCH  $9 $1
                        ADDI  $1 1
                        EXCH  $3 $1
                        XORI  $7 8
                        SUB   $9 $7
arr_con_27_i:           ADDI  $9 -2
                        XORI  $7 8
                        ADD   $6 $3
                        ADDI  $6 4
```

```
                        XORI    $7 8
                        XOR     $9 $7
                        EXCH    $9 $8
                        ADDI    $8 1
                        XORI    $9 1
                        EXCH    $9 $8
                        ADDI    $8 -1
arr_con_27_bot:         EXCH    $8 $6
                        XORI    $7 8
                        ADDI    $6 -4
                        SUB     $6 $3
                        XORI    $7 8
arr_con_28:             ADDI    $9 2
                        ADD     $9 $7
                        XORI    $7 8
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        BRA     l_malloc
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        XORI    $7 8
                        SUB     $9 $7
arr_con_28_i:           ADDI    $9 -2
                        XORI    $7 8
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 8
                        XOR     $9 $7
                        EXCH    $9 $8
                        ADDI    $8 1
                        XORI    $9 1
                        EXCH    $9 $8
                        ADDI    $8 -1
arr_con_28_bot:         EXCH    $8 $6
                        XORI    $7 8
                        ADDI    $6 -5
                        SUB     $6 $3
                        XORI    $7 8
arr_con_29:             ADDI    $9 2
                        ADD     $9 $7
                        XORI    $7 8
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        BRA     l_malloc
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        XORI    $7 8
                        SUB     $9 $7
arr_con_29_i:           ADDI    $9 -2
                        XORI    $7 8
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 8
                        XOR     $9 $7
                        EXCH    $9 $8
                        ADDI    $8 1
                        XORI    $9 1
                        EXCH    $9 $8
                        ADDI    $8 -1
arr_con_29_bot:         EXCH    $8 $6
                        XORI    $7 8
                        ADDI    $6 -6
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 3
                        EXCH    $8 $6
                        XOR     $7 $8
                        EXCH    $8 $6
                        ADDI    $7 2
                        ADD     $7 $0
                        ADDI    $6 -3
                        SUB     $6 $3
                        EXCH    $9 $7
                        ADD     $6 $3
                        ADDI    $6 3
                        SUB     $7 $0
                        ADDI    $7 -2
                        EXCH    $8 $6
                        XOR     $7 $8
                        EXCH    $8 $6
                        ADDI    $6 -3
                        SUB     $6 $3
                        XORI    $10 1
assArrElem_30:          ADD     $9 $10
                        XORI    $10 1
                        ADD     $6 $3
                        ADDI    $6 3
                        EXCH    $8 $6
                        XOR     $7 $8
                        EXCH    $8 $6
                        ADDI    $7 2
                        ADD     $7 $0
                        ADDI    $6 -3
                        SUB     $6 $3
                        EXCH    $9 $7
                        ADD     $6 $3
                        ADDI    $6 3
                        SUB     $7 $0
                        ADDI    $7 -2
                        EXCH    $8 $6
                        XOR     $7 $8
                        EXCH    $8 $6
                        ADDI    $6 -3
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 5
                        EXCH    $8 $6
                        XOR     $7 $8
                        EXCH    $8 $6
                        ADDI    $7 2
                        ADD     $7 $0
                        ADDI    $6 -5
                        SUB     $6 $3
                        EXCH    $9 $7
                        ADD     $6 $3
                        ADDI    $6 5
```

```
622                     SUB     $7 $0
623                     ADDI    $7 -2
624                     EXCH    $8 $6
625                     XOR     $7 $8
626                     EXCH    $8 $6
627                     ADDI    $6 -5
628                     SUB     $6 $3
629                     ADD     $10 $3
630                     ADDI    $10 10
631                     EXCH    $11 $10
632                     ADDI    $10 -10
633                     SUB     $10 $3
634 assArrElem_31:      ADD     $9 $11
635                     ADD     $10 $3
636                     ADDI    $10 10
637                     EXCH    $11 $10
638                     ADDI    $10 -10
639                     SUB     $10 $3
640                     ADD     $6 $3
641                     ADDI    $6 5
642                     EXCH    $8 $6
643                     XOR     $7 $8
644                     EXCH    $8 $6
645                     ADDI    $7 2
646                     ADD     $7 $0
647                     ADDI    $6 -5
648                     SUB     $6 $3
649                     EXCH    $9 $7
650                     ADD     $6 $3
651                     ADDI    $6 5
652                     SUB     $7 $0
653                     ADDI    $7 -2
654                     EXCH    $8 $6
655                     XOR     $7 $8
656                     EXCH    $8 $6
657                     ADDI    $6 -5
658                     SUB     $6 $3
659                     ADD     $6 $3
660                     ADDI    $6 6
661                     EXCH    $8 $6
662                     XOR     $7 $8
663                     EXCH    $8 $6
664                     ADDI    $7 2
665                     ADD     $7 $0
666                     ADDI    $6 -6
667                     SUB     $6 $3
668                     EXCH    $9 $7
669                     ADD     $6 $3
670                     ADDI    $6 6
671                     SUB     $7 $0
672                     ADDI    $7 -2
673                     EXCH    $8 $6
674                     XOR     $7 $8
675                     EXCH    $8 $6
676                     ADDI    $6 -6
677                     SUB     $6 $3
678                     ADD     $10 $3
679                     ADDI    $10 10
680                     EXCH    $11 $10
681                     ADDI    $10 -10
682                     SUB     $10 $3
683 assArrElem_32:      ADD     $9 $11
684                     ADD     $10 $3
685                     ADDI    $10 10
686                     EXCH    $11 $10
687                     ADDI    $10 -10
688                     SUB     $10 $3
689                     ADD     $6 $3
690                     ADDI    $6 6
691                     EXCH    $8 $6
692                     XOR     $7 $8
693                     EXCH    $8 $6
694                     ADDI    $7 2
695                     ADD     $7 $0
696                     ADDI    $6 -6
697                     SUB     $6 $3
698                     EXCH    $9 $7
699                     ADD     $6 $3
700                     ADDI    $6 6
701                     SUB     $7 $0
702                     ADDI    $7 -2
703                     EXCH    $8 $6
704                     XOR     $7 $8
705                     EXCH    $8 $6
706                     ADDI    $6 -6
707                     SUB     $6 $3
708                     ADD     $6 $3
709                     ADDI    $6 4
710                     EXCH    $8 $6
711                     XOR     $7 $8
712                     EXCH    $8 $6
713                     ADDI    $7 2
714                     ADD     $7 $0
715                     ADDI    $6 -4
716                     SUB     $6 $3
717                     EXCH    $9 $7
718                     ADD     $6 $3
719                     ADDI    $6 4
720                     SUB     $7 $0
721                     ADDI    $7 -2
722                     EXCH    $8 $6
723                     XOR     $7 $8
724                     EXCH    $8 $6
725                     ADDI    $6 -4
726                     SUB     $6 $3
727                     XORI    $10 2
728 assArrElem_33:      ADD     $9 $10
729                     XORI    $10 2
730                     ADD     $6 $3
731                     ADDI    $6 4
732                     EXCH    $8 $6
733                     XOR     $7 $8
734                     EXCH    $8 $6
735                     ADDI    $7 2
736                     ADD     $7 $0
737                     ADDI    $6 -4
738                     SUB     $6 $3
739                     EXCH    $9 $7
740                     ADD     $6 $3
741                     ADDI    $6 4
742                     SUB     $7 $0
743                     ADDI    $7 -2
744                     EXCH    $8 $6
745                     XOR     $7 $8
746                     EXCH    $8 $6
747                     ADDI    $6 -4
748                     SUB     $6 $3
749                     ADD     $6 $3
750                     ADDI    $6 3
751                     XORI    $7 1
752                     EXCH    $9 $6
753                     XOR     $8 $9
```

```
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 1
                        ADDI    $6 -3
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 3
                        XORI    $7 1
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 1
                        ADDI    $6 -3
                        SUB     $6 $3
                        XORI    $11 2
assArrElem_34:          ADD     $10 $11
                        XORI    $11 2
                        ADD     $6 $3
                        ADDI    $6 3
                        XORI    $7 1
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 1
                        ADDI    $6 -3
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 3
                        XORI    $7 1
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 1
                        ADDI    $6 -3
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 1
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 1
                        ADDI    $6 -5
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 1
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 1
                        ADDI    $6 -5
                        SUB     $6 $3
                        SUB     $6 $3
                        ADD     $11 $3
                        ADDI    $11 7
                        EXCH    $12 $11
                        ADDI    $11 -7
                        SUB     $11 $3
assArrElem_35:          ADD     $10 $12
                        ADD     $11 $3
                        ADDI    $11 7
                        EXCH    $12 $11
                        ADDI    $11 -7
                        SUB     $11 $3
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 1
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 1
                        ADDI    $6 -5
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 1
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 1
                        ADDI    $6 -5
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 1
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 1
                        ADDI    $6 -6
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 1
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 1
                        ADDI    $6 -6
                        SUB     $6 $3
                        ADD     $11 $3
                        ADDI    $11 9
                        EXCH    $12 $11
                        ADDI    $11 -9
                        SUB     $11 $3
assArrElem_36:          ADD     $10 $12
                        ADD     $11 $3
                        ADDI    $11 9
```

```
                        EXCH    $12 $11
                        ADDI    $11 -9
                        SUB     $11 $3
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 1
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 1
                        ADDI    $6 -6
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 1
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 1
                        ADDI    $6 -6
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 4
                        XORI    $7 1
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 1
                        ADDI    $6 -4
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 4
                        XORI    $7 1
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 1
                        ADDI    $6 -4
                        SUB     $6 $3
                        XORI    $11 3
assArrElem_37:          ADD     $10 $11
                        XORI    $11 3
                        ADD     $6 $3
                        ADDI    $6 4
                        XORI    $7 1
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 1
                        ADDI    $6 -4
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 4
                        XORI    $7 1
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 1
                        ADDI    $6 -4
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 2
                        ADDI    $6 -3
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 3
                        XORI    $7 2
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 2
                        ADDI    $6 -3
                        SUB     $6 $3
                        XORI    $11 3
assArrElem_38:          ADD     $10 $11
                        XORI    $11 3
                        ADD     $6 $3
                        ADDI    $6 3
                        XORI    $7 2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 2
                        ADDI    $6 -3
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 3
                        XORI    $7 2
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 2
                        ADDI    $6 -3
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
```

```
1018                        XORI    $7 2
1019                        ADDI    $6 -5
1020                        SUB     $6 $3
1021                        EXCH    $10 $8
1022                        ADD     $6 $3
1023                        ADDI    $6 5
1024                        XORI    $7 2
1025                        SUB     $8 $7
1026                        ADDI    $8 -2
1027                        EXCH    $9 $6
1028                        XOR     $8 $9
1029                        EXCH    $9 $6
1030                        XORI    $7 2
1031                        ADDI    $6 -5
1032                        SUB     $6 $3
1033 assArrElem_39:        ADD     $10 $0
1034                        ADD     $6 $3
1035                        ADDI    $6 5
1036                        XORI    $7 2
1037                        EXCH    $9 $6
1038                        XOR     $8 $9
1039                        EXCH    $9 $6
1040                        ADDI    $8 2
1041                        ADD     $8 $7
1042                        XORI    $7 2
1043                        ADDI    $6 -5
1044                        SUB     $6 $3
1045                        EXCH    $10 $8
1046                        ADD     $6 $3
1047                        ADDI    $6 5
1048                        XORI    $7 2
1049                        SUB     $8 $7
1050                        ADDI    $8 -2
1051                        EXCH    $9 $6
1052                        XOR     $8 $9
1053                        EXCH    $9 $6
1054                        XORI    $7 2
1055                        ADDI    $6 -5
1056                        SUB     $6 $3
1057                        ADD     $6 $3
1058                        ADDI    $6 6
1059                        XORI    $7 2
1060                        EXCH    $9 $6
1061                        XOR     $8 $9
1062                        EXCH    $9 $6
1063                        ADDI    $8 2
1064                        ADD     $8 $7
1065                        XORI    $7 2
1066                        ADDI    $6 -6
1067                        SUB     $6 $3
1068                        EXCH    $10 $8
1069                        ADD     $6 $3
1070                        ADDI    $6 6
1071                        XORI    $7 2
1072                        SUB     $8 $7
1073                        ADDI    $8 -2
1074                        EXCH    $9 $6
1075                        XOR     $8 $9
1076                        EXCH    $9 $6
1077                        XORI    $7 2
1078                        ADDI    $6 -6
1079                        SUB     $6 $3
1080                        XORI    $11 1
1081 assArrElem_40:        ADD     $10 $11
1082                        XORI    $11 1
1083                        ADD     $6 $3
1084                        ADDI    $6 6
1085                        XORI    $7 2
1086                        EXCH    $9 $6
1087                        XOR     $8 $9
1088                        EXCH    $9 $6
1089                        ADDI    $8 2
1090                        ADD     $8 $7
1091                        XORI    $7 2
1092                        ADDI    $6 -6
1093                        SUB     $6 $3
1094                        EXCH    $10 $8
1095                        ADD     $6 $3
1096                        ADDI    $6 6
1097                        XORI    $7 2
1098                        SUB     $8 $7
1099                        ADDI    $8 -2
1100                        EXCH    $9 $6
1101                        XOR     $8 $9
1102                        EXCH    $9 $6
1103                        XORI    $7 2
1104                        ADDI    $6 -6
1105                        SUB     $6 $3
1106                        ADD     $6 $3
1107                        ADDI    $6 4
1108                        XORI    $7 2
1109                        EXCH    $9 $6
1110                        XOR     $8 $9
1111                        EXCH    $9 $6
1112                        ADDI    $8 2
1113                        ADD     $8 $7
1114                        XORI    $7 2
1115                        ADDI    $6 -4
1116                        SUB     $6 $3
1117                        EXCH    $10 $8
1118                        ADD     $6 $3
1119                        ADDI    $6 4
1120                        XORI    $7 2
1121                        SUB     $8 $7
1122                        ADDI    $8 -2
1123                        EXCH    $9 $6
1124                        XOR     $8 $9
1125                        EXCH    $9 $6
1126                        XORI    $7 2
1127                        ADDI    $6 -4
1128                        SUB     $6 $3
1129                        XORI    $11 4
1130 assArrElem_41:        ADD     $10 $11
1131                        XORI    $11 4
1132                        ADD     $6 $3
1133                        ADDI    $6 4
1134                        XORI    $7 2
1135                        EXCH    $9 $6
1136                        XOR     $8 $9
1137                        EXCH    $9 $6
1138                        ADDI    $8 2
1139                        ADD     $8 $7
1140                        XORI    $7 2
1141                        ADDI    $6 -4
1142                        SUB     $6 $3
1143                        EXCH    $10 $8
1144                        ADD     $6 $3
1145                        ADDI    $6 4
1146                        XORI    $7 2
1147                        SUB     $8 $7
1148                        ADDI    $8 -2
1149                        EXCH    $9 $6
```

```
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 2
                        ADDI    $6 -4
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 3
                        XORI    $7 3
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 3
                        ADDI    $6 -3
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 3
                        XORI    $7 3
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 3
                        ADDI    $6 -3
                        SUB     $6 $3
                        XORI    $11 3
assArrElem_42:          ADD     $10 $11
                        XORI    $11 3
                        ADD     $6 $3
                        ADDI    $6 3
                        XORI    $7 3
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 3
                        ADDI    $6 -3
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 3
                        XORI    $7 3
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 3
                        ADDI    $6 -3
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 3
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 3
                        ADDI    $6 -5
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 3
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 3
                        ADDI    $6 -5
                        SUB     $6 $3
                        XORI    $11 1
assArrElem_43:          ADD     $10 $11
                        XORI    $11 1
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 3
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 3
                        ADDI    $6 -5
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 3
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 3
                        ADDI    $6 -5
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 3
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 3
                        ADDI    $6 -6
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 3
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 3
                        ADDI    $6 -6
                        SUB     $6 $3
assArrElem_44:          ADD     $10 $0
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 3
                        EXCH    $9 $6
                        XOR     $8 $9
```

```
                    EXCH    $9 $6
                    ADDI    $8 2
                    ADD     $8 $7
                    XORI    $7 3
                    ADDI    $6 -6
                    SUB     $6 $3
                    EXCH    $10 $8
                    ADD     $6 $3
                    ADDI    $6 6
                    XORI    $7 3
                    SUB     $8 $7
                    ADDI    $8 -2
                    EXCH    $9 $6
                    XOR     $8 $9
                    EXCH    $9 $6
                    XORI    $7 3
                    ADDI    $6 -6
                    SUB     $6 $3
                    ADD     $6 $3
                    ADDI    $6 4
                    XORI    $7 3
                    EXCH    $9 $6
                    XOR     $8 $9
                    EXCH    $9 $6
                    ADDI    $8 2
                    ADD     $8 $7
                    XORI    $7 3
                    ADDI    $6 -4
                    SUB     $6 $3
                    EXCH    $10 $8
                    ADD     $6 $3
                    ADDI    $6 4
                    XORI    $7 3
                    SUB     $8 $7
                    ADDI    $8 -2
                    EXCH    $9 $6
                    XOR     $8 $9
                    EXCH    $9 $6
                    XORI    $7 3
                    ADDI    $6 -4
                    SUB     $6 $3
                    XORI    $11 2
assArrElem_45:      ADD     $10 $11
                    XORI    $11 2
                    ADD     $6 $3
                    ADDI    $6 4
                    XORI    $7 3
                    EXCH    $9 $6
                    XOR     $8 $9
                    EXCH    $9 $6
                    ADDI    $8 2
                    ADD     $8 $7
                    XORI    $7 3
                    ADDI    $6 -4
                    SUB     $6 $3
                    EXCH    $10 $8
                    ADD     $6 $3
                    ADDI    $6 4
                    XORI    $7 3
                    SUB     $8 $7
                    ADDI    $8 -2
                    EXCH    $9 $6
                    XOR     $8 $9
                    EXCH    $9 $6
                    XORI    $7 3
                    ADDI    $6 -4
                    SUB     $6 $3
                    ADD     $6 $3
                    ADDI    $6 3
                    XORI    $7 4
                    EXCH    $9 $6
                    XOR     $8 $9
                    EXCH    $9 $6
                    ADDI    $8 2
                    ADD     $8 $7
                    XORI    $7 4
                    ADDI    $6 -3
                    SUB     $6 $3
                    EXCH    $10 $8
                    ADD     $6 $3
                    ADDI    $6 3
                    XORI    $7 4
                    SUB     $8 $7
                    ADDI    $8 -2
                    EXCH    $9 $6
                    XOR     $8 $9
                    EXCH    $9 $6
                    XORI    $7 4
                    ADDI    $6 -3
                    SUB     $6 $3
                    XORI    $11 3
assArrElem_46:      ADD     $10 $11
                    XORI    $11 3
                    ADD     $6 $3
                    ADDI    $6 3
                    XORI    $7 4
                    EXCH    $9 $6
                    XOR     $8 $9
                    EXCH    $9 $6
                    ADDI    $8 2
                    ADD     $8 $7
                    XORI    $7 4
                    ADDI    $6 -3
                    SUB     $6 $3
                    EXCH    $10 $8
                    ADD     $6 $3
                    ADDI    $6 3
                    XORI    $7 4
                    SUB     $8 $7
                    ADDI    $8 -2
                    EXCH    $9 $6
                    XOR     $8 $9
                    EXCH    $9 $6
                    XORI    $7 4
                    ADDI    $6 -3
                    SUB     $6 $3
                    ADD     $6 $3
                    ADDI    $6 5
                    XORI    $7 4
                    EXCH    $9 $6
                    XOR     $8 $9
                    EXCH    $9 $6
                    ADDI    $8 2
                    ADD     $8 $7
                    XORI    $7 4
                    ADDI    $6 -5
                    SUB     $6 $3
                    EXCH    $10 $8
                    ADD     $6 $3
                    ADDI    $6 5
                    XORI    $7 4
                    SUB     $8 $7
```

```
1414                        ADDI   $8 -2
1415                        EXCH   $9 $6
1416                        XOR    $8 $9
1417                        EXCH   $9 $6
1418                        XORI   $7 4
1419                        ADDI   $6 -5
1420                        SUB    $6 $3
1421                        ADD    $11 $3
1422                        ADDI   $11 10
1423                        EXCH   $12 $11
1424                        ADDI   $11 -10
1425                        SUB    $11 $3
1426 assArrElem_47:        ADD    $10 $12
1427                        ADD    $11 $3
1428                        ADDI   $11 10
1429                        EXCH   $12 $11
1430                        ADDI   $11 -10
1431                        SUB    $11 $3
1432                        ADD    $6 $3
1433                        ADDI   $6 5
1434                        XORI   $7 4
1435                        EXCH   $9 $6
1436                        XOR    $8 $9
1437                        EXCH   $9 $6
1438                        ADDI   $8 2
1439                        ADD    $8 $7
1440                        XORI   $7 4
1441                        ADDI   $6 -5
1442                        SUB    $6 $3
1443                        EXCH   $10 $8
1444                        ADD    $6 $3
1445                        ADDI   $6 5
1446                        XORI   $7 4
1447                        SUB    $8 $7
1448                        ADDI   $8 -2
1449                        EXCH   $9 $6
1450                        XOR    $8 $9
1451                        EXCH   $9 $6
1452                        XORI   $7 4
1453                        ADDI   $6 -5
1454                        SUB    $6 $3
1455                        ADD    $6 $3
1456                        ADDI   $6 6
1457                        XORI   $7 4
1458                        EXCH   $9 $6
1459                        XOR    $8 $9
1460                        EXCH   $9 $6
1461                        ADDI   $8 2
1462                        ADD    $8 $7
1463                        XORI   $7 4
1464                        ADDI   $6 -6
1465                        SUB    $6 $3
1466                        EXCH   $10 $8
1467                        ADD    $6 $3
1468                        ADDI   $6 6
1469                        XORI   $7 4
1470                        SUB    $8 $7
1471                        ADDI   $8 -2
1472                        EXCH   $9 $6
1473                        XOR    $8 $9
1474                        EXCH   $9 $6
1475                        XORI   $7 4
1476                        ADDI   $6 -6
1477                        SUB    $6 $3
1478                        ADD    $11 $3
1479                        ADDI   $11 10
1480                        EXCH   $12 $11
1481                        ADDI   $11 -10
1482                        SUB    $11 $3
1483 assArrElem_48:        ADD    $10 $12
1484                        ADD    $11 $3
1485                        ADDI   $11 10
1486                        EXCH   $12 $11
1487                        ADDI   $11 -10
1488                        SUB    $11 $3
1489                        ADD    $6 $3
1490                        ADDI   $6 6
1491                        XORI   $7 4
1492                        EXCH   $9 $6
1493                        XOR    $8 $9
1494                        EXCH   $9 $6
1495                        ADDI   $8 2
1496                        ADD    $8 $7
1497                        XORI   $7 4
1498                        ADDI   $6 -6
1499                        SUB    $6 $3
1500                        EXCH   $10 $8
1501                        ADD    $6 $3
1502                        ADDI   $6 6
1503                        XORI   $7 4
1504                        SUB    $8 $7
1505                        ADDI   $8 -2
1506                        EXCH   $9 $6
1507                        XOR    $8 $9
1508                        EXCH   $9 $6
1509                        XORI   $7 4
1510                        ADDI   $6 -6
1511                        SUB    $6 $3
1512                        ADD    $6 $3
1513                        ADDI   $6 4
1514                        XORI   $7 4
1515                        EXCH   $9 $6
1516                        XOR    $8 $9
1517                        EXCH   $9 $6
1518                        ADDI   $8 2
1519                        ADD    $8 $7
1520                        XORI   $7 4
1521                        ADDI   $6 -4
1522                        SUB    $6 $3
1523                        EXCH   $10 $8
1524                        ADD    $6 $3
1525                        ADDI   $6 4
1526                        XORI   $7 4
1527                        SUB    $8 $7
1528                        ADDI   $8 -2
1529                        EXCH   $9 $6
1530                        XOR    $8 $9
1531                        EXCH   $9 $6
1532                        XORI   $7 4
1533                        ADDI   $6 -4
1534                        SUB    $6 $3
1535                        XORI   $11 4
1536 assArrElem_49:        ADD    $10 $11
1537                        XORI   $11 4
1538                        ADD    $6 $3
1539                        ADDI   $6 4
1540                        XORI   $7 4
1541                        EXCH   $9 $6
1542                        XOR    $8 $9
1543                        EXCH   $9 $6
1544                        ADDI   $8 2
1545                        ADD    $8 $7
```

```
1546                        XORI    $7 4
1547                        ADDI    $6 -4
1548                        SUB     $6 $3
1549                        EXCH    $10 $8
1550                        ADD     $6 $3
1551                        ADDI    $6 4
1552                        XORI    $7 4
1553                        SUB     $8 $7
1554                        ADDI    $8 -2
1555                        EXCH    $9 $6
1556                        XOR     $8 $9
1557                        EXCH    $9 $6
1558                        XORI    $7 4
1559                        ADDI    $6 -4
1560                        SUB     $6 $3
1561                        ADD     $6 $3
1562                        ADDI    $6 3
1563                        XORI    $7 5
1564                        EXCH    $9 $6
1565                        XOR     $8 $9
1566                        EXCH    $9 $6
1567                        ADDI    $8 2
1568                        ADD     $8 $7
1569                        XORI    $7 5
1570                        ADDI    $6 -3
1571                        SUB     $6 $3
1572                        EXCH    $10 $8
1573                        ADD     $6 $3
1574                        ADDI    $6 3
1575                        XORI    $7 5
1576                        SUB     $8 $7
1577                        ADDI    $8 -2
1578                        EXCH    $9 $6
1579                        XOR     $8 $9
1580                        EXCH    $9 $6
1581                        XORI    $7 5
1582                        ADDI    $6 -3
1583                        SUB     $6 $3
1584                        XORI    $11 4
1585 assArrElem_50:         ADD     $10 $11
1586                        XORI    $11 4
1587                        ADD     $6 $3
1588                        ADDI    $6 3
1589                        XORI    $7 5
1590                        EXCH    $9 $6
1591                        XOR     $8 $9
1592                        EXCH    $9 $6
1593                        ADDI    $8 2
1594                        ADD     $8 $7
1595                        XORI    $7 5
1596                        ADDI    $6 -3
1597                        SUB     $6 $3
1598                        EXCH    $10 $8
1599                        ADD     $6 $3
1600                        ADDI    $6 3
1601                        XORI    $7 5
1602                        SUB     $8 $7
1603                        ADDI    $8 -2
1604                        EXCH    $9 $6
1605                        XOR     $8 $9
1606                        EXCH    $9 $6
1607                        XORI    $7 5
1608                        ADDI    $6 -3
1609                        SUB     $6 $3
1610                        ADD     $6 $3
1611                        ADDI    $6 5
1612                        XORI    $7 5
1613                        EXCH    $9 $6
1614                        XOR     $8 $9
1615                        EXCH    $9 $6
1616                        ADDI    $8 2
1617                        ADD     $8 $7
1618                        XORI    $7 5
1619                        ADDI    $6 -5
1620                        SUB     $6 $3
1621                        EXCH    $10 $8
1622                        ADD     $6 $3
1623                        ADDI    $6 5
1624                        XORI    $7 5
1625                        SUB     $8 $7
1626                        ADDI    $8 -2
1627                        EXCH    $9 $6
1628                        XOR     $8 $9
1629                        EXCH    $9 $6
1630                        XORI    $7 5
1631                        ADDI    $6 -5
1632                        SUB     $6 $3
1633                        ADD     $11 $3
1634                        ADDI    $11 7
1635                        EXCH    $12 $11
1636                        ADDI    $11 -7
1637                        SUB     $11 $3
1638 assArrElem_51:         ADD     $10 $12
1639                        ADD     $11 $3
1640                        ADDI    $11 7
1641                        EXCH    $12 $11
1642                        ADDI    $11 -7
1643                        SUB     $11 $3
1644                        ADD     $6 $3
1645                        ADDI    $6 5
1646                        XORI    $7 5
1647                        EXCH    $9 $6
1648                        XOR     $8 $9
1649                        EXCH    $9 $6
1650                        ADDI    $8 2
1651                        ADD     $8 $7
1652                        XORI    $7 5
1653                        ADDI    $6 -5
1654                        SUB     $6 $3
1655                        EXCH    $10 $8
1656                        ADD     $6 $3
1657                        ADDI    $6 5
1658                        XORI    $7 5
1659                        SUB     $8 $7
1660                        ADDI    $8 -2
1661                        EXCH    $9 $6
1662                        XOR     $8 $9
1663                        EXCH    $9 $6
1664                        XORI    $7 5
1665                        ADDI    $6 -5
1666                        SUB     $6 $3
1667                        ADD     $6 $3
1668                        ADDI    $6 6
1669                        XORI    $7 5
1670                        EXCH    $9 $6
1671                        XOR     $8 $9
1672                        EXCH    $9 $6
1673                        ADDI    $8 2
1674                        ADD     $8 $7
1675                        XORI    $7 5
1676                        ADDI    $6 -6
1677                        SUB     $6 $3
```

```
1678                        EXCH    $10 $8
1679                        ADD     $6 $3
1680                        ADDI    $6 6
1681                        XORI    $7 5
1682                        SUB     $8 $7
1683                        ADDI    $8 -2
1684                        EXCH    $9 $6
1685                        XOR     $8 $9
1686                        EXCH    $9 $6
1687                        XORI    $7 5
1688                        ADDI    $6 -6
1689                        SUB     $6 $3
1690                        ADD     $11 $3
1691                        ADDI    $11 8
1692                        EXCH    $12 $11
1693                        ADDI    $11 -8
1694                        SUB     $11 $3
1695 assArrElem_52:         ADD     $10 $12
1696                        ADD     $11 $3
1697                        ADDI    $11 8
1698                        EXCH    $12 $11
1699                        ADDI    $11 -8
1700                        SUB     $11 $3
1701                        ADD     $6 $3
1702                        ADDI    $6 6
1703                        XORI    $7 5
1704                        EXCH    $9 $6
1705                        XOR     $8 $9
1706                        EXCH    $9 $6
1707                        ADDI    $8 2
1708                        ADD     $8 $7
1709                        XORI    $7 5
1710                        ADDI    $6 -6
1711                        SUB     $6 $3
1712                        EXCH    $10 $8
1713                        ADD     $6 $3
1714                        ADDI    $6 6
1715                        XORI    $7 5
1716                        SUB     $8 $7
1717                        ADDI    $8 -2
1718                        EXCH    $9 $6
1719                        XOR     $8 $9
1720                        EXCH    $9 $6
1721                        XORI    $7 5
1722                        ADDI    $6 -6
1723                        SUB     $6 $3
1724                        ADD     $6 $3
1725                        ADDI    $6 4
1726                        XORI    $7 5
1727                        EXCH    $9 $6
1728                        XOR     $8 $9
1729                        EXCH    $9 $6
1730                        ADDI    $8 2
1731                        ADD     $8 $7
1732                        XORI    $7 5
1733                        ADDI    $6 -4
1734                        SUB     $6 $3
1735                        EXCH    $10 $8
1736                        ADD     $6 $3
1737                        ADDI    $6 4
1738                        XORI    $7 5
1739                        SUB     $8 $7
1740                        ADDI    $8 -2
1741                        EXCH    $9 $6
1742                        XOR     $8 $9
1743                        EXCH    $9 $6
1744                        XORI    $7 5
1745                        ADDI    $6 -4
1746                        SUB     $6 $3
1747                        XORI    $11 5
1748 assArrElem_53:         ADD     $10 $11
1749                        XORI    $11 5
1750                        ADD     $6 $3
1751                        ADDI    $6 4
1752                        XORI    $7 5
1753                        EXCH    $9 $6
1754                        XOR     $8 $9
1755                        EXCH    $9 $6
1756                        ADDI    $8 2
1757                        ADD     $8 $7
1758                        XORI    $7 5
1759                        ADDI    $6 -4
1760                        SUB     $6 $3
1761                        EXCH    $10 $8
1762                        ADD     $6 $3
1763                        ADDI    $6 4
1764                        XORI    $7 5
1765                        SUB     $8 $7
1766                        ADDI    $8 -2
1767                        EXCH    $9 $6
1768                        XOR     $8 $9
1769                        EXCH    $9 $6
1770                        XORI    $7 5
1771                        ADDI    $6 -4
1772                        SUB     $6 $3
1773                        ADD     $6 $3
1774                        ADDI    $6 3
1775                        XORI    $7 6
1776                        EXCH    $9 $6
1777                        XOR     $8 $9
1778                        EXCH    $9 $6
1779                        ADDI    $8 2
1780                        ADD     $8 $7
1781                        XORI    $7 6
1782                        ADDI    $6 -3
1783                        SUB     $6 $3
1784                        EXCH    $10 $8
1785                        ADD     $6 $3
1786                        ADDI    $6 3
1787                        XORI    $7 6
1788                        SUB     $8 $7
1789                        ADDI    $8 -2
1790                        EXCH    $9 $6
1791                        XOR     $8 $9
1792                        EXCH    $9 $6
1793                        XORI    $7 6
1794                        ADDI    $6 -3
1795                        SUB     $6 $3
1796                        XORI    $11 5
1797 assArrElem_54:         ADD     $10 $11
1798                        XORI    $11 5
1799                        ADD     $6 $3
1800                        ADDI    $6 3
1801                        XORI    $7 6
1802                        EXCH    $9 $6
1803                        XOR     $8 $9
1804                        EXCH    $9 $6
1805                        ADDI    $8 2
1806                        ADD     $8 $7
1807                        XORI    $7 6
1808                        ADDI    $6 -3
1809                        SUB     $6 $3
```

```
1810                            EXCH    $10 $8
1811                            ADD     $6 $3
1812                            ADDI    $6 3
1813                            XORI    $7 6
1814                            SUB     $8 $7
1815                            ADDI    $8 -2
1816                            EXCH    $9 $6
1817                            XOR     $8 $9
1818                            EXCH    $9 $6
1819                            XORI    $7 6
1820                            ADDI    $6 -3
1821                            SUB     $6 $3
1822                            ADD     $6 $3
1823                            ADDI    $6 5
1824                            XORI    $7 6
1825                            EXCH    $9 $6
1826                            XOR     $8 $9
1827                            EXCH    $9 $6
1828                            ADDI    $8 2
1829                            ADD     $8 $7
1830                            XORI    $7 6
1831                            ADDI    $6 -5
1832                            SUB     $6 $3
1833                            EXCH    $10 $8
1834                            ADD     $6 $3
1835                            ADDI    $6 5
1836                            XORI    $7 6
1837                            SUB     $8 $7
1838                            ADDI    $8 -2
1839                            EXCH    $9 $6
1840                            XOR     $8 $9
1841                            EXCH    $9 $6
1842                            XORI    $7 6
1843                            ADDI    $6 -5
1844                            SUB     $6 $3
1845 assArrElem_55:            ADD     $10 $0
1846                            ADD     $6 $3
1847                            ADDI    $6 5
1848                            XORI    $7 6
1849                            EXCH    $9 $6
1850                            XOR     $8 $9
1851                            EXCH    $9 $6
1852                            ADDI    $8 2
1853                            ADD     $8 $7
1854                            XORI    $7 6
1855                            ADDI    $6 -5
1856                            SUB     $6 $3
1857                            EXCH    $10 $8
1858                            ADD     $6 $3
1859                            ADDI    $6 5
1860                            XORI    $7 6
1861                            SUB     $8 $7
1862                            ADDI    $8 -2
1863                            EXCH    $9 $6
1864                            XOR     $8 $9
1865                            EXCH    $9 $6
1866                            XORI    $7 6
1867                            ADDI    $6 -5
1868                            SUB     $6 $3
1869                            ADD     $6 $3
1870                            ADDI    $6 6
1871                            XORI    $7 6
1872                            EXCH    $9 $6
1873                            XOR     $8 $9
1874                            EXCH    $9 $6
1875                            ADDI    $8 2
1876                            ADD     $8 $7
1877                            XORI    $7 6
1878                            ADDI    $6 -6
1879                            SUB     $6 $3
1880                            EXCH    $10 $8
1881                            ADD     $6 $3
1882                            ADDI    $6 6
1883                            XORI    $7 6
1884                            SUB     $8 $7
1885                            ADDI    $8 -2
1886                            EXCH    $9 $6
1887                            XOR     $8 $9
1888                            EXCH    $9 $6
1889                            XORI    $7 6
1890                            ADDI    $6 -6
1891                            SUB     $6 $3
1892 assArrElem_56:            ADD     $10 $0
1893                            ADD     $6 $3
1894                            ADDI    $6 6
1895                            XORI    $7 6
1896                            EXCH    $9 $6
1897                            XOR     $8 $9
1898                            EXCH    $9 $6
1899                            ADDI    $8 2
1900                            ADD     $8 $7
1901                            XORI    $7 6
1902                            ADDI    $6 -6
1903                            SUB     $6 $3
1904                            EXCH    $10 $8
1905                            ADD     $6 $3
1906                            ADDI    $6 6
1907                            XORI    $7 6
1908                            SUB     $8 $7
1909                            ADDI    $8 -2
1910                            EXCH    $9 $6
1911                            XOR     $8 $9
1912                            EXCH    $9 $6
1913                            XORI    $7 6
1914                            ADDI    $6 -6
1915                            SUB     $6 $3
1916                            ADD     $6 $3
1917                            ADDI    $6 4
1918                            XORI    $7 6
1919                            EXCH    $9 $6
1920                            XOR     $8 $9
1921                            EXCH    $9 $6
1922                            ADDI    $8 2
1923                            ADD     $8 $7
1924                            XORI    $7 6
1925                            ADDI    $6 -4
1926                            SUB     $6 $3
1927                            EXCH    $10 $8
1928                            ADD     $6 $3
1929                            ADDI    $6 4
1930                            XORI    $7 6
1931                            SUB     $8 $7
1932                            ADDI    $8 -2
1933                            EXCH    $9 $6
1934                            XOR     $8 $9
1935                            EXCH    $9 $6
1936                            XORI    $7 6
1937                            ADDI    $6 -4
1938                            SUB     $6 $3
1939                            XORI    $11 4
1940 assArrElem_57:            ADD     $10 $11
1941                            XORI    $11 4
```

```
                        ADD     $6 $3
                        ADDI    $6 4
                        XORI    $7 6
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 6
                        ADDI    $6 -4
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 4
                        XORI    $7 6
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 6
                        ADDI    $6 -4
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 3
                        XORI    $7 7
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 7
                        ADDI    $6 -3
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 3
                        XORI    $7 7
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 6
                        ADDI    $6 -3
                        SUB     $6 $3
                        XORI    $11 5
assArrElem_58:          ADD     $10 $11
                        XORI    $11 5
                        ADD     $6 $3
                        ADDI    $6 3
                        XORI    $7 7
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 7
                        ADDI    $6 -3
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 3
                        XORI    $7 7
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 7
                        ADDI    $6 -3
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 7
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 7
                        ADDI    $6 -5
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 7
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 7
                        ADDI    $6 -5
                        SUB     $6 $3
                        ADD     $11 $3
                        ADDI    $11 10
                        EXCH    $12 $11
                        ADDI    $11 -10
                        SUB     $11 $3
assArrElem_59:          ADD     $10 $12
                        ADD     $11 $3
                        ADDI    $11 10
                        EXCH    $12 $11
                        ADDI    $11 -10
                        SUB     $11 $3
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 7
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 7
                        ADDI    $6 -5
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 5
                        XORI    $7 7
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 7
                        ADDI    $6 -5
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 7
```

```
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 7
                        ADDI    $6 -6
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 7
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 7
                        ADDI    $6 -6
                        SUB     $6 $3
                        ADD     $11 $3
                        ADDI    $11 10
                        EXCH    $12 $11
                        ADDI    $11 -10
                        SUB     $11 $3
assArrElem_60:          ADD     $10 $12
                        ADD     $11 $3
                        ADDI    $11 10
                        EXCH    $12 $11
                        ADDI    $11 -10
                        SUB     $11 $3
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 7
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 7
                        ADDI    $6 -6
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 7
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 7
                        ADDI    $6 -6
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 4
                        XORI    $7 7
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 7
                        ADDI    $6 -4
                        SUB     $6 $3
                        EXCH    $10 $8
```

```
                        ADD     $6 $3
                        ADDI    $6 4
                        XORI    $7 7
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 7
                        ADDI    $6 -4
                        SUB     $6 $3
                        XORI    $11 6
assArrElem_61:          ADD     $10 $11
                        XORI    $11 6
                        ADD     $6 $3
                        ADDI    $6 4
                        XORI    $7 7
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 7
                        ADDI    $6 -4
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 4
                        XORI    $7 7
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 7
                        ADDI    $6 -4
                        SUB     $6 $3
l_initRules_2_bot:      BRA
    l_initRules_2_top
l_initTape_3_top:       BRA
    l_initTape_3_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $3 $1
                        ADDI    $1 -1
l_initTape_3:           SWAPBR $2
                        NEG     $2
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
localBlock_149:         XOR     $6 $1
                        XOR     $7 $0
                        EXCH    $7 $1
                        ADDI    $1 -1
localBlock_148:         XOR     $7 $1
                        XOR     $8 $0
                        EXCH    $8 $1
                        ADDI    $1 -1
localBlock_147:         XOR     $8 $1
                        XOR     $9 $0
                        EXCH    $9 $1
                        ADDI    $1 -1
localBlock_146:         XOR     $9 $1
                        XOR     $10 $0
                        EXCH    $10 $1
```

```
                        ADDI    $1 -1
localBlock_145:         XOR     $10 $1
                        XOR     $11 $0
                        EXCH    $11 $1
                        ADDI    $1 -1
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
obj_con_62:             ADDI    $12 8
                        EXCH    $12 $1
                        ADDI    $1 -1
                        EXCH    $11 $1
                        ADDI    $1 -1
                        BRA     l_malloc
                        ADDI    $1 1
                        EXCH    $11 $1
                        ADDI    $1 1
                        EXCH    $12 $1
obj_con_62_i:           ADDI    $12 -8
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        XORI    $12 10
                        EXCH    $12 $11
                        ADDI    $11 1
                        XORI    $12 1
                        EXCH    $12 $11
obj_con_62_bot:         ADDI    $11 -1
                        EXCH    $11 $6
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
obj_con_63:             ADDI    $12 8
                        EXCH    $12 $1
                        ADDI    $1 -1
                        EXCH    $11 $1
                        ADDI    $1 -1
                        BRA     l_malloc
                        ADDI    $1 1
                        EXCH    $11 $1
                        ADDI    $1 1
                        EXCH    $12 $1
obj_con_63_i:           ADDI    $12 -8
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        XORI    $12 10
                        EXCH    $12 $11
                        ADDI    $11 1
                        XORI    $12 1
                        EXCH    $12 $11
obj_con_63_bot:         ADDI    $11 -1
                        EXCH    $11 $7
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
obj_con_64:             ADDI    $12 8
                        EXCH    $12 $1
                        ADDI    $1 -1
                        EXCH    $11 $1
                        ADDI    $1 -1
                        BRA     l_malloc
                        ADDI    $1 1
                        EXCH    $11 $1
                        ADDI    $1 1
                        EXCH    $12 $1
obj_con_64_i:           ADDI    $12 -8
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        XORI    $12 10
                        EXCH    $12 $11
                        ADDI    $11 1
                        XORI    $12 1
                        EXCH    $12 $11
obj_con_64_bot:         ADDI    $11 -1
                        EXCH    $11 $8
                        EXCH    $3 $1
```

```
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
obj_con_65:             ADDI    $12 8
                        EXCH    $12 $1
                        ADDI    $1 -1
                        EXCH    $11 $1
                        ADDI    $1 -1
                        BRA     l_malloc
                        ADDI    $1 1
                        EXCH    $11 $1
                        ADDI    $1 1
                        EXCH    $12 $1
obj_con_65_i:           ADDI    $12 -8
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        XORI    $12 10
                        EXCH    $12 $11
                        ADDI    $11 1
                        XORI    $12 1
                        EXCH    $12 $11
obj_con_65_bot:         ADDI    $11 -1
                        EXCH    $11 $9
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
obj_con_66:             ADDI    $12 8
                        EXCH    $12 $1
                        ADDI    $1 -1
                        EXCH    $11 $1
                        ADDI    $1 -1
                        BRA     l_malloc
                        ADDI    $1 1
                        EXCH    $11 $1
                        ADDI    $1 1
                        EXCH    $12 $1
obj_con_66_i:           ADDI    $12 -8
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        XORI    $12 10
                        EXCH    $12 $11
                        ADDI    $11 1
                        XORI    $12 1
                        EXCH    $12 $11
obj_con_66_bot:         ADDI    $11 -1
                        EXCH    $11 $10
                        ADD     $11 $3
                        ADDI    $11 14
                        EXCH    $12 $11
                        ADDI    $11 -14
                        SUB     $11 $3
                        ADD     $13 $3
                        ADDI    $13 10
                        EXCH    $14 $13
                        ADDI    $13 -10
                        SUB     $13 $3
                        ADD     $12 $14
                        ADD     $13 $3
                        ADDI    $13 10
                        EXCH    $14 $13
                        ADDI    $13 -10
                        SUB     $13 $3
                        ADD     $11 $3
                        ADDI    $11 14
                        EXCH    $12 $11
                        ADDI    $11 -14
                        SUB     $11 $3
                        EXCH    $11 $6
                        XOR     $12 $11
loadMetAdd_67:          EXCH    $13 $12
                        ADDI    $13 3
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -3
                        EXCH    $13 $12
                        EXCH    $11 $6
                        ADD     $16 $3
                        ADDI    $16 14
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $16 $1
                        ADDI    $1 -1
                        EXCH    $12 $1
                        ADDI    $1 -1
```

```
                        ADDI    $15 -2468
l_rjmp_top_69:          RBRA    l_rjmp_bot_70
l_jmp_68:               SWAPBR  $15
                        NEG     $15
l_rjmp_bot_70:          BRA     l_rjmp_top_69
                        ADDI    $15 2468
                        ADDI    $1 1
                        EXCH    $12 $1
                        ADDI    $1 1
                        EXCH    $16 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $16 -14
                        SUB     $16 $3
                        EXCH    $11 $6
                        EXCH    $13 $12
                        ADDI    $13 3
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -3
loadMetAdd_67_i:        EXCH    $13 $12
                        XOR     $12 $11
                        EXCH    $11 $6
                        ADD     $11 $3
                        ADDI    $11 14
                        EXCH    $12 $11
                        ADDI    $11 -14
                        SUB     $11 $3
                        XORI    $13 1
                        ADD     $12 $13
                        XORI    $13 1
                        ADD     $11 $3
                        ADDI    $11 14
                        EXCH    $12 $11
                        ADDI    $11 -14
                        SUB     $11 $3
                        EXCH    $11 $7
                        XOR     $12 $11
loadMetAdd_71:          EXCH    $13 $12
                        ADDI    $13 3
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -3
                        EXCH    $13 $12
                        EXCH    $11 $7
                        ADD     $16 $3
                        ADDI    $16 14
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $16 $1
                        ADDI    $1 -1
                        EXCH    $12 $1
                        ADDI    $1 -1
                        ADDI    $15 -2543
l_rjmp_top_73:          RBRA    l_rjmp_bot_74
l_jmp_72:               SWAPBR  $15
                        NEG     $15
l_rjmp_bot_74:          BRA     l_rjmp_top_73
                        ADDI    $15 2543
                        ADDI    $1 1
                        EXCH    $12 $1
                        ADDI    $1 1
                        EXCH    $16 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $16 -14
                        SUB     $16 $3
                        EXCH    $11 $7
                        EXCH    $13 $12
                        ADDI    $13 3
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -3
loadMetAdd_71_i:        EXCH    $13 $12
                        XOR     $12 $11
                        EXCH    $11 $7
                        ADD     $11 $3
                        ADDI    $11 14
                        EXCH    $12 $11
                        ADDI    $11 -14
                        SUB     $11 $3
                        XORI    $13 1
                        ADD     $12 $13
                        XORI    $13 1
                        ADD     $11 $3
                        ADDI    $11 14
                        EXCH    $12 $11
                        ADDI    $11 -14
                        SUB     $11 $3
                        EXCH    $11 $8
                        XOR     $12 $11
loadMetAdd_75:          EXCH    $13 $12
                        ADDI    $13 3
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -3
                        EXCH    $13 $12
                        EXCH    $11 $8
```

```
                        ADD     $16 $3
                        ADDI    $16 14
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $16 $1
                        ADDI    $1 -1
                        EXCH    $12 $1
                        ADDI    $1 -1
                        ADDI    $15 -2618
l_rjmp_top_77:          RBRA    l_rjmp_bot_78
l_jmp_76:               SWAPBR  $15
                        NEG     $15
l_rjmp_bot_78:          BRA     l_rjmp_top_77
                        ADDI    $15 2618
                        ADDI    $1 1
                        EXCH    $12 $1
                        ADDI    $1 1
                        EXCH    $16 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $16 -14
                        SUB     $16 $3
                        EXCH    $11 $8
                        EXCH    $13 $12
                        ADDI    $13 3
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -3
loadMetAdd_75_i:        EXCH    $13 $12
                        XOR     $12 $11
                        EXCH    $11 $8
                        ADD     $11 $3
                        ADDI    $11 14
                        EXCH    $12 $11
                        ADDI    $11 -14
                        SUB     $11 $3
                        XORI    $13 1
                        ADD     $12 $13
                        XORI    $13 1
                        ADD     $11 $3
                        ADDI    $11 14
                        EXCH    $12 $11
                        ADDI    $11 -14
                        SUB     $11 $3
                        EXCH    $11 $10
                        XOR     $12 $11
loadMetAdd_79:          EXCH    $13 $12
                        ADDI    $13 3
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -3
                        EXCH    $13 $12
                        EXCH    $11 $10
                        ADD     $16 $3
                        ADDI    $16 14
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $16 $1
                        ADDI    $1 -1
                        EXCH    $12 $1
                        ADDI    $1 -1
                        ADDI    $15 -2693
l_rjmp_top_81:          RBRA    l_rjmp_bot_82
l_jmp_80:               SWAPBR  $15
                        NEG     $15
l_rjmp_bot_82:          BRA     l_rjmp_top_81
                        ADDI    $15 2693
                        ADDI    $1 1
                        EXCH    $12 $1
                        ADDI    $1 1
                        EXCH    $16 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $16 -14
                        SUB     $16 $3
                        EXCH    $11 $10
                        EXCH    $13 $12
                        ADDI    $13 3
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -3
loadMetAdd_79_i:        EXCH    $13 $12
                        XOR     $12 $11
                        EXCH    $11 $10
                        ADD     $11 $3
                        ADDI    $11 2
                        EXCH    $12 $11
                        ADDI    $11 -2
                        SUB     $11 $3
```

```
2732                            EXCH    $13 $6
2733 swap_83:                  XOR     $12 $13
2734                            XOR     $13 $12
2735                            XOR     $12 $13
2736                            EXCH    $13 $6
2737                            ADD     $11 $3
2738                            ADDI    $11 2
2739                            EXCH    $12 $11
2740                            ADDI    $11 -2
2741                            SUB     $11 $3
2742                            EXCH    $11 $6
2743                            ADD     $12 $3
2744                            ADDI    $12 2
2745                            EXCH    $13 $12
2746                            ADDI    $12 -2
2747                            SUB     $12 $3
2748 copy_84:                   XOR     $11 $13
2749                            ADDI    $13 1
2750                            EXCH    $14 $13
2751                            ADDI    $14 1
2752                            EXCH    $14 $13
2753                            ADDI    $13 -1
2754                            ADD     $12 $3
2755                            ADDI    $12 2
2756                            EXCH    $13 $12
2757                            ADDI    $12 -2
2758                            SUB     $12 $3
2759                            EXCH    $11 $6
2760                            ADD     $11 $3
2761                            ADDI    $11 2
2762                            EXCH    $12 $11
2763                            ADDI    $11 -2
2764                            SUB     $11 $3
2765                            XOR     $13 $12
2766 loadMetAdd_85:             EXCH    $14 $13
2767                            ADDI    $14 2
2768                            EXCH    $15 $14
2769                            XOR     $16 $15
2770                            EXCH    $15 $14
2771                            ADDI    $14 -2
2772                            EXCH    $14 $13
2773                            ADD     $11 $3
2774                            ADDI    $11 2
2775                            EXCH    $12 $11
2776                            ADDI    $11 -2
2777                            SUB     $11 $3
2778                            EXCH    $3 $1
2779                            ADDI    $1 -1
2780                            EXCH    $10 $1
2781                            ADDI    $1 -1
2782                            EXCH    $9 $1
2783                            ADDI    $1 -1
2784                            EXCH    $8 $1
2785                            ADDI    $1 -1
2786                            EXCH    $7 $1
2787                            ADDI    $1 -1
2788                            EXCH    $6 $1
2789                            ADDI    $1 -1
2790                            EXCH    $13 $1
2791                            ADDI    $1 -1
2792                            ADDI    $16 -2792
2793 l_rjmp_top_87:             RBRA    l_rjmp_bot_88
2794 l_jmp_86:                  SWAPBR  $16
2795                            NEG     $16
2796 l_rjmp_bot_88:             BRA     l_rjmp_top_87
2797                            ADDI    $16 2792
2798                            ADDI    $1 1
2799                            EXCH    $13 $1
2800                            ADDI    $1 1
2801                            EXCH    $6 $1
2802                            ADDI    $1 1
2803                            EXCH    $7 $1
2804                            ADDI    $1 1
2805                            EXCH    $8 $1
2806                            ADDI    $1 1
2807                            EXCH    $9 $1
2808                            ADDI    $1 1
2809                            EXCH    $10 $1
2810                            ADDI    $1 1
2811                            EXCH    $3 $1
2812                            ADD     $11 $3
2813                            ADDI    $11 2
2814                            EXCH    $12 $11
2815                            ADDI    $11 -2
2816                            SUB     $11 $3
2817                            EXCH    $14 $13
2818                            ADDI    $14 2
2819                            EXCH    $15 $14
2820                            XOR     $16 $15
2821                            EXCH    $15 $14
2822                            ADDI    $14 -2
2823 loadMetAdd_85_i:           EXCH    $14 $13
2824                            XOR     $13 $12
2825                            ADD     $11 $3
2826                            ADDI    $11 2
2827                            EXCH    $12 $11
2828                            ADDI    $11 -2
2829                            SUB     $11 $3
2830                            EXCH    $11 $6
2831                            EXCH    $12 $7
2832 copy_89:                   XOR     $11 $12
2833                            ADDI    $12 1
2834                            EXCH    $13 $12
2835                            ADDI    $13 1
2836                            EXCH    $13 $12
2837                            ADDI    $12 -1
2838                            EXCH    $12 $7
2839                            EXCH    $11 $6
2840                            EXCH    $11 $7
2841                            XOR     $12 $11
2842 loadMetAdd_90:             EXCH    $13 $12
2843                            ADDI    $13 2
2844                            EXCH    $14 $13
2845                            XOR     $15 $14
2846                            EXCH    $14 $13
2847                            ADDI    $13 -2
2848                            EXCH    $13 $12
2849                            EXCH    $11 $7
2850                            EXCH    $3 $1
2851                            ADDI    $1 -1
2852                            EXCH    $10 $1
2853                            ADDI    $1 -1
2854                            EXCH    $9 $1
2855                            ADDI    $1 -1
2856                            EXCH    $8 $1
2857                            ADDI    $1 -1
2858                            EXCH    $7 $1
2859                            ADDI    $1 -1
2860                            EXCH    $6 $1
2861                            ADDI    $1 -1
2862                            EXCH    $12 $1
2863                            ADDI    $1 -1
```

```
                        ADDI   $15 -2864
l_rjmp_top_92:          RBRA   l_rjmp_bot_93
l_jmp_91:               SWAPBR $15
                        NEG    $15
l_rjmp_bot_93:          BRA    l_rjmp_top_92
                        ADDI   $15 2864
                        ADDI   $1 1
                        EXCH   $12 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        ADDI   $1 1
                        EXCH   $7 $1
                        ADDI   $1 1
                        EXCH   $8 $1
                        ADDI   $1 1
                        EXCH   $9 $1
                        ADDI   $1 1
                        EXCH   $10 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        EXCH   $11 $7
                        EXCH   $13 $12
                        ADDI   $13 2
                        EXCH   $14 $13
                        XOR    $15 $14
                        EXCH   $14 $13
                        ADDI   $13 -2
loadMetAdd_90_i:        EXCH   $13 $12
                        XOR    $12 $11
                        EXCH   $11 $7
                        EXCH   $11 $6
                        EXCH   $12 $8
copy_94:                XOR    $11 $12
                        ADDI   $12 1
                        EXCH   $13 $12
                        ADDI   $13 1
                        EXCH   $13 $12
                        ADDI   $12 -1
                        EXCH   $12 $8
                        EXCH   $11 $6
                        EXCH   $11 $8
                        XOR    $12 $11
loadMetAdd_95:          EXCH   $13 $12
                        ADDI   $13 2
                        EXCH   $14 $13
                        XOR    $15 $14
                        EXCH   $14 $13
                        ADDI   $13 -2
                        EXCH   $13 $12
                        EXCH   $11 $8
                        EXCH   $3 $1
                        ADDI   $1 -1
                        EXCH   $10 $1
                        ADDI   $1 -1
                        EXCH   $9 $1
                        ADDI   $1 -1
                        EXCH   $8 $1
                        ADDI   $1 -1
                        EXCH   $7 $1
                        ADDI   $1 -1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $12 $1
                        ADDI   $1 -1
                        ADDI   $15 -2928
l_rjmp_top_97:          RBRA   l_rjmp_bot_98
l_jmp_96:               SWAPBR $15
                        NEG    $15
l_rjmp_bot_98:          BRA    l_rjmp_top_97
                        ADDI   $15 2928
                        ADDI   $1 1
                        EXCH   $12 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        ADDI   $1 1
                        EXCH   $7 $1
                        ADDI   $1 1
                        EXCH   $8 $1
                        ADDI   $1 1
                        EXCH   $9 $1
                        ADDI   $1 1
                        EXCH   $10 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        EXCH   $11 $8
                        EXCH   $13 $12
                        ADDI   $13 2
                        EXCH   $14 $13
                        XOR    $15 $14
                        EXCH   $14 $13
                        ADDI   $13 -2
loadMetAdd_95_i:        EXCH   $13 $12
                        XOR    $12 $11
                        EXCH   $11 $8
                        EXCH   $11 $6
                        EXCH   $12 $9
copy_99:                XOR    $11 $12
                        ADDI   $12 1
                        EXCH   $13 $12
                        ADDI   $13 1
                        EXCH   $13 $12
                        ADDI   $12 -1
                        EXCH   $12 $9
                        EXCH   $11 $6
                        EXCH   $11 $9
                        XOR    $12 $11
loadMetAdd_100:         EXCH   $13 $12
                        ADDI   $13 2
                        EXCH   $14 $13
                        XOR    $15 $14
                        EXCH   $14 $13
                        ADDI   $13 -2
                        EXCH   $13 $12
                        EXCH   $11 $9
                        EXCH   $3 $1
                        ADDI   $1 -1
                        EXCH   $10 $1
                        ADDI   $1 -1
                        EXCH   $9 $1
                        ADDI   $1 -1
                        EXCH   $8 $1
                        ADDI   $1 -1
                        EXCH   $7 $1
                        ADDI   $1 -1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $12 $1
                        ADDI   $1 -1
                        ADDI   $15 -2992
l_rjmp_top_102:         RBRA   l_rjmp_bot_103
l_jmp_101:              SWAPBR $15
                        NEG    $15
```

```
l_rjmp_bot_103:         BRA     l_rjmp_top_102
                        ADDI    $15 2992
                        ADDI    $1 1
                        EXCH    $12 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $11 $9
                        EXCH    $13 $12
                        ADDI    $13 2
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -2
loadMetAdd_100_i:       EXCH    $13 $12
                        XOR     $12 $11
                        EXCH    $11 $9
                        EXCH    $11 $6
                        EXCH    $12 $10
copy_104:               XOR     $11 $12
                        ADDI    $12 1
                        EXCH    $13 $12
                        ADDI    $13 1
                        EXCH    $13 $12
                        ADDI    $12 -1
                        EXCH    $12 $10
                        EXCH    $11 $6
                        EXCH    $11 $10
                        XOR     $12 $11
loadMetAdd_105:         EXCH    $13 $12
                        ADDI    $13 2
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -2
                        EXCH    $13 $12
                        EXCH    $11 $10
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $12 $1
                        ADDI    $1 -1
                        ADDI    $15 -3056
l_rjmp_top_107:         RBRA    l_rjmp_bot_108
l_jmp_106:              SWAPBR  $15
                        NEG     $15
l_rjmp_bot_108:         BRA     l_rjmp_top_107
                        ADDI    $15 3056
                        ADDI    $1 1
                        EXCH    $12 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $11 $10
                        EXCH    $13 $12
                        ADDI    $13 2
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -2
loadMetAdd_105_i:       EXCH    $13 $12
                        XOR     $12 $11
                        EXCH    $11 $10
                        EXCH    $11 $6
                        EXCH    $12 $9
copy_109:               XOR     $11 $12
                        ADDI    $12 1
                        EXCH    $13 $12
                        ADDI    $13 1
                        EXCH    $13 $12
                        ADDI    $12 -1
                        EXCH    $12 $9
                        EXCH    $11 $6
                        EXCH    $11 $10
                        XOR     $12 $11
loadMetAdd_110:         EXCH    $13 $12
                        ADDI    $13 0
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 0
                        EXCH    $13 $12
                        EXCH    $11 $10
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $12 $1
                        ADDI    $1 -1
                        ADDI    $15 -3120
l_rjmp_top_112:         RBRA    l_rjmp_bot_113
l_jmp_111:              SWAPBR  $15
                        NEG     $15
l_rjmp_bot_113:         BRA     l_rjmp_top_112
                        ADDI    $15 3120
                        ADDI    $1 1
                        EXCH    $12 $1
```

```
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $11 $10
                        EXCH    $13 $12
                        ADDI    $13 0
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 0
loadMetAdd_110_i:       EXCH    $13 $12
                        XOR     $12 $11
                        EXCH    $11 $10
                        EXCH    $11 $9
                        XOR     $12 $11
loadMetAdd_114:         EXCH    $13 $12
                        ADDI    $13 1
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -1
                        EXCH    $13 $12
                        EXCH    $11 $9
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $12 $1
                        ADDI    $1 -1
                        ADDI    $15 -3174
l_rjmp_top_116:         RBRA    l_rjmp_bot_117
l_jmp_115:              SWAPBR  $15
                        NEG     $15
l_rjmp_bot_117:         BRA     l_rjmp_top_116
                        ADDI    $15 3174
                        ADDI    $1 1
                        EXCH    $12 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $11 $9
                        EXCH    $13 $12
                        ADDI    $13 1
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -1
loadMetAdd_114_i:       EXCH    $13 $12
                        XOR     $12 $11
                        EXCH    $11 $9
                        EXCH    $11 $6
                        EXCH    $12 $8
copy_118:               XOR     $11 $12
                        ADDI    $12 1
                        EXCH    $13 $12
                        ADDI    $13 1
                        EXCH    $13 $12
                        ADDI    $12 -1
                        EXCH    $12 $8
                        EXCH    $11 $6
                        EXCH    $11 $9
                        XOR     $12 $11
loadMetAdd_119:         EXCH    $13 $12
                        ADDI    $13 0
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 0
                        EXCH    $13 $12
                        EXCH    $11 $9
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $12 $1
                        ADDI    $1 -1
                        ADDI    $15 -3238
l_rjmp_top_121:         RBRA    l_rjmp_bot_122
l_jmp_120:              SWAPBR  $15
                        NEG     $15
l_rjmp_bot_122:         BRA     l_rjmp_top_121
                        ADDI    $15 3238
                        ADDI    $1 1
                        EXCH    $12 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $11 $9
                        EXCH    $13 $12
```

```
                        ADDI   $13 0
                        EXCH   $14 $13
                        XOR    $15 $14
                        EXCH   $14 $13
                        ADDI   $13 0
loadMetAdd_119_i:       EXCH   $13 $12
                        XOR    $12 $11
                        EXCH   $11 $9
                        EXCH   $11 $8
                        XOR    $12 $11
loadMetAdd_123:         EXCH   $13 $12
                        ADDI   $13 1
                        EXCH   $14 $13
                        XOR    $15 $14
                        EXCH   $14 $13
                        ADDI   $13 -1
                        EXCH   $13 $12
                        EXCH   $11 $8
                        EXCH   $3 $1
                        ADDI   $1 -1
                        EXCH   $10 $1
                        ADDI   $1 -1
                        EXCH   $8 $1
                        ADDI   $1 -1
                        EXCH   $7 $1
                        ADDI   $1 -1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $9 $1
                        ADDI   $1 -1
                        EXCH   $12 $1
                        ADDI   $1 -1
                        ADDI   $15 -3292
l_rjmp_top_125:         RBRA   l_rjmp_bot_126
l_jmp_124:              SWAPBR $15
                        NEG    $15
l_rjmp_bot_126:         BRA    l_rjmp_top_125
                        ADDI   $15 3292
                        ADDI   $1 1
                        EXCH   $12 $1
                        ADDI   $1 1
                        EXCH   $9 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        ADDI   $1 1
                        EXCH   $7 $1
                        ADDI   $1 1
                        EXCH   $8 $1
                        ADDI   $1 1
                        EXCH   $10 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        EXCH   $11 $8
                        EXCH   $13 $12
                        ADDI   $13 1
                        EXCH   $14 $13
                        XOR    $15 $14
                        EXCH   $14 $13
                        ADDI   $13 -1
loadMetAdd_123_i:       EXCH   $13 $12
                        XOR    $12 $11
                        EXCH   $11 $8
                        EXCH   $11 $6
                        EXCH   $12 $7
copy_127:               XOR    $11 $12
                        ADDI   $12 1
                        EXCH   $13 $12
                        ADDI   $13 1
                        EXCH   $13 $12
                        ADDI   $12 -1
                        EXCH   $12 $7
                        EXCH   $11 $6
                        EXCH   $11 $8
                        XOR    $12 $11
loadMetAdd_128:         EXCH   $13 $12
                        ADDI   $13 0
                        EXCH   $14 $13
                        XOR    $15 $14
                        EXCH   $14 $13
                        ADDI   $13 0
                        EXCH   $13 $12
                        EXCH   $11 $8
                        EXCH   $3 $1
                        ADDI   $1 -1
                        EXCH   $10 $1
                        ADDI   $1 -1
                        EXCH   $9 $1
                        ADDI   $1 -1
                        EXCH   $8 $1
                        ADDI   $1 -1
                        EXCH   $7 $1
                        ADDI   $1 -1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $12 $1
                        ADDI   $1 -1
                        ADDI   $15 -3356
l_rjmp_top_130:         RBRA   l_rjmp_bot_131
l_jmp_129:              SWAPBR $15
                        NEG    $15
l_rjmp_bot_131:         BRA    l_rjmp_top_130
                        ADDI   $15 3356
                        ADDI   $1 1
                        EXCH   $12 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        ADDI   $1 1
                        EXCH   $7 $1
                        ADDI   $1 1
                        EXCH   $8 $1
                        ADDI   $1 1
                        EXCH   $9 $1
                        ADDI   $1 1
                        EXCH   $10 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        EXCH   $11 $8
                        EXCH   $13 $12
                        ADDI   $13 0
                        EXCH   $14 $13
                        XOR    $15 $14
                        EXCH   $14 $13
                        ADDI   $13 0
loadMetAdd_128_i:       EXCH   $13 $12
                        XOR    $12 $11
                        EXCH   $11 $8
                        EXCH   $11 $7
                        XOR    $12 $11
loadMetAdd_132:         EXCH   $13 $12
                        ADDI   $13 1
                        EXCH   $14 $13
                        XOR    $15 $14
```

```
                        EXCH    $14 $13
                        ADDI    $13 -1
                        EXCH    $13 $12
                        EXCH    $11 $7
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $12 $1
                        ADDI    $1 -1
                        ADDI    $15 -3410
l_rjmp_top_134:         RBRA    l_rjmp_bot_135
l_jmp_133:              SWAPBR  $15
                        NEG     $15
l_rjmp_bot_135:         BRA     l_rjmp_top_134
                        ADDI    $15 3410
                        ADDI    $1 1
                        EXCH    $12 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $11 $7
                        EXCH    $13 $12
                        ADDI    $13 1
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 -1
loadMetAdd_132_i:       EXCH    $13 $12
                        XOR     $12 $11
                        EXCH    $11 $7
                        EXCH    $11 $6
                        ADD     $12 $3
                        ADDI    $12 2
                        EXCH    $13 $12
                        ADDI    $12 -2
                        SUB     $12 $3
copy_136:               XOR     $11 $13
                        ADDI    $13 1
                        EXCH    $14 $13
                        ADDI    $14 1
                        EXCH    $14 $13
                        ADDI    $13 -1
                        ADD     $12 $3
                        ADDI    $12 2
                        EXCH    $13 $12
                        ADDI    $12 -2
                        SUB     $12 $3
                        EXCH    $11 $6
                        EXCH    $11 $7
                        XOR     $12 $11
loadMetAdd_137:         EXCH    $13 $12
                        ADDI    $13 0
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 0
                        EXCH    $13 $12
                        EXCH    $11 $7
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $12 $1
                        ADDI    $1 -1
                        ADDI    $15 -3482
l_rjmp_top_139:         RBRA    l_rjmp_bot_140
l_jmp_138:              SWAPBR  $15
                        NEG     $15
l_rjmp_bot_140:         BRA     l_rjmp_top_139
                        ADDI    $15 3482
                        ADDI    $1 1
                        EXCH    $12 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $11 $7
                        EXCH    $13 $12
                        ADDI    $13 0
                        EXCH    $14 $13
                        XOR     $15 $14
                        EXCH    $14 $13
                        ADDI    $13 0
loadMetAdd_137_i:       EXCH    $13 $12
                        XOR     $12 $11
                        EXCH    $11 $7
                        ADD     $11 $3
                        ADDI    $11 2
                        EXCH    $12 $11
                        ADDI    $11 -2
                        SUB     $11 $3
                        XOR     $13 $12
loadMetAdd_141:         EXCH    $14 $13
                        ADDI    $14 1
                        EXCH    $15 $14
                        XOR     $16 $15
                        EXCH    $15 $14
                        ADDI    $14 -1
```

```
                        EXCH    $14 $13
                        ADD     $11 $3
                        ADDI    $11 2
                        EXCH    $12 $11
                        ADDI    $11 -2
                        SUB     $11 $3
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        EXCH    $8 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $13 $1
                        ADDI    $1 -1
                        ADDI    $16 -3544
l_rjmp_top_143:         RBRA    l_rjmp_bot_144
l_jmp_142:              SWAPBR  $16
                        NEG     $16
l_rjmp_bot_144:         BRA     l_rjmp_top_143
                        ADDI    $16 3544
                        ADDI    $1 1
                        EXCH    $13 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADD     $11 $3
                        ADDI    $11 2
                        EXCH    $12 $11
                        ADDI    $11 -2
                        SUB     $11 $3
                        EXCH    $14 $13
                        ADDI    $14 1
                        EXCH    $15 $14
                        XOR     $16 $15
                        EXCH    $15 $14
                        ADDI    $14 -1
loadMetAdd_141_i:       EXCH    $14 $13
                        XOR     $13 $12
                        ADD     $11 $3
                        ADDI    $11 2
                        EXCH    $12 $11
                        ADDI    $11 -2
                        SUB     $11 $3
                        ADDI    $1 1
                        EXCH    $11 $1
                        XOR     $11 $0
localBlock_145_i:       XOR     $10 $1
                        ADDI    $1 1
                        EXCH    $10 $1
                        XOR     $10 $0
localBlock_146_i:       XOR     $9 $1
                        ADDI    $1 1
                        EXCH    $9 $1
                        XOR     $9 $0
localBlock_147_i:       XOR     $8 $1
                        ADDI    $1 1
                        EXCH    $8 $1
                        XOR     $8 $0
localBlock_148_i:       XOR     $7 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        XOR     $7 $0
localBlock_149_i:       XOR     $6 $1
l_initTape_3_bot:       BRA
                            l_initTape_3_top
l_init_4_top:           BRA     l_init_4_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $3 $1
                        ADDI    $1 -1
l_init_4:               SWAPBR  $2
                        NEG     $2
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
                        EXCH    $3 $1
                        ADDI    $1 -1
                        BRA
                            l_initLiterals_1
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $3 $1
                        ADDI    $1 -1
                        BRA     l_initRules_2
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $3 $1
                        ADDI    $1 -1
                        BRA     l_initTape_3
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADD     $6 $3
                        ADDI    $6 11
                        EXCH    $7 $6
                        ADDI    $6 -11
                        SUB     $6 $3
                        XORI    $8 1
                        ADD     $7 $8
                        XORI    $8 1
                        ADD     $6 $3
                        ADDI    $6 11
                        EXCH    $7 $6
                        ADDI    $6 -11
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 12
                        EXCH    $7 $6
                        ADDI    $6 -12
                        SUB     $6 $3
                        XORI    $8 1
                        ADD     $7 $8
                        XORI    $8 1
                        ADD     $6 $3
                        ADDI    $6 12
                        EXCH    $7 $6
                        ADDI    $6 -12
```

```
3654                            SUB     $6 $3
3655                            ADD     $6 $3
3656                            ADDI    $6 13
3657                            EXCH    $7 $6
3658                            ADDI    $6 -13
3659                            SUB     $6 $3
3660                            XORI    $8 6
3661                            ADD     $7 $8
3662                            XORI    $8 6
3663                            ADD     $6 $3
3664                            ADDI    $6 13
3665                            EXCH    $7 $6
3666                            ADDI    $6 -13
3667                            SUB     $6 $3
3668                            EXCH    $3 $1
3669                            ADDI    $1 -1
3670                            BRA     l_simulate_5
3671                            ADDI    $1 1
3672                            EXCH    $3 $1
3673 l_init_4_bot:              BRA     l_init_4_top
3674 l_simulate_5_top:          BRA
        l_simulate_5_bot
3675                            ADDI    $1 1
3676                            EXCH    $2 $1
3677                            EXCH    $3 $1
3678                            ADDI    $1 -1
3679 l_simulate_5:              SWAPBR  $2
3680                            NEG     $2
3681                            ADDI    $1 1
3682                            EXCH    $3 $1
3683                            EXCH    $2 $1
3684                            ADDI    $1 -1
3685                            XORI    $6 1
3686 entry_150:                 BEQ     $6 $0
        assert_152
3687                            ADD     $7 $3
3688                            ADDI    $7 11
3689                            EXCH    $8 $7
3690                            ADDI    $7 -11
3691                            SUB     $7 $3
3692                            ADD     $9 $3
3693                            ADDI    $9 12
3694                            EXCH    $10 $9
3695                            ADDI    $9 -12
3696                            SUB     $9 $3
3697 cmp_top_154:               BNE     $8 $10
        cmp_bot_155
3698                            XORI    $11 1
3699 cmp_bot_155:               BNE     $8 $10
        cmp_top_154
3700 f_top_156:                 BEQ     $11 $0
        f_bot_157
3701                            XORI    $12 1
3702 f_bot_157:                 BEQ     $11 $0
        f_top_156
3703                            XOR     $6 $12
3704 f_bot_157_i:               BEQ     $11 $0
        f_top_156_i
3705                            XORI    $12 1
3706 f_top_156_i:               BEQ     $11 $0
        f_bot_157_i
3707 cmp_bot_155_i:             BNE     $8 $10
        cmp_top_154_i
3708                            XORI    $11 1
3709 cmp_top_154_i:             BNE     $8 $10
        cmp_bot_155_i
3710                            ADD     $9 $3
3711                            ADDI    $9 12
3712                            EXCH    $10 $9
3713                            ADDI    $9 -12
3714                            SUB     $9 $3
3715                            ADD     $7 $3
3716                            ADDI    $7 11
3717                            EXCH    $8 $7
3718                            ADDI    $7 -11
3719                            SUB     $7 $3
3720                            ADD     $7 $3
3721                            ADDI    $7 2
3722                            EXCH    $8 $7
3723                            ADDI    $7 -2
3724                            SUB     $7 $3
3725                            XOR     $9 $8
3726 loadMetAdd_158:            EXCH    $10 $9
3727                            ADDI    $10 3
3728                            EXCH    $11 $10
3729                            XOR     $12 $11
3730                            EXCH    $11 $10
3731                            ADDI    $10 -3
3732                            EXCH    $10 $9
3733                            ADD     $7 $3
3734                            ADDI    $7 2
3735                            EXCH    $8 $7
3736                            ADDI    $7 -2
3737                            SUB     $7 $3
3738                            ADD     $13 $3
3739                            ADDI    $13 14
3740                            EXCH    $3 $1
3741                            ADDI    $1 -1
3742                            EXCH    $13 $1
3743                            ADDI    $1 -1
3744                            EXCH    $9 $1
3745                            ADDI    $1 -1
3746                            ADDI    $12 -3745
3747 l_jmp_159:                 SWAPBR  $12
3748                            NEG     $12
3749                            ADDI    $12 3745
3750                            ADDI    $1 1
3751                            EXCH    $9 $1
3752                            ADDI    $1 1
3753                            EXCH    $13 $1
3754                            ADDI    $1 1
3755                            EXCH    $3 $1
3756                            ADDI    $13 -14
3757                            SUB     $13 $3
3758                            ADD     $7 $3
3759                            ADDI    $7 2
3760                            EXCH    $8 $7
3761                            ADDI    $7 -2
3762                            SUB     $7 $3
3763                            EXCH    $10 $9
3764                            ADDI    $10 3
3765                            EXCH    $11 $10
3766                            XOR     $12 $11
3767                            EXCH    $11 $10
3768                            ADDI    $10 -3
3769 loadMetAdd_158_i:          EXCH    $10 $9
3770                            XOR     $9 $8
3771                            ADD     $7 $3
3772                            ADDI    $7 2
3773                            EXCH    $8 $7
3774                            ADDI    $7 -2
3775                            SUB     $7 $3
```

```
                        EXCH    $3 $1
                        ADDI    $1 -1
                        BRA     l_inst_6
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADD     $7 $3
                        ADDI    $7 2
                        EXCH    $8 $7
                        ADDI    $7 -2
                        SUB     $7 $3
                        XOR     $9 $8
loadMetAdd_160:         EXCH    $10 $9
                        ADDI    $10 3
                        EXCH    $11 $10
                        XOR     $12 $11
                        EXCH    $11 $10
                        ADDI    $10 -3
                        EXCH    $10 $9
                        ADD     $7 $3
                        ADDI    $7 2
                        EXCH    $8 $7
                        ADDI    $7 -2
                        SUB     $7 $3
                        ADD     $13 $3
                        ADDI    $13 14
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $13 $1
                        ADDI    $1 -1
                        EXCH    $9 $1
                        ADDI    $1 -1
                        ADDI    $12 -3807
l_rjmp_top_162:         RBRA    l_rjmp_bot_163
l_jmp_161:              SWAPBR  $12
                        NEG     $12
l_rjmp_bot_163:         BRA     l_rjmp_top_162
                        ADDI    $12 3807
                        ADDI    $1 1
                        EXCH    $9 $1
                        ADDI    $1 1
                        EXCH    $13 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $13 -14
                        SUB     $13 $3
                        ADD     $7 $3
                        ADDI    $7 2
                        EXCH    $8 $7
                        ADDI    $7 -2
                        SUB     $7 $3
                        EXCH    $10 $9
                        ADDI    $10 3
                        EXCH    $11 $10
                        XOR     $12 $11
                        EXCH    $11 $10
                        ADDI    $10 -3
loadMetAdd_160_i:       EXCH    $10 $9
                        XOR     $9 $8
                        ADD     $7 $3
                        ADDI    $7 2
                        EXCH    $8 $7
                        ADDI    $7 -2
                        SUB     $7 $3
                        ADD     $7 $3
                        ADDI    $7 16
                        EXCH    $8 $7
                        ADDI    $7 -16
                        SUB     $7 $3
                        XORI    $9 1
                        ADD     $8 $9
                        XORI    $9 1
                        ADD     $7 $3
                        ADDI    $7 16
                        EXCH    $8 $7
                        ADDI    $7 -16
                        SUB     $7 $3
                        ADD     $8 $3
                        ADDI    $8 16
                        EXCH    $9 $8
                        ADDI    $8 -16
                        SUB     $8 $3
                        ADD     $10 $3
                        ADDI    $10 15
                        EXCH    $11 $10
                        ADDI    $10 -15
                        SUB     $10 $3
cmp_top_168:            BNE     $9 $11
                            cmp_bot_169
                        XORI    $12 1
cmp_bot_169:            BNE     $9 $11
                            cmp_top_168
f_top_170:              BEQ     $12 $0
                            f_bot_171
                        XORI    $13 1
f_bot_171:              BEQ     $12 $0
                            f_top_170
                        XOR     $7 $13
f_bot_171_i:            BEQ     $12 $0
                            f_top_170_i
                        XORI    $13 1
f_top_170_i:            BEQ     $12 $0
                            f_bot_171_i
cmp_bot_169_i:          BNE     $9 $11
                            cmp_top_168_i
                        XORI    $12 1
cmp_top_168_i:          BNE     $9 $11
                            cmp_bot_169_i
                        ADD     $10 $3
                        ADDI    $10 15
                        EXCH    $11 $10
                        ADDI    $10 -15
                        SUB     $10 $3
                        ADD     $8 $3
                        ADDI    $8 16
                        EXCH    $9 $8
                        ADDI    $8 -16
                        SUB     $8 $3
test_164:               BEQ     $7 $0
                            test_false_166
                        XORI    $7 1
                        ADD     $8 $3
                        ADDI    $8 16
                        EXCH    $9 $8
                        ADDI    $8 -16
                        SUB     $8 $3
                        ADD     $10 $3
                        ADDI    $10 15
                        EXCH    $11 $10
                        ADDI    $10 -15
                        SUB     $10 $3
                        XOR     $9 $11
                        ADD     $10 $3
```

```
                        ADDI    $10 15
                        EXCH    $11 $10
                        ADDI    $10 -15
                        SUB     $10 $3
                        ADD     $8 $3
                        ADDI    $8 16
                        EXCH    $9 $8
                        ADDI    $8 -16
                        SUB     $8 $3
                        XORI    $7 1
assert_true_165:        BRA     assert_167
test_false_166:         BRA     test_164
assert_167:             BNE     $7 $0
    assert_true_165
                        ADD     $8 $3
                        ADDI    $8 16
                        EXCH    $9 $8
                        ADDI    $8 -16
                        SUB     $8 $3
cmp_top_172:            BNE     $9 $0
    cmp_bot_173
                        XORI    $10 1
cmp_bot_173:            BNE     $9 $0
    cmp_top_172
f_top_174:              BEQ     $10 $0
    f_bot_175
                        XORI    $11 1
f_bot_175:              BEQ     $10 $0
    f_top_174
                        XOR     $7 $11
f_bot_175_i:            BEQ     $10 $0
    f_top_174_i
                        XORI    $11 1
f_top_174_i:            BEQ     $10 $0
    f_bot_175_i
cmp_bot_173_i:          BNE     $9 $0
    cmp_top_172_i
                        XORI    $10 1
cmp_top_172_i:          BNE     $9 $0
    cmp_bot_173_i
                        ADD     $8 $3
                        ADDI    $8 16
                        EXCH    $9 $8
                        ADDI    $8 -16
                        SUB     $8 $3
                        ADD     $7 $3
                        ADDI    $7 11
                        EXCH    $8 $7
                        ADDI    $7 -11
                        SUB     $7 $3
                        ADD     $9 $3
                        ADDI    $9 13
                        EXCH    $10 $9
                        ADDI    $9 -13
                        SUB     $9 $3
cmp_top_176:            BNE     $8 $10
    cmp_bot_177
                        XORI    $11 1
cmp_bot_177:            BNE     $8 $10
    cmp_top_176
f_top_178:              BEQ     $11 $0
    f_bot_179
                        XORI    $12 1
f_bot_179:              BEQ     $11 $0
    f_top_178
                        XOR     $6 $12
f_bot_179_i:            BEQ     $11 $0
    f_top_178_i
                        XORI    $12 1
f_top_178_i:            BEQ     $11 $0
    f_bot_179_i
cmp_bot_177_i:          BNE     $8 $10
    cmp_top_176_i
                        XORI    $11 1
cmp_top_176_i:          BNE     $8 $10
    cmp_bot_177_i
                        ADD     $9 $3
                        ADDI    $9 13
                        EXCH    $10 $9
                        ADDI    $9 -13
                        SUB     $9 $3
                        ADD     $7 $3
                        ADDI    $7 11
                        EXCH    $8 $7
                        ADDI    $7 -11
                        SUB     $7 $3
test_151:               BNE     $6 $0 exit_153
assert_152:             BRA     entry_150
exit_153:               BRA     test_151
                        XORI    $6 1
l_simulate_5_bot:       BRA
    l_simulate_5_top
l_inst_6_top:           BRA     l_inst_6_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $3 $1
                        ADDI    $1 -1
l_inst_6:               SWAPBR  $2
                        NEG     $2
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
                        ADD     $7 $3
                        ADDI    $7 11
                        EXCH    $8 $7
                        ADDI    $7 -11
                        SUB     $7 $3
                        ADD     $9 $3
                        ADDI    $9 3
                        ADD     $10 $3
                        ADDI    $10 16
                        EXCH    $11 $10
                        ADDI    $10 -16
                        SUB     $10 $3
                        EXCH    $13 $9
                        XOR     $12 $13
                        EXCH    $13 $9
                        ADDI    $12 2
                        ADD     $12 $11
                        ADD     $10 $3
                        ADDI    $10 16
                        EXCH    $11 $10
                        ADDI    $10 -16
                        SUB     $10 $3
                        ADDI    $9 -3
                        SUB     $9 $3
                        EXCH    $14 $12
                        ADD     $9 $3
                        ADDI    $9 3
                        ADD     $10 $3
                        ADDI    $10 16
```

```
4013                        EXCH   $11 $10
4014                        ADDI   $10 -16
4015                        SUB    $10 $3
4016                        SUB    $12 $11
4017                        ADDI   $12 -2
4018                        EXCH   $13 $9
4019                        XOR    $12 $13
4020                        EXCH   $13 $9
4021                        ADD    $10 $3
4022                        ADDI   $10 16
4023                        EXCH   $11 $10
4024                        ADDI   $10 -16
4025                        SUB    $10 $3
4026                        ADDI   $9 -3
4027                        SUB    $9 $3
4028 cmp_top_184:           BNE    $8 $14
         cmp_bot_185
4029                        XORI   $15 1
4030 cmp_bot_185:           BNE    $8 $14
         cmp_top_184
4031                        ADD    $16 $3
4032                        ADDI   $16 14
4033                        EXCH   $17 $16
4034                        ADDI   $16 -14
4035                        SUB    $16 $3
4036                        ADD    $18 $3
4037                        ADDI   $18 5
4038                        ADD    $19 $3
4039                        ADDI   $19 16
4040                        EXCH   $20 $19
4041                        ADDI   $19 -16
4042                        SUB    $19 $3
4043                        EXCH   $22 $18
4044                        XOR    $21 $22
4045                        EXCH   $22 $18
4046                        ADDI   $21 2
4047                        ADD    $21 $20
4048                        ADD    $19 $3
4049                        ADDI   $19 16
4050                        EXCH   $20 $19
4051                        ADDI   $19 -16
4052                        SUB    $19 $3
4053                        ADDI   $18 -5
4054                        SUB    $18 $3
4055                        EXCH   $23 $21
4056                        ADD    $18 $3
4057                        ADDI   $18 5
4058                        ADD    $19 $3
4059                        ADDI   $19 16
4060                        EXCH   $20 $19
4061                        ADDI   $19 -16
4062                        SUB    $19 $3
4063                        SUB    $21 $20
4064                        ADDI   $21 -2
4065                        EXCH   $22 $18
4066                        XOR    $21 $22
4067                        EXCH   $22 $18
4068                        ADD    $19 $3
4069                        ADDI   $19 16
4070                        EXCH   $20 $19
4071                        ADDI   $19 -16
4072                        SUB    $19 $3
4073                        ADDI   $18 -5
4074                        SUB    $18 $3
4075 cmp_top_186:           BNE    $17 $23
         cmp_bot_187
4076                        XORI   $24 1
4077 cmp_bot_187:           BNE    $17 $23
         cmp_top_186
4078                        ANDX   $25 $15 $24
4079 f_top_188:             BEQ    $25 $0
         f_bot_189
4080                        XORI   $26 1
4081 f_bot_189:             BEQ    $25 $0
         f_top_188
4082                        XOR    $6 $26
4083 f_bot_189_i:           BEQ    $25 $0
         f_top_188_i
4084                        XORI   $26 1
4085 f_top_188_i:           BEQ    $25 $0
         f_bot_189_i
4086                        ANDX   $25 $15 $24
4087 cmp_bot_187_i:         BNE    $17 $23
         cmp_top_186_i
4088                        XORI   $24 1
4089 cmp_top_186_i:         BNE    $17 $23
         cmp_bot_187_i
4090                        ADD    $18 $3
4091                        ADDI   $18 5
4092                        ADD    $19 $3
4093                        ADDI   $19 16
4094                        EXCH   $20 $19
4095                        ADDI   $19 -16
4096                        SUB    $19 $3
4097                        EXCH   $22 $18
4098                        XOR    $21 $22
4099                        EXCH   $22 $18
4100                        ADDI   $21 2
4101                        ADD    $21 $20
4102                        ADD    $19 $3
4103                        ADDI   $19 16
4104                        EXCH   $20 $19
4105                        ADDI   $19 -16
4106                        SUB    $19 $3
4107                        ADDI   $18 -5
4108                        SUB    $18 $3
4109                        EXCH   $23 $21
4110                        ADD    $18 $3
4111                        ADDI   $18 5
4112                        ADD    $19 $3
4113                        ADDI   $19 16
4114                        EXCH   $20 $19
4115                        ADDI   $19 -16
4116                        SUB    $19 $3
4117                        SUB    $21 $20
4118                        ADDI   $21 -2
4119                        EXCH   $22 $18
4120                        XOR    $21 $22
4121                        EXCH   $22 $18
4122                        ADD    $19 $3
4123                        ADDI   $19 16
4124                        EXCH   $20 $19
4125                        ADDI   $19 -16
4126                        SUB    $19 $3
4127                        ADDI   $18 -5
4128                        SUB    $18 $3
4129                        ADD    $16 $3
4130                        ADDI   $16 14
4131                        EXCH   $17 $16
4132                        ADDI   $16 -14
4133                        SUB    $16 $3
4134 cmp_bot_185_i:         BNE    $8 $14
```

```
    cmp_top_184_i
                        XORI    $15 1
cmp_top_184_i:          BNE     $8 $14
    cmp_bot_185_i
                        ADD     $9 $3
                        ADDI    $9 3
                        ADD     $10 $3
                        ADDI    $10 16
                        EXCH    $11 $10
                        ADDI    $10 -16
                        SUB     $10 $3
                        EXCH    $13 $9
                        XOR     $12 $13
                        EXCH    $13 $9
                        ADDI    $12 2
                        ADD     $12 $11
                        ADD     $10 $3
                        ADDI    $10 16
                        EXCH    $11 $10
                        ADDI    $10 -16
                        SUB     $10 $3
                        ADDI    $9 -3
                        SUB     $9 $3
                        EXCH    $14 $12
                        ADD     $9 $3
                        ADDI    $9 3
                        ADD     $10 $3
                        ADDI    $10 16
                        EXCH    $11 $10
                        ADDI    $10 -16
                        SUB     $10 $3
                        SUB     $12 $11
                        ADDI    $12 -2
                        EXCH    $13 $9
                        XOR     $12 $13
                        EXCH    $13 $9
                        ADD     $10 $3
                        ADDI    $10 16
                        EXCH    $11 $10
                        ADDI    $10 -16
                        SUB     $10 $3
                        ADDI    $9 -3
                        SUB     $9 $3
                        ADD     $7 $3
                        ADDI    $7 11
                        EXCH    $8 $7
                        ADDI    $7 -11
                        SUB     $7 $3
test_180:               BEQ     $6 $0
    test_false_182
                        XORI    $6 1
                        ADD     $7 $3
                        ADDI    $7 11
                        EXCH    $8 $7
                        ADDI    $7 -11
                        SUB     $7 $3
                        ADD     $9 $3
                        ADDI    $9 4
                        ADD     $10 $3
                        ADDI    $10 16
                        EXCH    $11 $10
                        ADDI    $10 -16
                        SUB     $10 $3
                        EXCH    $13 $9
                        XOR     $12 $13
                        EXCH    $13 $9
                        ADDI    $12 2
                        ADD     $12 $11
                        ADD     $10 $3
                        ADDI    $10 16
                        EXCH    $11 $10
                        ADDI    $10 -16
                        SUB     $10 $3
                        ADDI    $9 -4
                        SUB     $9 $3
                        EXCH    $14 $12
                        ADD     $9 $3
                        ADDI    $9 4
                        ADD     $10 $3
                        ADDI    $10 16
                        EXCH    $11 $10
                        ADDI    $10 -16
                        SUB     $10 $3
                        SUB     $12 $11
                        ADDI    $12 -2
                        EXCH    $13 $9
                        XOR     $12 $13
                        EXCH    $13 $9
                        ADD     $10 $3
                        ADDI    $10 16
                        EXCH    $11 $10
                        ADDI    $10 -16
                        SUB     $10 $3
                        ADDI    $9 -4
                        SUB     $9 $3
                        ADD     $15 $3
                        ADDI    $15 3
                        ADD     $16 $3
                        ADDI    $16 16
                        EXCH    $17 $16
                        ADDI    $16 -16
                        SUB     $16 $3
                        EXCH    $19 $15
                        XOR     $18 $19
                        EXCH    $19 $15
                        ADDI    $18 2
                        ADD     $18 $17
                        ADD     $16 $3
                        ADDI    $16 16
                        EXCH    $17 $16
                        ADDI    $16 -16
                        SUB     $16 $3
                        ADDI    $15 -3
                        SUB     $15 $3
                        EXCH    $20 $18
                        ADD     $15 $3
                        ADDI    $15 3
                        ADD     $16 $3
                        ADDI    $16 16
                        EXCH    $17 $16
                        ADDI    $16 -16
                        SUB     $16 $3
                        SUB     $18 $17
                        ADDI    $18 -2
                        EXCH    $19 $15
                        XOR     $18 $19
                        EXCH    $19 $15
                        ADD     $16 $3
                        ADDI    $16 16
                        EXCH    $17 $16
                        ADDI    $16 -16
                        SUB     $16 $3
```

```
4264                ADDI    $15 -3
4265                SUB     $15 $3
4266                XOR     $21 $14
4267                SUB     $21 $20
4268                ADD     $8 $21
4269                ADD     $21 $20
4270                XOR     $21 $14
4271                ADD     $15 $3
4272                ADDI    $15 3
4273                ADD     $16 $3
4274                ADDI    $16 16
4275                EXCH    $17 $16
4276                ADDI    $16 -16
4277                SUB     $16 $3
4278                EXCH    $19 $15
4279                XOR     $18 $19
4280                EXCH    $19 $15
4281                ADDI    $18 2
4282                ADD     $18 $17
4283                ADD     $16 $3
4284                ADDI    $16 16
4285                EXCH    $17 $16
4286                ADDI    $16 -16
4287                SUB     $16 $3
4288                ADDI    $15 -3
4289                SUB     $15 $3
4290                EXCH    $20 $18
4291                ADD     $15 $3
4292                ADDI    $15 3
4293                ADD     $16 $3
4294                ADDI    $16 16
4295                EXCH    $17 $16
4296                ADDI    $16 -16
4297                SUB     $16 $3
4298                SUB     $18 $17
4299                ADDI    $18 -2
4300                EXCH    $19 $15
4301                XOR     $18 $19
4302                EXCH    $19 $15
4303                ADD     $16 $3
4304                ADDI    $16 16
4305                EXCH    $17 $16
4306                ADDI    $16 -16
4307                SUB     $16 $3
4308                ADDI    $15 -3
4309                SUB     $15 $3
4310                ADD     $9 $3
4311                ADDI    $9 4
4312                ADD     $10 $3
4313                ADDI    $10 16
4314                EXCH    $11 $10
4315                ADDI    $10 -16
4316                SUB     $10 $3
4317                EXCH    $13 $9
4318                XOR     $12 $13
4319                EXCH    $13 $9
4320                ADDI    $12 2
4321                ADD     $12 $11
4322                ADD     $10 $3
4323                ADDI    $10 16
4324                EXCH    $11 $10
4325                ADDI    $10 -16
4326                SUB     $10 $3
4327                ADDI    $9 -4
4328                SUB     $9 $3
4329                EXCH    $14 $12
4330                ADD     $9 $3
4331                ADDI    $9 4
4332                ADD     $10 $3
4333                ADDI    $10 16
4334                EXCH    $11 $10
4335                ADDI    $10 -16
4336                SUB     $10 $3
4337                SUB     $12 $11
4338                ADDI    $12 -2
4339                EXCH    $13 $9
4340                XOR     $12 $13
4341                EXCH    $13 $9
4342                ADD     $10 $3
4343                ADDI    $10 16
4344                EXCH    $11 $10
4345                ADDI    $10 -16
4346                SUB     $10 $3
4347                ADDI    $9 -4
4348                SUB     $9 $3
4349                ADD     $7 $3
4350                ADDI    $7 11
4351                EXCH    $8 $7
4352                ADDI    $7 -11
4353                SUB     $7 $3
4354                ADD     $7 $3
4355                ADDI    $7 14
4356                EXCH    $8 $7
4357                ADDI    $7 -14
4358                SUB     $7 $3
4359                ADD     $9 $3
4360                ADDI    $9 6
4361                ADD     $10 $3
4362                ADDI    $10 16
4363                EXCH    $11 $10
4364                ADDI    $10 -16
4365                SUB     $10 $3
4366                EXCH    $13 $9
4367                XOR     $12 $13
4368                EXCH    $13 $9
4369                ADDI    $12 2
4370                ADD     $12 $11
4371                ADD     $10 $3
4372                ADDI    $10 16
4373                EXCH    $11 $10
4374                ADDI    $10 -16
4375                SUB     $10 $3
4376                ADDI    $9 -6
4377                SUB     $9 $3
4378                EXCH    $14 $12
4379                ADD     $9 $3
4380                ADDI    $9 6
4381                ADD     $10 $3
4382                ADDI    $10 16
4383                EXCH    $11 $10
4384                ADDI    $10 -16
4385                SUB     $10 $3
4386                SUB     $12 $11
4387                ADDI    $12 -2
4388                EXCH    $13 $9
4389                XOR     $12 $13
4390                EXCH    $13 $9
4391                ADD     $10 $3
4392                ADDI    $10 16
4393                EXCH    $11 $10
4394                ADDI    $10 -16
4395                SUB     $10 $3
```

```
                    ADDI    $9 -6
                    SUB     $9 $3
                    ADD     $15 $3
                    ADDI    $15 5
                    ADD     $16 $3
                    ADDI    $16 16
                    EXCH    $17 $16
                    ADDI    $16 -16
                    SUB     $16 $3
                    EXCH    $19 $15
                    XOR     $18 $19
                    EXCH    $19 $15
                    ADDI    $18 2
                    ADD     $18 $17
                    ADD     $16 $3
                    ADDI    $16 16
                    EXCH    $17 $16
                    ADDI    $16 -16
                    SUB     $16 $3
                    ADDI    $15 -5
                    SUB     $15 $3
                    EXCH    $20 $18
                    ADD     $15 $3
                    ADDI    $15 5
                    ADD     $16 $3
                    ADDI    $16 16
                    EXCH    $17 $16
                    ADDI    $16 -16
                    SUB     $16 $3
                    SUB     $18 $17
                    ADDI    $18 -2
                    EXCH    $19 $15
                    XOR     $18 $19
                    EXCH    $19 $15
                    ADD     $16 $3
                    ADDI    $16 16
                    EXCH    $17 $16
                    ADDI    $16 -16
                    SUB     $16 $3
                    ADDI    $15 -5
                    SUB     $15 $3
                    XOR     $21 $14
                    SUB     $21 $20
                    ADD     $8 $21
                    ADD     $21 $20
                    XOR     $21 $14
                    ADD     $15 $3
                    ADDI    $15 5
                    ADD     $16 $3
                    ADDI    $16 16
                    EXCH    $17 $16
                    ADDI    $16 -16
                    SUB     $16 $3
                    EXCH    $19 $15
                    XOR     $18 $19
                    EXCH    $19 $15
                    ADDI    $18 2
                    ADD     $18 $17
                    ADD     $16 $3
                    ADDI    $16 16
                    EXCH    $17 $16
                    ADDI    $16 -16
                    SUB     $16 $3
                    ADDI    $15 -5
                    SUB     $15 $3
                    EXCH    $20 $18
                    ADD     $15 $3
                    ADDI    $15 5
                    ADD     $16 $3
                    ADDI    $16 16
                    EXCH    $17 $16
                    ADDI    $16 -16
                    SUB     $16 $3
                    SUB     $18 $17
                    ADDI    $18 -2
                    EXCH    $19 $15
                    XOR     $18 $19
                    EXCH    $19 $15
                    ADD     $16 $3
                    ADDI    $16 16
                    EXCH    $17 $16
                    ADDI    $16 -16
                    SUB     $16 $3
                    ADDI    $15 -5
                    SUB     $15 $3
                    ADD     $9 $3
                    ADDI    $9 6
                    ADD     $10 $3
                    ADDI    $10 16
                    EXCH    $11 $10
                    ADDI    $10 -16
                    SUB     $10 $3
                    EXCH    $13 $9
                    XOR     $12 $13
                    EXCH    $13 $9
                    ADDI    $12 2
                    ADD     $12 $11
                    ADD     $10 $3
                    ADDI    $10 16
                    EXCH    $11 $10
                    ADDI    $10 -16
                    SUB     $10 $3
                    ADDI    $9 -6
                    SUB     $9 $3
                    EXCH    $14 $12
                    ADD     $9 $3
                    ADDI    $9 6
                    ADD     $10 $3
                    ADDI    $10 16
                    EXCH    $11 $10
                    ADDI    $10 -16
                    SUB     $10 $3
                    SUB     $12 $11
                    ADDI    $12 -2
                    EXCH    $13 $9
                    XOR     $12 $13
                    EXCH    $13 $9
                    ADD     $10 $3
                    ADDI    $10 16
                    EXCH    $11 $10
                    ADDI    $10 -16
                    SUB     $10 $3
                    ADDI    $9 -6
                    SUB     $9 $3
                    ADD     $7 $3
                    ADDI    $7 14
                    EXCH    $8 $7
                    ADDI    $7 -14
                    SUB     $7 $3
                    XORI    $6 1
assert_true_181:    BRA     assert_183
test_false_182:     BRA     test_180
```

```
4528 assert_183:              BNE    $6 $0
         assert_true_181
4529                          ADD    $7 $3
4530                          ADDI   $7 11
4531                          EXCH   $8 $7
4532                          ADDI   $7 -11
4533                          SUB    $7 $3
4534                          ADD    $9 $3
4535                          ADDI   $9 4
4536                          ADD    $10 $3
4537                          ADDI   $10 16
4538                          EXCH   $11 $10
4539                          ADDI   $10 -16
4540                          SUB    $10 $3
4541                          EXCH   $13 $9
4542                          XOR    $12 $13
4543                          EXCH   $13 $9
4544                          ADDI   $12 2
4545                          ADD    $12 $11
4546                          ADD    $10 $3
4547                          ADDI   $10 16
4548                          EXCH   $11 $10
4549                          ADDI   $10 -16
4550                          SUB    $10 $3
4551                          ADDI   $9 -4
4552                          SUB    $9 $3
4553                          EXCH   $14 $12
4554                          ADD    $9 $3
4555                          ADDI   $9 4
4556                          ADD    $10 $3
4557                          ADDI   $10 16
4558                          EXCH   $11 $10
4559                          ADDI   $10 -16
4560                          SUB    $10 $3
4561                          SUB    $12 $11
4562                          ADDI   $12 -2
4563                          EXCH   $13 $9
4564                          XOR    $12 $13
4565                          EXCH   $13 $9
4566                          ADD    $10 $3
4567                          ADDI   $10 16
4568                          EXCH   $11 $10
4569                          ADDI   $10 -16
4570                          SUB    $10 $3
4571                          ADDI   $9 -4
4572                          SUB    $9 $3
4573 cmp_top_190:             BNE    $8 $14
         cmp_bot_191
4574                          XORI   $15 1
4575 cmp_bot_191:             BNE    $8 $14
         cmp_top_190
4576                          ADD    $16 $3
4577                          ADDI   $16 14
4578                          EXCH   $17 $16
4579                          ADDI   $16 -14
4580                          SUB    $16 $3
4581                          ADD    $18 $3
4582                          ADDI   $18 6
4583                          ADD    $19 $3
4584                          ADDI   $19 16
4585                          EXCH   $20 $19
4586                          ADDI   $19 -16
4587                          SUB    $19 $3
4588                          EXCH   $22 $18
4589                          XOR    $21 $22
4590                          EXCH   $22 $18
4591                          ADDI   $21 2
4592                          ADD    $21 $20
4593                          ADD    $19 $3
4594                          ADDI   $19 16
4595                          EXCH   $20 $19
4596                          ADDI   $19 -16
4597                          SUB    $19 $3
4598                          ADDI   $18 -6
4599                          SUB    $18 $3
4600                          EXCH   $23 $21
4601                          ADD    $18 $3
4602                          ADDI   $18 6
4603                          ADD    $19 $3
4604                          ADDI   $19 16
4605                          EXCH   $20 $19
4606                          ADDI   $19 -16
4607                          SUB    $19 $3
4608                          SUB    $21 $20
4609                          ADDI   $21 -2
4610                          EXCH   $22 $18
4611                          XOR    $21 $22
4612                          EXCH   $22 $18
4613                          ADD    $19 $3
4614                          ADDI   $19 16
4615                          EXCH   $20 $19
4616                          ADDI   $19 -16
4617                          SUB    $19 $3
4618                          ADDI   $18 -6
4619                          SUB    $18 $3
4620 cmp_top_192:             BNE    $17 $23
         cmp_bot_193
4621                          XORI   $24 1
4622 cmp_bot_193:             BNE    $17 $23
         cmp_top_192
4623                          ANDX   $25 $15 $24
4624 f_top_194:               BEQ    $25 $0
         f_bot_195
4625                          XORI   $26 1
4626 f_bot_195:               BEQ    $25 $0
         f_top_194
4627                          XOR    $6 $26
4628 f_bot_195_i:             BEQ    $25 $0
         f_top_194_i
4629                          XORI   $26 1
4630 f_top_194_i:             BEQ    $25 $0
         f_bot_195_i
4631                          ANDX   $25 $15 $24
4632 cmp_bot_193_i:           BNE    $17 $23
         cmp_top_192_i
4633                          XORI   $24 1
4634 cmp_top_192_i:           BNE    $17 $23
         cmp_bot_193_i
4635                          ADD    $18 $3
4636                          ADDI   $18 6
4637                          ADD    $19 $3
4638                          ADDI   $19 16
4639                          EXCH   $20 $19
4640                          ADDI   $19 -16
4641                          SUB    $19 $3
4642                          EXCH   $22 $18
4643                          XOR    $21 $22
4644                          EXCH   $22 $18
4645                          ADDI   $21 2
4646                          ADD    $21 $20
4647                          ADD    $19 $3
4648                          ADDI   $19 16
```

```
4649                        EXCH   $20 $19
4650                        ADDI   $19 -16
4651                        SUB    $19 $3
4652                        ADDI   $18 -6
4653                        SUB    $18 $3
4654                        EXCH   $23 $21
4655                        ADD    $18 $3
4656                        ADDI   $18 6
4657                        ADD    $19 $3
4658                        ADDI   $19 16
4659                        EXCH   $20 $19
4660                        ADDI   $19 -16
4661                        SUB    $19 $3
4662                        SUB    $21 $20
4663                        ADDI   $21 -2
4664                        EXCH   $22 $18
4665                        XOR    $21 $22
4666                        EXCH   $22 $18
4667                        ADD    $19 $3
4668                        ADDI   $19 16
4669                        EXCH   $20 $19
4670                        ADDI   $19 -16
4671                        SUB    $19 $3
4672                        ADDI   $18 -6
4673                        SUB    $18 $3
4674                        ADD    $16 $3
4675                        ADDI   $16 14
4676                        EXCH   $17 $16
4677                        ADDI   $16 -14
4678                        SUB    $16 $3
4679 cmp_bot_191_i:        BNE    $8 $14
        cmp_top_190_i
4680                        XORI   $15 1
4681 cmp_top_190_i:        BNE    $8 $14
        cmp_bot_191_i
4682                        ADD    $9 $3
4683                        ADDI   $9 4
4684                        ADD    $10 $3
4685                        ADDI   $10 16
4686                        EXCH   $11 $10
4687                        ADDI   $10 -16
4688                        SUB    $10 $3
4689                        EXCH   $13 $9
4690                        XOR    $12 $13
4691                        EXCH   $13 $9
4692                        ADDI   $12 2
4693                        ADD    $12 $11
4694                        ADD    $10 $3
4695                        ADDI   $10 16
4696                        EXCH   $11 $10
4697                        ADDI   $10 -16
4698                        SUB    $10 $3
4699                        ADDI   $9 -4
4700                        SUB    $9 $3
4701                        EXCH   $14 $12
4702                        ADD    $9 $3
4703                        ADDI   $9 4
4704                        ADD    $10 $3
4705                        ADDI   $10 16
4706                        EXCH   $11 $10
4707                        ADDI   $10 -16
4708                        SUB    $10 $3
4709                        SUB    $12 $11
4710                        ADDI   $12 -2
4711                        EXCH   $13 $9
4712                        XOR    $12 $13
4713                        EXCH   $13 $9
4714                        ADD    $10 $3
4715                        ADDI   $10 16
4716                        EXCH   $11 $10
4717                        ADDI   $10 -16
4718                        SUB    $10 $3
4719                        ADDI   $9 -4
4720                        SUB    $9 $3
4721                        ADD    $7 $3
4722                        ADDI   $7 11
4723                        EXCH   $8 $7
4724                        ADDI   $7 -11
4725                        SUB    $7 $3
4726                        ADD    $7 $3
4727                        ADDI   $7 11
4728                        EXCH   $8 $7
4729                        ADDI   $7 -11
4730                        SUB    $7 $3
4731                        ADD    $9 $3
4732                        ADDI   $9 3
4733                        ADD    $10 $3
4734                        ADDI   $10 16
4735                        EXCH   $11 $10
4736                        ADDI   $10 -16
4737                        SUB    $10 $3
4738                        EXCH   $13 $9
4739                        XOR    $12 $13
4740                        EXCH   $13 $9
4741                        ADDI   $12 2
4742                        ADD    $12 $11
4743                        ADD    $10 $3
4744                        ADDI   $10 16
4745                        EXCH   $11 $10
4746                        ADDI   $10 -16
4747                        SUB    $10 $3
4748                        ADDI   $9 -3
4749                        SUB    $9 $3
4750                        EXCH   $14 $12
4751                        ADD    $9 $3
4752                        ADDI   $9 3
4753                        ADD    $10 $3
4754                        ADDI   $10 16
4755                        EXCH   $11 $10
4756                        ADDI   $10 -16
4757                        SUB    $10 $3
4758                        SUB    $12 $11
4759                        ADDI   $12 -2
4760                        EXCH   $13 $9
4761                        XOR    $12 $13
4762                        EXCH   $13 $9
4763                        ADD    $10 $3
4764                        ADDI   $10 16
4765                        EXCH   $11 $10
4766                        ADDI   $10 -16
4767                        SUB    $10 $3
4768                        ADDI   $9 -3
4769                        SUB    $9 $3
4770 cmp_top_200:          BNE    $8 $14
        cmp_bot_201
4771                        XORI   $15 1
4772 cmp_bot_201:          BNE    $8 $14
        cmp_top_200
4773                        ADD    $16 $3
4774                        ADDI   $16 5
4775                        ADD    $17 $3
4776                        ADDI   $17 16
```

```
4777                        EXCH   $18 $17
4778                        ADDI   $17 -16
4779                        SUB    $17 $3
4780                        EXCH   $20 $16
4781                        XOR    $19 $20
4782                        EXCH   $20 $16
4783                        ADDI   $19 2
4784                        ADD    $19 $18
4785                        ADD    $17 $3
4786                        ADDI   $17 16
4787                        EXCH   $18 $17
4788                        ADDI   $17 -16
4789                        SUB    $17 $3
4790                        ADDI   $16 -5
4791                        SUB    $16 $3
4792                        EXCH   $21 $19
4793                        ADD    $16 $3
4794                        ADDI   $16 5
4795                        ADD    $17 $3
4796                        ADDI   $17 16
4797                        EXCH   $18 $17
4798                        ADDI   $17 -16
4799                        SUB    $17 $3
4800                        SUB    $19 $18
4801                        ADDI   $19 -2
4802                        EXCH   $20 $16
4803                        XOR    $19 $20
4804                        EXCH   $20 $16
4805                        ADD    $17 $3
4806                        ADDI   $17 16
4807                        EXCH   $18 $17
4808                        ADDI   $17 -16
4809                        SUB    $17 $3
4810                        ADDI   $16 -5
4811                        SUB    $16 $3
4812                        ADD    $22 $3
4813                        ADDI   $22 7
4814                        EXCH   $23 $22
4815                        ADDI   $22 -7
4816                        SUB    $22 $3
4817 cmp_top_202:          BNE    $21 $23
         cmp_bot_203
4818                        XORI   $24 1
4819 cmp_bot_203:          BNE    $21 $23
         cmp_top_202
4820                        ANDX   $25 $15 $24
4821 f_top_204:            BEQ    $25 $0
         f_bot_205
4822                        XORI   $26 1
4823 f_bot_205:            BEQ    $25 $0
         f_top_204
4824                        XOR    $6 $26
4825 f_bot_205_i:          BEQ    $25 $0
         f_top_204_i
4826                        XORI   $26 1
4827 f_top_204_i:          BEQ    $25 $0
         f_bot_205_i
4828                        ANDX   $25 $15 $24
4829 cmp_bot_203_i:        BNE    $21 $23
         cmp_top_202_i
4830                        XORI   $24 1
4831 cmp_top_202_i:        BNE    $21 $23
         cmp_bot_203_i
4832                        ADD    $22 $3
4833                        ADDI   $22 7
4834                        EXCH   $23 $22
4835                        ADDI   $22 -7
4836                        SUB    $22 $3
4837                        ADD    $16 $3
4838                        ADDI   $16 5
4839                        ADD    $17 $3
4840                        ADDI   $17 16
4841                        EXCH   $18 $17
4842                        ADDI   $17 -16
4843                        SUB    $17 $3
4844                        EXCH   $20 $16
4845                        XOR    $19 $20
4846                        EXCH   $20 $16
4847                        ADDI   $19 2
4848                        ADD    $19 $18
4849                        ADD    $17 $3
4850                        ADDI   $17 16
4851                        EXCH   $18 $17
4852                        ADDI   $17 -16
4853                        SUB    $17 $3
4854                        ADDI   $16 -5
4855                        SUB    $16 $3
4856                        EXCH   $21 $19
4857                        ADD    $16 $3
4858                        ADDI   $16 5
4859                        ADD    $17 $3
4860                        ADDI   $17 16
4861                        EXCH   $18 $17
4862                        ADDI   $17 -16
4863                        SUB    $17 $3
4864                        SUB    $19 $18
4865                        ADDI   $19 -2
4866                        EXCH   $20 $16
4867                        XOR    $19 $20
4868                        EXCH   $20 $16
4869                        ADD    $17 $3
4870                        ADDI   $17 16
4871                        EXCH   $18 $17
4872                        ADDI   $17 -16
4873                        SUB    $17 $3
4874                        ADDI   $16 -5
4875                        SUB    $16 $3
4876 cmp_bot_201_i:        BNE    $8 $14
         cmp_top_200_i
4877                        XORI   $15 1
4878 cmp_top_200_i:        BNE    $8 $14
         cmp_bot_201_i
4879                        ADD    $9 $3
4880                        ADDI   $9 3
4881                        ADD    $10 $3
4882                        ADDI   $10 16
4883                        EXCH   $11 $10
4884                        ADDI   $10 -16
4885                        SUB    $10 $3
4886                        EXCH   $13 $9
4887                        XOR    $12 $13
4888                        EXCH   $13 $9
4889                        ADDI   $12 2
4890                        ADD    $12 $11
4891                        ADD    $10 $3
4892                        ADDI   $10 16
4893                        EXCH   $11 $10
4894                        ADDI   $10 -16
4895                        SUB    $10 $3
4896                        ADDI   $9 -3
4897                        SUB    $9 $3
4898                        EXCH   $14 $12
```

```
4899                            ADD     $9 $3
4900                            ADDI    $9 3
4901                            ADD     $10 $3
4902                            ADDI    $10 16
4903                            EXCH    $11 $10
4904                            ADDI    $10 -16
4905                            SUB     $10 $3
4906                            SUB     $12 $11
4907                            ADDI    $12 -2
4908                            EXCH    $13 $9
4909                            XOR     $12 $13
4910                            EXCH    $13 $9
4911                            ADD     $10 $3
4912                            ADDI    $10 16
4913                            EXCH    $11 $10
4914                            ADDI    $10 -16
4915                            SUB     $10 $3
4916                            ADDI    $9 -3
4917                            SUB     $9 $3
4918                            ADD     $7 $3
4919                            ADDI    $7 11
4920                            EXCH    $8 $7
4921                            ADDI    $7 -11
4922                            SUB     $7 $3
4923 test_196:                  BEQ     $6 $0
         test_false_198
4924                            XORI    $6 1
4925                            ADD     $7 $3
4926                            ADDI    $7 11
4927                            EXCH    $8 $7
4928                            ADDI    $7 -11
4929                            SUB     $7 $3
4930                            ADD     $9 $3
4931                            ADDI    $9 4
4932                            ADD     $10 $3
4933                            ADDI    $10 16
4934                            EXCH    $11 $10
4935                            ADDI    $10 -16
4936                            SUB     $10 $3
4937                            EXCH    $13 $9
4938                            XOR     $12 $13
4939                            EXCH    $13 $9
4940                            ADDI    $12 2
4941                            ADD     $12 $11
4942                            ADD     $10 $3
4943                            ADDI    $10 16
4944                            EXCH    $11 $10
4945                            ADDI    $10 -16
4946                            SUB     $10 $3
4947                            ADDI    $9 -4
4948                            SUB     $9 $3
4949                            EXCH    $14 $12
4950                            ADD     $9 $3
4951                            ADDI    $9 4
4952                            ADD     $10 $3
4953                            ADDI    $10 16
4954                            EXCH    $11 $10
4955                            ADDI    $10 -16
4956                            SUB     $10 $3
4957                            SUB     $12 $11
4958                            ADDI    $12 -2
4959                            EXCH    $13 $9
4960                            XOR     $12 $13
4961                            EXCH    $13 $9
4962                            ADD     $10 $3
4963                            ADDI    $10 16
4964                            EXCH    $11 $10
4965                            ADDI    $10 -16
4966                            SUB     $10 $3
4967                            ADDI    $9 -4
4968                            SUB     $9 $3
4969                            ADD     $15 $3
4970                            ADDI    $15 3
4971                            ADD     $16 $3
4972                            ADDI    $16 16
4973                            EXCH    $17 $16
4974                            ADDI    $16 -16
4975                            SUB     $16 $3
4976                            EXCH    $19 $15
4977                            XOR     $18 $19
4978                            EXCH    $19 $15
4979                            ADDI    $18 2
4980                            ADD     $18 $17
4981                            ADD     $16 $3
4982                            ADDI    $16 16
4983                            EXCH    $17 $16
4984                            ADDI    $16 -16
4985                            SUB     $16 $3
4986                            ADDI    $15 -3
4987                            SUB     $15 $3
4988                            EXCH    $20 $18
4989                            ADD     $15 $3
4990                            ADDI    $15 3
4991                            ADD     $16 $3
4992                            ADDI    $16 16
4993                            EXCH    $17 $16
4994                            ADDI    $16 -16
4995                            SUB     $16 $3
4996                            SUB     $18 $17
4997                            ADDI    $18 -2
4998                            EXCH    $19 $15
4999                            XOR     $18 $19
5000                            EXCH    $19 $15
5001                            ADD     $16 $3
5002                            ADDI    $16 16
5003                            EXCH    $17 $16
5004                            ADDI    $16 -16
5005                            SUB     $16 $3
5006                            ADDI    $15 -3
5007                            SUB     $15 $3
5008                            XOR     $21 $14
5009                            SUB     $21 $20
5010                            ADD     $8 $21
5011                            ADD     $21 $20
5012                            XOR     $21 $14
5013                            ADD     $15 $3
5014                            ADDI    $15 3
5015                            ADD     $16 $3
5016                            ADDI    $16 16
5017                            EXCH    $17 $16
5018                            ADDI    $16 -16
5019                            SUB     $16 $3
5020                            EXCH    $19 $15
5021                            XOR     $18 $19
5022                            EXCH    $19 $15
5023                            ADDI    $18 2
5024                            ADD     $18 $17
5025                            ADD     $16 $3
5026                            ADDI    $16 16
5027                            EXCH    $17 $16
5028                            ADDI    $16 -16
5029                            SUB     $16 $3
```

```
5030                    ADDI    $15 -3
5031                    SUB     $15 $3
5032                    EXCH    $20 $18
5033                    ADD     $15 $3
5034                    ADDI    $15 3
5035                    ADD     $16 $3
5036                    ADDI    $16 16
5037                    EXCH    $17 $16
5038                    ADDI    $16 -16
5039                    SUB     $16 $3
5040                    SUB     $18 $17
5041                    ADDI    $18 -2
5042                    EXCH    $19 $15
5043                    XOR     $18 $19
5044                    EXCH    $19 $15
5045                    ADD     $16 $3
5046                    ADDI    $16 16
5047                    EXCH    $17 $16
5048                    ADDI    $16 -16
5049                    SUB     $16 $3
5050                    ADDI    $15 -3
5051                    SUB     $15 $3
5052                    ADD     $9 $3
5053                    ADDI    $9 4
5054                    ADD     $10 $3
5055                    ADDI    $10 16
5056                    EXCH    $11 $10
5057                    ADDI    $10 -16
5058                    SUB     $10 $3
5059                    EXCH    $13 $9
5060                    XOR     $12 $13
5061                    EXCH    $13 $9
5062                    ADDI    $12 2
5063                    ADD     $12 $11
5064                    ADD     $10 $3
5065                    ADDI    $10 16
5066                    EXCH    $11 $10
5067                    ADDI    $10 -16
5068                    SUB     $10 $3
5069                    ADDI    $9 -4
5070                    SUB     $9 $3
5071                    EXCH    $14 $12
5072                    ADD     $9 $3
5073                    ADDI    $9 4
5074                    ADD     $10 $3
5075                    ADDI    $10 16
5076                    EXCH    $11 $10
5077                    ADDI    $10 -16
5078                    SUB     $10 $3
5079                    SUB     $12 $11
5080                    ADDI    $12 -2
5081                    EXCH    $13 $9
5082                    XOR     $12 $13
5083                    EXCH    $13 $9
5084                    ADD     $10 $3
5085                    ADDI    $10 16
5086                    EXCH    $11 $10
5087                    ADDI    $10 -16
5088                    SUB     $10 $3
5089                    ADDI    $9 -4
5090                    SUB     $9 $3
5091                    ADD     $7 $3
5092                    ADDI    $7 11
5093                    EXCH    $8 $7
5094                    ADDI    $7 -11
5095                    SUB     $7 $3
5096                    ADD     $8 $3
5097                    ADDI    $8 6
5098                    ADD     $9 $3
5099                    ADDI    $9 16
5100                    EXCH    $10 $9
5101                    ADDI    $9 -16
5102                    SUB     $9 $3
5103                    EXCH    $12 $8
5104                    XOR     $11 $12
5105                    EXCH    $12 $8
5106                    ADDI    $11 2
5107                    ADD     $11 $10
5108                    ADD     $9 $3
5109                    ADDI    $9 16
5110                    EXCH    $10 $9
5111                    ADDI    $9 -16
5112                    SUB     $9 $3
5113                    ADDI    $8 -6
5114                    SUB     $8 $3
5115                    EXCH    $13 $11
5116                    ADD     $8 $3
5117                    ADDI    $8 6
5118                    ADD     $9 $3
5119                    ADDI    $9 16
5120                    EXCH    $10 $9
5121                    ADDI    $9 -16
5122                    SUB     $9 $3
5123                    SUB     $11 $10
5124                    ADDI    $11 -2
5125                    EXCH    $12 $8
5126                    XOR     $11 $12
5127                    EXCH    $12 $8
5128                    ADD     $9 $3
5129                    ADDI    $9 16
5130                    EXCH    $10 $9
5131                    ADDI    $9 -16
5132                    SUB     $9 $3
5133                    ADDI    $8 -6
5134                    SUB     $8 $3
5135                    ADD     $14 $3
5136                    ADDI    $14 9
5137                    EXCH    $15 $14
5138                    ADDI    $14 -9
5139                    SUB     $14 $3
5140 cmp_top_210:       BNE     $13 $15
         cmp_bot_211
5141                    XORI    $16 1
5142 cmp_bot_211:       BNE     $13 $15
         cmp_top_210
5143 f_top_212:         BEQ     $16 $0
         f_bot_213
5144                    XORI    $17 1
5145 f_bot_213:         BEQ     $16 $0
         f_top_212
5146                    XOR     $7 $17
5147 f_bot_213_i:       BEQ     $16 $0
         f_top_212_i
5148                    XORI    $17 1
5149 f_top_212_i:       BEQ     $16 $0
         f_bot_213_i
5150 cmp_bot_211_i:     BNE     $13 $15
         cmp_top_210_i
5151                    XORI    $16 1
5152 cmp_top_210_i:     BNE     $13 $15
         cmp_bot_211_i
5153                    ADD     $14 $3
```

```
                        ADDI    $14 9
                        EXCH    $15 $14
                        ADDI    $14 -9
                        SUB     $14 $3
                        ADD     $8 $3
                        ADDI    $8 6
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        EXCH    $12 $8
                        XOR     $11 $12
                        EXCH    $12 $8
                        ADDI    $11 2
                        ADD     $11 $10
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        ADDI    $8 -6
                        SUB     $8 $3
                        EXCH    $13 $11
                        ADD     $8 $3
                        ADDI    $8 6
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        SUB     $11 $10
                        ADDI    $11 -2
                        EXCH    $12 $8
                        XOR     $11 $12
                        EXCH    $12 $8
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        ADDI    $8 -6
                        SUB     $8 $3
test_206:               BEQ     $7 $0
    test_false_208
                        XORI    $7 1
                        EXCH    $3 $1
                        ADDI    $1 -1
                        BRA     l_moveRight_7
                        ADDI    $1 1
                        EXCH    $3 $1
                        XORI    $7 1
assert_true_207:        BRA     assert_209
test_false_208:         BRA     test_206
assert_209:             BNE     $7 $0
    assert_true_207
                        ADD     $8 $3
                        ADDI    $8 6
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        EXCH    $12 $8
                        XOR     $11 $12
                        EXCH    $12 $8
                        ADDI    $11 2
                        ADD     $11 $10
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        SUB     $11 $10
                        ADDI    $11 -2
                        EXCH    $12 $8
                        XOR     $11 $12
                        EXCH    $12 $8
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        ADDI    $8 -6
                        SUB     $8 $3
                        ADD     $14 $3
                        ADDI    $14 9
                        EXCH    $15 $14
                        ADDI    $14 -9
                        SUB     $14 $3
cmp_top_214:            BNE     $13 $15
    cmp_bot_215
                        XORI    $16 1
cmp_bot_215:            BNE     $13 $15
    cmp_top_214
f_top_216:              BEQ     $16 $0
    f_bot_217
                        XORI    $17 1
f_bot_217:              BEQ     $16 $0
    f_top_216
                        XOR     $7 $17
f_bot_217_i:            BEQ     $16 $0
    f_top_216_i
                        XORI    $17 1
f_top_216_i:            BEQ     $16 $0
    f_bot_217_i
cmp_bot_215_i:          BNE     $13 $15
    cmp_top_214_i
                        XORI    $16 1
cmp_top_214_i:          BNE     $13 $15
    cmp_bot_215_i
                        ADD     $14 $3
                        ADDI    $14 9
                        EXCH    $15 $14
                        ADDI    $14 -9
                        SUB     $14 $3
                        ADD     $8 $3
                        ADDI    $8 6
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
```

```
5276                    SUB     $9 $3
5277                    EXCH    $12 $8
5278                    XOR     $11 $12
5279                    EXCH    $12 $8
5280                    ADDI    $11 2
5281                    ADD     $11 $10
5282                    ADD     $9 $3
5283                    ADDI    $9 16
5284                    EXCH    $10 $9
5285                    ADDI    $9 -16
5286                    SUB     $9 $3
5287                    ADDI    $8 -6
5288                    SUB     $8 $3
5289                    EXCH    $13 $11
5290                    ADD     $8 $3
5291                    ADDI    $8 6
5292                    ADD     $9 $3
5293                    ADDI    $9 16
5294                    EXCH    $10 $9
5295                    ADDI    $9 -16
5296                    SUB     $9 $3
5297                    SUB     $11 $10
5298                    ADDI    $11 -2
5299                    EXCH    $12 $8
5300                    XOR     $11 $12
5301                    EXCH    $12 $8
5302                    ADD     $9 $3
5303                    ADDI    $9 16
5304                    EXCH    $10 $9
5305                    ADDI    $9 -16
5306                    SUB     $9 $3
5307                    ADDI    $8 -6
5308                    SUB     $8 $3
5309                    ADD     $8 $3
5310                    ADDI    $8 6
5311                    ADD     $9 $3
5312                    ADDI    $9 16
5313                    EXCH    $10 $9
5314                    ADDI    $9 -16
5315                    SUB     $9 $3
5316                    EXCH    $12 $8
5317                    XOR     $11 $12
5318                    EXCH    $12 $8
5319                    ADDI    $11 2
5320                    ADD     $11 $10
5321                    ADD     $9 $3
5322                    ADDI    $9 16
5323                    EXCH    $10 $9
5324                    ADDI    $9 -16
5325                    SUB     $9 $3
5326                    ADDI    $8 -6
5327                    SUB     $8 $3
5328                    EXCH    $13 $11
5329                    ADD     $8 $3
5330                    ADDI    $8 6
5331                    ADD     $9 $3
5332                    ADDI    $9 16
5333                    EXCH    $10 $9
5334                    ADDI    $9 -16
5335                    SUB     $9 $3
5336                    SUB     $11 $10
5337                    ADDI    $11 -2
5338                    EXCH    $12 $8
5339                    XOR     $11 $12
5340                    EXCH    $12 $8
5341                    ADD     $9 $3
5342                    ADDI    $9 16
5343                    EXCH    $10 $9
5344                    ADDI    $9 -16
5345                    SUB     $9 $3
5346                    ADDI    $8 -6
5347                    SUB     $8 $3
5348                    ADD     $14 $3
5349                    ADDI    $14 8
5350                    EXCH    $15 $14
5351                    ADDI    $14 -8
5352                    SUB     $14 $3
5353 cmp_top_222:       BNE     $13 $15
         cmp_bot_223
5354                    XORI    $16 1
5355 cmp_bot_223:       BNE     $13 $15
         cmp_top_222
5356 f_top_224:         BEQ     $16 $0
         f_bot_225
5357                    XORI    $17 1
5358 f_bot_225:         BEQ     $16 $0
         f_top_224
5359                    XOR     $7 $17
5360 f_bot_225_i:       BEQ     $16 $0
         f_top_224_i
5361                    XORI    $17 1
5362 f_top_224_i:       BEQ     $16 $0
         f_bot_225_i
5363 cmp_bot_223_i:     BNE     $13 $15
         cmp_top_222_i
5364                    XORI    $16 1
5365 cmp_top_222_i:     BNE     $13 $15
         cmp_bot_223_i
5366                    ADD     $14 $3
5367                    ADDI    $14 8
5368                    EXCH    $15 $14
5369                    ADDI    $14 -8
5370                    SUB     $14 $3
5371                    ADD     $8 $3
5372                    ADDI    $8 6
5373                    ADD     $9 $3
5374                    ADDI    $9 16
5375                    EXCH    $10 $9
5376                    ADDI    $9 -16
5377                    SUB     $9 $3
5378                    EXCH    $12 $8
5379                    XOR     $11 $12
5380                    EXCH    $12 $8
5381                    ADDI    $11 2
5382                    ADD     $11 $10
5383                    ADD     $9 $3
5384                    ADDI    $9 16
5385                    EXCH    $10 $9
5386                    ADDI    $9 -16
5387                    SUB     $9 $3
5388                    ADDI    $8 -6
5389                    SUB     $8 $3
5390                    EXCH    $13 $11
5391                    ADD     $8 $3
5392                    ADDI    $8 6
5393                    ADD     $9 $3
5394                    ADDI    $9 16
5395                    EXCH    $10 $9
5396                    ADDI    $9 -16
5397                    SUB     $9 $3
5398                    SUB     $11 $10
5399                    ADDI    $11 -2
```

```
                        EXCH    $12 $8
                        XOR     $11 $12
                        EXCH    $12 $8
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        ADDI    $8 -6
                        SUB     $8 $3
test_218:               BEQ     $7 $0
    test_false_220
                        XORI    $7 1
                        EXCH    $3 $1
                        ADDI    $1 -1
                        RBRA    l_moveRight_7
                        ADDI    $1 1
                        EXCH    $3 $1
                        XORI    $7 1
assert_true_219:        BRA     assert_221
test_false_220:         BRA     test_218
assert_221:             BNE     $7 $0
    assert_true_219
                        ADD     $8 $3
                        ADDI    $8 6
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        EXCH    $12 $8
                        XOR     $11 $12
                        EXCH    $12 $8
                        ADDI    $11 2
                        ADD     $11 $10
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        ADDI    $8 -6
                        SUB     $8 $3
                        EXCH    $13 $11
                        ADD     $8 $3
                        ADDI    $8 6
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        SUB     $11 $10
                        ADDI    $11 -2
                        EXCH    $12 $8
                        XOR     $11 $12
                        EXCH    $12 $8
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        ADDI    $8 -6
                        SUB     $8 $3
                        ADD     $14 $3
                        ADDI    $14 8
                        EXCH    $15 $14
                        ADDI    $14 -8
                        SUB     $14 $3
cmp_top_226:            BNE     $13 $15
    cmp_bot_227
                        XORI    $16 1
cmp_bot_227:            BNE     $13 $15
    cmp_top_226
f_top_228:              BEQ     $16 $0
    f_bot_229
                        XORI    $17 1
f_bot_229:              BEQ     $16 $0
    f_top_228
                        XOR     $7 $17
f_bot_229_i:            BEQ     $16 $0
    f_top_228_i
                        XORI    $17 1
f_top_228_i:            BEQ     $16 $0
    f_bot_229_i
cmp_bot_227_i:          BNE     $13 $15
    cmp_top_226_i
                        XORI    $16 1
cmp_top_226_i:          BNE     $13 $15
    cmp_bot_227_i
                        ADD     $14 $3
                        ADDI    $14 8
                        EXCH    $15 $14
                        ADDI    $14 -8
                        SUB     $14 $3
                        ADD     $8 $3
                        ADDI    $8 6
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        EXCH    $12 $8
                        XOR     $11 $12
                        EXCH    $12 $8
                        ADDI    $11 2
                        ADD     $11 $10
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        ADDI    $8 -6
                        SUB     $8 $3
                        EXCH    $13 $11
                        ADD     $8 $3
                        ADDI    $8 6
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        SUB     $11 $10
                        ADDI    $11 -2
                        EXCH    $12 $8
                        XOR     $11 $12
                        EXCH    $12 $8
                        ADD     $9 $3
                        ADDI    $9 16
                        EXCH    $10 $9
                        ADDI    $9 -16
                        SUB     $9 $3
                        ADDI    $8 -6
                        SUB     $8 $3
```

```
                        XORI    $6 1
assert_true_197:        BRA     assert_199
test_false_198:         BRA     test_196
assert_199:             BNE     $6 $0
    assert_true_197
                        ADD     $7 $3
                        ADDI    $7 11
                        EXCH    $8 $7
                        ADDI    $7 -11
                        SUB     $7 $3
                        ADD     $9 $3
                        ADDI    $9 4
                        ADD     $10 $3
                        ADDI    $10 16
                        EXCH    $11 $10
                        ADDI    $10 -16
                        SUB     $10 $3
                        EXCH    $13 $9
                        XOR     $12 $13
                        EXCH    $13 $9
                        ADDI    $12 2
                        ADD     $12 $11
                        ADD     $10 $3
                        ADDI    $10 16
                        EXCH    $11 $10
                        ADDI    $10 -16
                        SUB     $10 $3
                        ADDI    $9 -4
                        SUB     $9 $3
                        EXCH    $14 $12
                        ADD     $9 $3
                        ADDI    $9 4
                        ADD     $10 $3
                        ADDI    $10 16
                        EXCH    $11 $10
                        ADDI    $10 -16
                        SUB     $10 $3
                        SUB     $12 $11
                        ADDI    $12 -2
                        EXCH    $13 $9
                        XOR     $12 $13
                        EXCH    $13 $9
                        ADD     $10 $3
                        ADDI    $10 16
                        EXCH    $11 $10
                        ADDI    $10 -16
                        SUB     $10 $3
                        ADDI    $9 -4
                        SUB     $9 $3
cmp_top_230:            BNE     $8 $14
    cmp_bot_231
                        XORI    $15 1
cmp_bot_231:            BNE     $8 $14
    cmp_top_230
                        ADD     $16 $3
                        ADDI    $16 5
                        ADD     $17 $3
                        ADDI    $17 16
                        EXCH    $18 $17
                        ADDI    $17 -16
                        SUB     $17 $3
                        EXCH    $20 $16
                        XOR     $19 $20
                        EXCH    $20 $16
                        ADDI    $19 2
                        ADD     $19 $18
                        ADD     $17 $3
                        ADDI    $17 16
                        EXCH    $18 $17
                        ADDI    $17 -16
                        SUB     $17 $3
                        ADDI    $16 -5
                        SUB     $16 $3
                        EXCH    $21 $19
                        ADD     $16 $3
                        ADDI    $16 5
                        ADD     $17 $3
                        ADDI    $17 16
                        EXCH    $18 $17
                        ADDI    $17 -16
                        SUB     $17 $3
                        SUB     $19 $18
                        ADDI    $19 -2
                        EXCH    $20 $16
                        XOR     $19 $20
                        EXCH    $20 $16
                        ADD     $17 $3
                        ADDI    $17 16
                        EXCH    $18 $17
                        ADDI    $17 -16
                        SUB     $17 $3
                        ADDI    $16 -5
                        SUB     $16 $3
                        ADD     $22 $3
                        ADDI    $22 7
                        EXCH    $23 $22
                        ADDI    $22 -7
                        SUB     $22 $3
cmp_top_232:            BNE     $21 $23
    cmp_bot_233
                        XORI    $24 1
cmp_bot_233:            BNE     $21 $23
    cmp_top_232
                        ANDX    $25 $15 $24
f_top_234:              BEQ     $25 $0
    f_bot_235
                        XORI    $26 1
f_bot_235:              BEQ     $25 $0
    f_top_234
                        XOR     $6 $26
f_bot_235_i:            BEQ     $25 $0
    f_top_234_i
                        XORI    $26 1
f_top_234_i:            BEQ     $25 $0
    f_bot_235_i
                        ANDX    $25 $15 $24
cmp_bot_233_i:          BNE     $21 $23
    cmp_top_232_i
                        XORI    $24 1
cmp_top_232_i:          BNE     $21 $23
    cmp_bot_233_i
                        ADD     $22 $3
                        ADDI    $22 7
                        EXCH    $23 $22
                        ADDI    $22 -7
                        SUB     $22 $3
                        ADD     $16 $3
                        ADDI    $16 5
                        ADD     $17 $3
                        ADDI    $17 16
                        EXCH    $18 $17
                        ADDI    $17 -16
```

```
5643                            SUB     $17 $3
5644                            EXCH    $20 $16
5645                            XOR     $19 $20
5646                            EXCH    $20 $16
5647                            ADDI    $19 2
5648                            ADD     $19 $18
5649                            ADD     $17 $3
5650                            ADDI    $17 16
5651                            EXCH    $18 $17
5652                            ADDI    $17 -16
5653                            SUB     $17 $3
5654                            ADDI    $16 -5
5655                            SUB     $16 $3
5656                            EXCH    $21 $19
5657                            ADD     $16 $3
5658                            ADDI    $16 5
5659                            ADD     $17 $3
5660                            ADDI    $17 16
5661                            EXCH    $18 $17
5662                            ADDI    $17 -16
5663                            SUB     $17 $3
5664                            SUB     $19 $18
5665                            ADDI    $19 -2
5666                            EXCH    $20 $16
5667                            XOR     $19 $20
5668                            EXCH    $20 $16
5669                            ADD     $17 $3
5670                            ADDI    $17 16
5671                            EXCH    $18 $17
5672                            ADDI    $17 -16
5673                            SUB     $17 $3
5674                            ADDI    $16 -5
5675                            SUB     $16 $3
5676 cmp_bot_231_i:             BNE     $8 $14
        cmp_top_230_i
5677                            XORI    $15 1
5678 cmp_top_230_i:             BNE     $8 $14
        cmp_bot_231_i
5679                            ADD     $9 $3
5680                            ADDI    $9 4
5681                            ADD     $10 $3
5682                            ADDI    $10 16
5683                            EXCH    $11 $10
5684                            ADDI    $10 -16
5685                            SUB     $10 $3
5686                            EXCH    $13 $9
5687                            XOR     $12 $13
5688                            EXCH    $13 $9
5689                            ADDI    $12 2
5690                            ADD     $12 $11
5691                            ADD     $10 $3
5692                            ADDI    $10 16
5693                            EXCH    $11 $10
5694                            ADDI    $10 -16
5695                            SUB     $10 $3
5696                            ADDI    $9 -4
5697                            SUB     $9 $3
5698                            EXCH    $14 $12
5699                            ADD     $9 $3
5700                            ADDI    $9 4
5701                            ADD     $10 $3
5702                            ADDI    $10 16
5703                            EXCH    $11 $10
5704                            ADDI    $10 -16
5705                            SUB     $10 $3
5706                            SUB     $12 $11
5707                            ADDI    $12 -2
5708                            EXCH    $13 $9
5709                            XOR     $12 $13
5710                            EXCH    $13 $9
5711                            ADD     $10 $3
5712                            ADDI    $10 16
5713                            EXCH    $11 $10
5714                            ADDI    $10 -16
5715                            SUB     $10 $3
5716                            ADDI    $9 -4
5717                            SUB     $9 $3
5718                            ADD     $7 $3
5719                            ADDI    $7 11
5720                            EXCH    $8 $7
5721                            ADDI    $7 -11
5722                            SUB     $7 $3
5723 l_inst_6_bot:              BRA     l_inst_6_top
5724 l_moveRight_7_top:         BRA
        l_moveRight_7_bot
5725                            ADDI    $1 1
5726                            EXCH    $2 $1
5727                            EXCH    $3 $1
5728                            ADDI    $1 -1
5729 l_moveRight_7:             SWAPBR  $2
5730                            NEG     $2
5731                            ADDI    $1 1
5732                            EXCH    $3 $1
5733                            EXCH    $2 $1
5734                            ADDI    $1 -1
5735 localBlock_302:            XOR     $6 $1
5736                            XOR     $7 $0
5737                            EXCH    $7 $1
5738                            ADDI    $1 -1
5739 localBlock_301:            XOR     $7 $1
5740                            XOR     $8 $0
5741                            EXCH    $8 $1
5742                            ADDI    $1 -1
5743                            ADD     $8 $3
5744                            ADDI    $8 2
5745                            EXCH    $9 $8
5746                            ADDI    $8 -2
5747                            SUB     $8 $3
5748                            XOR     $10 $9
5749 loadMetAdd_236:            EXCH    $11 $10
5750                            ADDI    $11 3
5751                            EXCH    $12 $11
5752                            XOR     $13 $12
5753                            EXCH    $12 $11
5754                            ADDI    $11 -3
5755                            EXCH    $11 $10
5756                            ADD     $8 $3
5757                            ADDI    $8 2
5758                            EXCH    $9 $8
5759                            ADDI    $8 -2
5760                            SUB     $8 $3
5761                            ADD     $14 $3
5762                            ADDI    $14 14
5763                            EXCH    $3 $1
5764                            ADDI    $1 -1
5765                            EXCH    $7 $1
5766                            ADDI    $1 -1
5767                            EXCH    $6 $1
5768                            ADDI    $1 -1
5769                            EXCH    $14 $1
5770                            ADDI    $1 -1
5771                            EXCH    $10 $1
```

```
                        ADDI   $1 -1
                        ADDI   $13 -5773
l_rjmp_top_238:         RBRA   l_rjmp_bot_239
l_jmp_237:              SWAPBR $13
                        NEG    $13
l_rjmp_bot_239:         BRA    l_rjmp_top_238
                        ADDI   $13 5773
                        ADDI   $1 1
                        EXCH   $10 $1
                        ADDI   $1 1
                        EXCH   $14 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        ADDI   $1 1
                        EXCH   $7 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        ADDI   $14 -14
                        SUB    $14 $3
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
                        EXCH   $11 $10
                        ADDI   $11 3
                        EXCH   $12 $11
                        XOR    $13 $12
                        EXCH   $12 $11
                        ADDI   $11 -3
loadMetAdd_236_i:       EXCH   $11 $10
                        XOR    $10 $9
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
                        XOR    $10 $9
loadMetAdd_240:         EXCH   $11 $10
                        ADDI   $11 1
                        EXCH   $12 $11
                        XOR    $13 $12
                        EXCH   $12 $11
                        ADDI   $11 -1
                        EXCH   $11 $10
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
                        EXCH   $3 $1
                        ADDI   $1 -1
                        EXCH   $7 $1
                        ADDI   $1 -1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $10 $1
                        ADDI   $1 -1
                        ADDI   $13 -5834
l_jmp_241:              SWAPBR $13
                        NEG    $13
                        ADDI   $13 5834
                        ADDI   $1 1
                        EXCH   $10 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        ADDI   $1 1
                        EXCH   $7 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
                        EXCH   $11 $10
                        ADDI   $11 1
                        EXCH   $12 $11
                        XOR    $13 $12
                        EXCH   $12 $11
                        ADDI   $11 -1
loadMetAdd_240_i:       EXCH   $11 $10
                        XOR    $10 $9
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
                        EXCH   $9 $6
cmp_top_246:            BNE    $9 $0
                            cmp_bot_247
                        XORI   $10 1
cmp_bot_247:            BNE    $9 $0
                            cmp_top_246
                        ADD    $11 $3
                        ADDI   $11 14
                        EXCH   $12 $11
                        ADDI   $11 -14
                        SUB    $11 $3
                        ADD    $13 $3
                        ADDI   $13 10
                        EXCH   $14 $13
                        ADDI   $13 -10
                        SUB    $13 $3
cmp_top_248:            BNE    $12 $14
                            cmp_bot_249
                        XORI   $15 1
cmp_bot_249:            BNE    $12 $14
                            cmp_top_248
                        ANDX   $16 $10 $15
f_top_250:              BEQ    $16 $0
                            f_bot_251
                        XORI   $17 1
f_bot_251:              BEQ    $16 $0
                            f_top_250
                        XOR    $8 $17
f_bot_251_i:            BEQ    $16 $0
                            f_top_250_i
                        XORI   $17 1
f_top_250_i:            BEQ    $16 $0
                            f_bot_251_i
                        ANDX   $16 $10 $15
cmp_bot_249_i:          BNE    $12 $14
                            cmp_top_248_i
                        XORI   $15 1
cmp_top_248_i:          BNE    $12 $14
                            cmp_bot_249_i
```

```
5894                            ADD     $13 $3
5895                            ADDI    $13 10
5896                            EXCH    $14 $13
5897                            ADDI    $13 -10
5898                            SUB     $13 $3
5899                            ADD     $11 $3
5900                            ADDI    $11 14
5901                            EXCH    $12 $11
5902                            ADDI    $11 -14
5903                            SUB     $11 $3
5904 cmp_bot_247_i:             BNE     $9 $0
         cmp_top_246_i
5905                            XORI    $10 1
5906 cmp_top_246_i:             BNE     $9 $0
         cmp_bot_247_i
5907                            EXCH    $9 $6
5908 test_242:                  BEQ     $8 $0
         test_false_244
5909                            XORI    $8 1
5910                            ADD     $9 $3
5911                            ADDI    $9 14
5912                            EXCH    $10 $9
5913                            ADDI    $9 -14
5914                            SUB     $9 $3
5915                            ADD     $11 $3
5916                            ADDI    $11 10
5917                            EXCH    $12 $11
5918                            ADDI    $11 -10
5919                            SUB     $11 $3
5920                            XOR     $10 $12
5921                            ADD     $11 $3
5922                            ADDI    $11 10
5923                            EXCH    $12 $11
5924                            ADDI    $11 -10
5925                            SUB     $11 $3
5926                            ADD     $9 $3
5927                            ADDI    $9 14
5928                            EXCH    $10 $9
5929                            ADDI    $9 -14
5930                            SUB     $9 $3
5931                            EXCH    $3 $1
5932                            ADDI    $1 -1
5933                            EXCH    $7 $1
5934                            ADDI    $1 -1
5935                            EXCH    $6 $1
5936                            ADDI    $1 -1
5937 obj_con_252:               ADDI    $10 8
5938                            EXCH    $10 $1
5939                            ADDI    $1 -1
5940                            EXCH    $9 $1
5941                            ADDI    $1 -1
5942                            BRA     l_malloc
5943                            ADDI    $1 1
5944                            EXCH    $9 $1
5945                            ADDI    $1 1
5946                            EXCH    $10 $1
5947 obj_con_252_i:             ADDI    $10 -8
5948                            ADDI    $1 1
5949                            EXCH    $6 $1
5950                            ADDI    $1 1
5951                            EXCH    $7 $1
5952                            ADDI    $1 1
5953                            EXCH    $3 $1
5954                            XORI    $10 10
5955                            EXCH    $10 $9
5956                            ADDI    $9 1
5957                            XORI    $10 1
5958                            EXCH    $10 $9
5959 obj_con_252_bot:           ADDI    $9 -1
5960                            EXCH    $9 $6
5961                            EXCH    $9 $7
5962                            EXCH    $10 $6
5963 copy_253:                  XOR     $9 $10
5964                            ADDI    $10 1
5965                            EXCH    $11 $10
5966                            ADDI    $11 1
5967                            EXCH    $11 $10
5968                            ADDI    $10 -1
5969                            EXCH    $10 $6
5970                            EXCH    $9 $7
5971                            EXCH    $9 $6
5972                            XOR     $10 $9
5973 loadMetAdd_254:            EXCH    $11 $10
5974                            ADDI    $11 2
5975                            EXCH    $12 $11
5976                            XOR     $13 $12
5977                            EXCH    $12 $11
5978                            ADDI    $11 -2
5979                            EXCH    $11 $10
5980                            EXCH    $9 $6
5981                            EXCH    $3 $1
5982                            ADDI    $1 -1
5983                            EXCH    $6 $1
5984                            ADDI    $1 -1
5985                            EXCH    $7 $1
5986                            ADDI    $1 -1
5987                            EXCH    $10 $1
5988                            ADDI    $1 -1
5989                            ADDI    $13 -5989
5990 l_rjmp_top_256:            RBRA    l_rjmp_bot_257
5991 l_jmp_255:                 SWAPBR  $13
5992                            NEG     $13
5993 l_rjmp_bot_257:            BRA     l_rjmp_top_256
5994                            ADDI    $13 5989
5995                            ADDI    $1 1
5996                            EXCH    $10 $1
5997                            ADDI    $1 1
5998                            EXCH    $7 $1
5999                            ADDI    $1 1
6000                            EXCH    $6 $1
6001                            ADDI    $1 1
6002                            EXCH    $3 $1
6003                            EXCH    $9 $6
6004                            EXCH    $11 $10
6005                            ADDI    $11 2
6006                            EXCH    $12 $11
6007                            XOR     $13 $12
6008                            EXCH    $12 $11
6009                            ADDI    $11 -2
6010 loadMetAdd_254_i:          EXCH    $11 $10
6011                            XOR     $10 $9
6012                            EXCH    $9 $6
6013                            EXCH    $9 $6
6014                            XOR     $10 $9
6015 loadMetAdd_258:            EXCH    $11 $10
6016                            ADDI    $11 0
6017                            EXCH    $12 $11
6018                            XOR     $13 $12
6019                            EXCH    $12 $11
6020                            ADDI    $11 0
6021                            EXCH    $11 $10
6022                            EXCH    $9 $6
```

```
                        ADD     $14 $3
                        ADDI    $14 2
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $14 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        ADDI    $13 -6035
l_rjmp_top_260:         RBRA    l_rjmp_bot_261
l_jmp_259:              SWAPBR  $13
                        NEG     $13
l_rjmp_bot_261:         BRA     l_rjmp_top_260
                        ADDI    $13 6035
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $14 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        ADDI    $14 -2
                        SUB     $14 $3
                        EXCH    $9 $6
                        EXCH    $11 $10
                        ADDI    $11 0
                        EXCH    $12 $11
                        XOR     $13 $12
                        EXCH    $12 $11
                        ADDI    $11 0
loadMetAdd_258_i:       EXCH    $11 $10
                        XOR     $10 $9
                        EXCH    $9 $6
                        EXCH    $9 $6
                        ADD     $10 $3
                        ADDI    $10 2
                        EXCH    $11 $10
                        ADDI    $10 -2
                        SUB     $10 $3
swap_262:               XOR     $9 $11
                        XOR     $11 $9
                        XOR     $9 $11
                        ADD     $10 $3
                        ADDI    $10 2
                        EXCH    $11 $10
                        ADDI    $10 -2
                        SUB     $10 $3
                        EXCH    $9 $6
                        XORI    $8 1
assert_true_243:        BRA     assert_245
test_false_244:         BRA     test_242
                        EXCH    $9 $6
                        XOR     $10 $9
loadMetAdd_263:         EXCH    $11 $10
                        ADDI    $11 0
                        EXCH    $12 $11
                        XOR     $13 $12
                        EXCH    $12 $11
                        ADDI    $11 0
                        EXCH    $11 $10
                        EXCH    $9 $6
                        EXCH    $3 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $10 $1
                        ADDI    $1 -1
                        ADDI    $13 -6098
l_jmp_264:              SWAPBR  $13
                        NEG     $13
                        ADDI    $13 6098
                        ADDI    $1 1
                        EXCH    $10 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $9 $6
                        EXCH    $11 $10
                        ADDI    $11 0
                        EXCH    $12 $11
                        XOR     $13 $12
                        EXCH    $12 $11
                        ADDI    $11 0
loadMetAdd_263_i:       EXCH    $11 $10
                        XOR     $10 $9
                        EXCH    $9 $6
                        ADD     $9 $3
                        ADDI    $9 2
                        EXCH    $10 $9
                        ADDI    $9 -2
                        SUB     $9 $3
                        EXCH    $11 $7
uncopy_265:             XOR     $10 $11
                        ADDI    $11 1
                        EXCH    $12 $11
                        ADDI    $12 -1
                        EXCH    $12 $11
                        ADDI    $11 -1
                        EXCH    $11 $7
                        ADD     $9 $3
                        ADDI    $9 2
                        EXCH    $10 $9
                        ADDI    $9 -2
                        SUB     $9 $3
                        ADD     $10 $3
                        ADDI    $10 2
                        EXCH    $11 $10
                        ADDI    $10 -2
                        SUB     $10 $3
cmp_top_270:            BNE     $11 $0
                            cmp_bot_271
                        XORI    $12 1
cmp_bot_271:            BNE     $11 $0
                            cmp_top_270
                        ADD     $13 $3
                        ADDI    $13 14
                        EXCH    $14 $13
                        ADDI    $13 -14
                        SUB     $13 $3
                        ADD     $15 $3
```

```
6153                            ADDI    $15 10
6154                            EXCH    $16 $15
6155                            ADDI    $15 -10
6156                            SUB     $15 $3
6157 cmp_top_272:               BNE     $14 $16
         cmp_bot_273
6158                            XORI    $17 1
6159 cmp_bot_273:               BNE     $14 $16
         cmp_top_272
6160                            ANDX    $18 $12 $17
6161 f_top_274:                 BEQ     $18 $0
         f_bot_275
6162                            XORI    $19 1
6163 f_bot_275:                 BEQ     $18 $0
         f_top_274
6164                            XOR     $9 $19
6165 f_bot_275_i:               BEQ     $18 $0
         f_top_274_i
6166                            XORI    $19 1
6167 f_top_274_i:               BEQ     $18 $0
         f_bot_275_i
6168                            ANDX    $18 $12 $17
6169 cmp_bot_273_i:             BNE     $14 $16
         cmp_top_272_i
6170                            XORI    $17 1
6171 cmp_top_272_i:             BNE     $14 $16
         cmp_bot_273_i
6172                            ADD     $15 $3
6173                            ADDI    $15 10
6174                            EXCH    $16 $15
6175                            ADDI    $15 -10
6176                            SUB     $15 $3
6177                            ADD     $13 $3
6178                            ADDI    $13 14
6179                            EXCH    $14 $13
6180                            ADDI    $13 -14
6181                            SUB     $13 $3
6182 cmp_bot_271_i:             BNE     $11 $0
         cmp_top_270_i
6183                            XORI    $12 1
6184 cmp_top_270_i:             BNE     $11 $0
         cmp_bot_271_i
6185                            ADD     $10 $3
6186                            ADDI    $10 2
6187                            EXCH    $11 $10
6188                            ADDI    $10 -2
6189                            SUB     $10 $3
6190 test_266:                  BEQ     $9 $0
         test_false_268
6191                            XORI    $9 1
6192                            EXCH    $10 $7
6193                            XOR     $11 $10
6194 loadMetAdd_276:            EXCH    $12 $11
6195                            ADDI    $12 2
6196                            EXCH    $13 $12
6197                            XOR     $14 $13
6198                            EXCH    $13 $12
6199                            ADDI    $12 -2
6200                            EXCH    $12 $11
6201                            EXCH    $10 $7
6202                            ADD     $15 $3
6203                            ADDI    $15 2
6204                            EXCH    $3 $1
6205                            ADDI    $1 -1
6206                            EXCH    $7 $1
6207                            ADDI    $1 -1
6208                            EXCH    $6 $1
6209                            ADDI    $1 -1
6210                            EXCH    $15 $1
6211                            ADDI    $1 -1
6212                            EXCH    $11 $1
6213                            ADDI    $1 -1
6214                            ADDI    $14 -6213
6215 l_jmp_277:                 SWAPBR  $14
6216                            NEG     $14
6217                            ADDI    $14 6213
6218                            ADDI    $1 1
6219                            EXCH    $11 $1
6220                            ADDI    $1 1
6221                            EXCH    $15 $1
6222                            ADDI    $1 1
6223                            EXCH    $6 $1
6224                            ADDI    $1 1
6225                            EXCH    $7 $1
6226                            ADDI    $1 1
6227                            EXCH    $3 $1
6228                            ADDI    $15 -2
6229                            SUB     $15 $3
6230                            EXCH    $10 $7
6231                            EXCH    $12 $11
6232                            ADDI    $12 2
6233                            EXCH    $13 $12
6234                            XOR     $14 $13
6235                            EXCH    $13 $12
6236                            ADDI    $12 -2
6237 loadMetAdd_276_i:          EXCH    $12 $11
6238                            XOR     $11 $10
6239                            EXCH    $10 $7
6240                            ADD     $10 $3
6241                            ADDI    $10 2
6242                            EXCH    $11 $10
6243                            ADDI    $10 -2
6244                            SUB     $10 $3
6245                            EXCH    $12 $7
6246 uncopy_278:                XOR     $11 $12
6247                            ADDI    $12 1
6248                            EXCH    $13 $12
6249                            ADDI    $13 -1
6250                            EXCH    $13 $12
6251                            ADDI    $12 -1
6252                            EXCH    $12 $7
6253                            ADD     $10 $3
6254                            ADDI    $10 2
6255                            EXCH    $11 $10
6256                            ADDI    $10 -2
6257                            SUB     $10 $3
6258                            EXCH    $10 $7
6259 obj_des_279_top:           EXCH    $11 $10
6260                            XORI    $11 10
6261                            ADDI    $10 1
6262                            EXCH    $11 $10
6263                            XORI    $11 1
6264                            ADDI    $10 -1
6265                            EXCH    $3 $1
6266                            ADDI    $1 -1
6267                            EXCH    $7 $1
6268                            ADDI    $1 -1
6269                            EXCH    $6 $1
6270                            ADDI    $1 -1
6271 obj_des_279:                ADDI    $11 8
6272                            EXCH    $11 $1
6273                            ADDI    $1 -1
```

```
6274                            EXCH    $10 $1
6275                            ADDI    $1 -1
6276                            RBRA    l_malloc
6277                            ADDI    $1 1
6278                            EXCH    $10 $1
6279                            ADDI    $1 1
6280                            EXCH    $11 $1
6281 obj_des_279_i:            ADDI    $11 -8
6282                            ADDI    $1 1
6283                            EXCH    $6 $1
6284                            ADDI    $1 1
6285                            EXCH    $7 $1
6286                            ADDI    $1 1
6287                            EXCH    $3 $1
6288                            EXCH    $10 $7
6289                            ADD     $10 $3
6290                            ADDI    $10 14
6291                            EXCH    $11 $10
6292                            ADDI    $10 -14
6293                            SUB     $10 $3
6294                            ADD     $12 $3
6295                            ADDI    $12 10
6296                            EXCH    $13 $12
6297                            ADDI    $12 -10
6298                            SUB     $12 $3
6299                            XOR     $11 $13
6300                            ADD     $12 $3
6301                            ADDI    $12 10
6302                            EXCH    $13 $12
6303                            ADDI    $12 -10
6304                            SUB     $12 $3
6305                            ADD     $10 $3
6306                            ADDI    $10 14
6307                            EXCH    $11 $10
6308                            ADDI    $10 -14
6309                            SUB     $10 $3
6310                            XORI    $9 1
6311 assert_true_267:          BRA     assert_269
6312 test_false_268:           BRA     test_266
6313 assert_269:                BNE     $9 $0
         assert_true_267
6314                            EXCH    $10 $7
6315 cmp_top_280:               BNE     $10 $0
         cmp_bot_281
6316                            XORI    $11 1
6317 cmp_bot_281:               BNE     $10 $0
         cmp_top_280
6318 f_top_282:                 BEQ     $11 $0
         f_bot_283
6319                            XORI    $12 1
6320 f_bot_283:                 BEQ     $11 $0
         f_top_282
6321                            XOR     $9 $12
6322 f_bot_283_i:               BEQ     $11 $0
         f_top_282_i
6323                            XORI    $12 1
6324 f_top_282_i:               BEQ     $11 $0
         f_bot_283_i
6325 cmp_bot_281_i:             BNE     $10 $0
         cmp_top_280_i
6326                            XORI    $11 1
6327 cmp_top_280_i:             BNE     $10 $0
         cmp_bot_281_i
6328                            EXCH    $10 $7
6329                            EXCH    $9 $6
6330                            XOR     $10 $9
6331 loadMetAdd_284:            EXCH    $11 $10
6332                            ADDI    $11 0
6333                            EXCH    $12 $11
6334                            XOR     $13 $12
6335                            EXCH    $12 $11
6336                            ADDI    $11 0
6337                            EXCH    $11 $10
6338                            EXCH    $9 $6
6339                            EXCH    $3 $1
6340                            ADDI    $1 -1
6341                            EXCH    $6 $1
6342                            ADDI    $1 -1
6343                            EXCH    $7 $1
6344                            ADDI    $1 -1
6345                            EXCH    $10 $1
6346                            ADDI    $1 -1
6347                            ADDI    $13 -6347
6348 l_rjmp_top_286:            RBRA    l_rjmp_bot_287
6349 l_jmp_285:                 SWAPBR  $13
6350                            NEG     $13
6351 l_rjmp_bot_287:            BRA     l_rjmp_top_286
6352                            ADDI    $13 6347
6353                            ADDI    $1 1
6354                            EXCH    $10 $1
6355                            ADDI    $1 1
6356                            EXCH    $7 $1
6357                            ADDI    $1 1
6358                            EXCH    $6 $1
6359                            ADDI    $1 1
6360                            EXCH    $3 $1
6361                            EXCH    $9 $6
6362                            EXCH    $11 $10
6363                            ADDI    $11 0
6364                            EXCH    $12 $11
6365                            XOR     $13 $12
6366                            EXCH    $12 $11
6367                            ADDI    $11 0
6368 loadMetAdd_284_i:          EXCH    $11 $10
6369                            XOR     $10 $9
6370                            EXCH    $9 $6
6371                            ADD     $9 $3
6372                            ADDI    $9 2
6373                            EXCH    $10 $9
6374                            ADDI    $9 -2
6375                            SUB     $9 $3
6376                            EXCH    $11 $6
6377 swap_288:                  XOR     $10 $11
6378                            XOR     $11 $10
6379                            XOR     $10 $11
6380                            EXCH    $11 $6
6381                            ADD     $9 $3
6382                            ADDI    $9 2
6383                            EXCH    $10 $9
6384                            ADDI    $9 -2
6385                            SUB     $9 $3
6386                            ADD     $9 $3
6387                            ADDI    $9 2
6388                            EXCH    $10 $9
6389                            ADDI    $9 -2
6390                            SUB     $9 $3
6391                            XOR     $11 $10
6392 loadMetAdd_289:            EXCH    $12 $11
6393                            ADDI    $12 1
6394                            EXCH    $13 $12
6395                            XOR     $14 $13
6396                            EXCH    $13 $12
```

```
                        ADDI   $12 -1
                        EXCH   $12 $11
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
                        EXCH   $3 $1
                        ADDI   $1 -1
                        EXCH   $7 $1
                        ADDI   $1 -1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $11 $1
                        ADDI   $1 -1
                        ADDI   $14 -6411
l_jmp_290:              SWAPBR $14
                        NEG    $14
                        ADDI   $14 6411
                        ADDI   $1 1
                        EXCH   $11 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        ADDI   $1 1
                        EXCH   $7 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
                        EXCH   $12 $11
                        ADDI   $12 1
                        EXCH   $13 $12
                        XOR    $14 $13
                        EXCH   $13 $12
                        ADDI   $12 -1
loadMetAdd_289_i:       EXCH   $12 $11
                        XOR    $11 $10
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
                        XOR    $11 $10
loadMetAdd_291:         EXCH   $12 $11
                        ADDI   $12 3
                        EXCH   $13 $12
                        XOR    $14 $13
                        EXCH   $13 $12
                        ADDI   $12 -3
                        EXCH   $12 $11
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
                        ADD    $15 $3
                        ADDI   $15 14
                        EXCH   $3 $1
                        ADDI   $1 -1
                        EXCH   $7 $1
                        ADDI   $1 -1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $15 $1
                        ADDI   $1 -1
                        EXCH   $11 $1
                        ADDI   $1 -1
                        ADDI   $14 -6471
l_jmp_292:              SWAPBR $14
                        NEG    $14
                        ADDI   $14 6471
                        ADDI   $1 1
                        EXCH   $11 $1
                        ADDI   $1 1
                        EXCH   $15 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        ADDI   $1 1
                        EXCH   $7 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        ADDI   $15 -14
                        SUB    $15 $3
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
                        EXCH   $12 $11
                        ADDI   $12 3
                        EXCH   $13 $12
                        XOR    $14 $13
                        EXCH   $13 $12
                        ADDI   $12 -3
loadMetAdd_291_i:       EXCH   $12 $11
                        XOR    $11 $10
                        ADD    $9 $3
                        ADDI   $9 2
                        EXCH   $10 $9
                        ADDI   $9 -2
                        SUB    $9 $3
assert_245:             BNE    $8 $0
                            assert_true_243
                        EXCH   $9 $6
cmp_top_293:            BNE    $9 $0
                            cmp_bot_294
                        XORI   $10 1
cmp_bot_294:            BNE    $9 $0
                            cmp_top_293
f_top_295:              BEQ    $10 $0
                            f_bot_296
                        XORI   $11 1
f_bot_296:              BEQ    $10 $0
                            f_top_295
                        XOR    $8 $11
f_bot_296_i:            BEQ    $10 $0
                            f_top_295_i
                        XORI   $11 1
f_top_295_i:            BEQ    $10 $0
                            f_bot_296_i
cmp_bot_294_i:          BNE    $9 $0
                            cmp_top_293_i
                        XORI   $10 1
cmp_top_293_i:          BNE    $9 $0
```

```
    cmp_bot_294_i
6521                        EXCH    $9 $6
6522                        ADD     $8 $3
6523                        ADDI    $8 2
6524                        EXCH    $9 $8
6525                        ADDI    $8 -2
6526                        SUB     $8 $3
6527                        XOR     $10 $9
6528 loadMetAdd_297:        EXCH    $11 $10
6529                        ADDI    $11 1
6530                        EXCH    $12 $11
6531                        XOR     $13 $12
6532                        EXCH    $12 $11
6533                        ADDI    $11 -1
6534                        EXCH    $11 $10
6535                        ADD     $8 $3
6536                        ADDI    $8 2
6537                        EXCH    $9 $8
6538                        ADDI    $8 -2
6539                        SUB     $8 $3
6540                        EXCH    $3 $1
6541                        ADDI    $1 -1
6542                        EXCH    $7 $1
6543                        ADDI    $1 -1
6544                        EXCH    $6 $1
6545                        ADDI    $1 -1
6546                        EXCH    $10 $1
6547                        ADDI    $1 -1
6548                        ADDI    $13 -6548
6549 l_rjmp_top_299:        RBRA    l_rjmp_bot_300
6550 l_jmp_298:            SWAPBR  $13
6551                        NEG     $13
6552 l_rjmp_bot_300:        BRA     l_rjmp_top_299
6553                        ADDI    $13 6548
6554                        ADDI    $1 1
6555                        EXCH    $10 $1
6556                        ADDI    $1 1
6557                        EXCH    $6 $1
6558                        ADDI    $1 1
6559                        EXCH    $7 $1
6560                        ADDI    $1 1
6561                        EXCH    $3 $1
6562                        ADD     $8 $3
6563                        ADDI    $8 2
6564                        EXCH    $9 $8
6565                        ADDI    $8 -2
6566                        SUB     $8 $3
6567                        EXCH    $11 $10
6568                        ADDI    $11 1
6569                        EXCH    $12 $11
6570                        XOR     $13 $12
6571                        EXCH    $12 $11
6572                        ADDI    $11 -1
6573 loadMetAdd_297_i:      EXCH    $11 $10
6574                        XOR     $10 $9
6575                        ADD     $8 $3
6576                        ADDI    $8 2
6577                        EXCH    $9 $8
6578                        ADDI    $8 -2
6579                        SUB     $8 $3
6580                        ADDI    $1 1
6581                        EXCH    $8 $1
6582                        XOR     $8 $0
6583 localBlock_301_i:      XOR     $7 $1
6584                        ADDI    $1 1
6585                        EXCH    $7 $1
6586                        XOR     $7 $0
6587 localBlock_302_i:      XOR     $6 $1
6588 l_moveRight_7_bot:     BRA
        l_moveRight_7_top
6589 l_main_0_top:          BRA     l_main_0_bot
6590                        ADDI    $1 1
6591                        EXCH    $2 $1
6592                        EXCH    $3 $1
6593                        ADDI    $1 -1
6594 l_main_0:              SWAPBR  $2
6595                        NEG     $2
6596                        ADDI    $1 1
6597                        EXCH    $3 $1
6598                        EXCH    $2 $1
6599                        ADDI    $1 -1
6600                        EXCH    $3 $1
6601                        ADDI    $1 -1
6602 obj_con_303:           ADDI    $8 32
6603                        EXCH    $8 $1
6604                        ADDI    $1 -1
6605                        EXCH    $7 $1
6606                        ADDI    $1 -1
6607                        BRA     l_malloc
6608                        ADDI    $1 1
6609                        EXCH    $7 $1
6610                        ADDI    $1 1
6611                        EXCH    $8 $1
6612 obj_con_303_i:         ADDI    $8 -32
6613                        ADDI    $1 1
6614                        EXCH    $3 $1
6615                        ADD     $6 $3
6616                        ADDI    $6 2
6617                        XORI    $8 3
6618                        EXCH    $8 $7
6619                        ADDI    $7 1
6620                        XORI    $8 1
6621                        EXCH    $8 $7
6622 obj_con_303_bot:       ADDI    $7 -1
6623                        EXCH    $7 $6
6624                        ADDI    $6 -2
6625                        SUB     $6 $3
6626                        ADD     $6 $3
6627                        ADDI    $6 2
6628                        EXCH    $7 $6
6629                        ADDI    $6 -2
6630                        SUB     $6 $3
6631                        XOR     $8 $7
6632 loadMetAdd_304:        EXCH    $9 $8
6633                        ADDI    $9 3
6634                        EXCH    $10 $9
6635                        XOR     $11 $10
6636                        EXCH    $10 $9
6637                        ADDI    $9 -3
6638                        EXCH    $9 $8
6639                        ADD     $6 $3
6640                        ADDI    $6 2
6641                        EXCH    $7 $6
6642                        ADDI    $6 -2
6643                        SUB     $6 $3
6644                        EXCH    $3 $1
6645                        ADDI    $1 -1
6646                        EXCH    $8 $1
6647                        ADDI    $1 -1
6648                        ADDI    $11 -6647
6649 l_jmp_305:            SWAPBR  $11
6650                        NEG     $11
```

```
6651                        ADDI   $11 6647
6652                        ADDI   $1 1
6653                        EXCH   $8 $1
6654                        ADDI   $1 1
6655                        EXCH   $3 $1
6656                        ADD    $6 $3
6657                        ADDI   $6 2
6658                        EXCH   $7 $6
6659                        ADDI   $6 -2
6660                        SUB    $6 $3
6661                        EXCH   $9 $8
6662                        ADDI   $9 3
6663                        EXCH   $10 $9
6664                        XOR    $11 $10
6665                        EXCH   $10 $9
6666                        ADDI   $9 -3
6667 loadMetAdd_304_i:     EXCH   $9 $8
6668                        XOR    $8 $7
6669                        ADD    $6 $3
6670                        ADDI   $6 2
6671                        EXCH   $7 $6
6672                        ADDI   $6 -2
6673                        SUB    $6 $3
6674 l_main_0_bot:          BRA    l_main_0_top
6675 start:                 BRA    top
6676                        START
6677                        ADDI   $4 6715
6678                        XOR    $5 $4
6679                        ADDI   $5 10
6680                        XOR    $7 $5
6681                        ADDI   $4 10
6682                        ADDI   $4 -1
6683                        EXCH   $7 $4
6684                        ADDI   $4 1
6685                        ADDI   $4 -10
6686                        XOR    $1 $5
6687                        ADDI   $1 2048
6688                        ADDI   $1 -4
6689                        XOR    $3 $1
6690                        XORI   $6 2
6691                        EXCH   $6 $3
6692                        ADDI   $1 -1
6693                        EXCH   $3 $1
6694                        ADDI   $1 -1
6695                        BRA    l_main_0
6696                        ADDI   $1 1
6697                        EXCH   $3 $1
6698                        ADDI   $3 1
6699                        ADDI   $3 1
6700                        EXCH   $6 $3
6701                        XORI   $7 1
6702                        EXCH   $6 $7
6703                        XORI   $7 1
6704                        ADDI   $3 -1
6705                        ADDI   $3 -1
6706                        ADDI   $1 1
6707                        EXCH   $6 $3
6708                        XORI   $6 2
6709                        XOR    $3 $1
6710                        ADDI   $1 4
6711                        ADDI   $1 -2048
6712                        XOR    $1 $5
6713                        ADDI   $5 -10
6714                        XOR    $5 $4
6715                        ADDI   $4 -6715
6716 finish:                FINISH
```