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 25th, 2018

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 230 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 [23].

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 [13].

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) [25, 3] and the BOBISA architecture and instruction set [22] and high level languages JANUS [15, 30, 28] 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 [16].

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 [21]. 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.

```
class Fib
    int[] xs

    method init()
        new int[2] xs

    method fib(int n)
        if n = 0 then
            xs[0] ^= 1
            xs[1] ^= 1
        else
            n -= 1
            call fib(n)
            xs[0] += xs[1]
            xs[0] <=> xs[1]
        fi xs[0] = xs[1]

    method get(int out)
        out ^= xs[1]

class Program
    int result
    int n

    method main()
        n ^= 4

        new Fib f
        call f::init()
        call f::fib(n)
        call f::get(result)
        uncall f::fib(n)
        uncall f::init()
        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

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

Syntax Domains

$$
\begin{array}{rcl}
prog \in \text{Programs} & s \in \text{Statements} & n \in \text{Constants} \\
cl \in \text{Classes} & e \in \text{Expressions} & x \in \text{VarIDs} \\
t \in \text{Types} & \odot \in \text{ModOps} & q \in \text{MethodIDs} \\
m \in \text{Methods} & \otimes \in \text{Operators} & c \in \text{ClassIDs}
\end{array}
$$

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

$$\textbf{new}\ c\ x \quad \dots \quad \textbf{delete}\ c\ x$$

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 [30].

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.

```
new int[5] intArray          // Init new integer array
new Foo[2] fooArray          // init new Foo array

intArray[1] += 10            // Legal array integer assignment
intArray[1] -= 10            // Legal Zero-clearing for integer array cells

new Foo fooObject
fooArray[0] <=> fooObject    // Legal object array cell assignment
new Foo fooArray[2]          // Legal object array cell assignment

...                          // Clear all array cells

delete Foo fooArray[0]       // Legal object array cell zero-clearing
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 [28] 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.

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

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.

```
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) // Recursively search until we reach end of list
        else skip
        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

```
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 until we hit end of list
        else skip
        fi head != nil

        listLength += 1                 // Increment length

    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
        else skip
        fi cell != nil & head = nil

        if cell != nil & head = nil then
            cell <=> head               // Set head = cell. Cell is nil after execution
        else skip
        fi cell = nil & head != nil

        listLength += 1                 // Increment length

    method length(int result)
        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.

```
class Tree
    Node root

    method insertNode(Node node, int value)
        if root = nil & node != nil then
            root <=> node
        else skip
        fi root != nil & node = nil

        if root != nil then
            call root::insertNode(node, value)
        else skip
        fi root != nil

    method sum(int result)
        if root != nil then
            call root::getSum(result)
        else skip
        fi root != nil

    method mirror()
        if root != nil then
            call root::mirror()
        else skip
        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.

```
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
```

Figure 2.7: Binary Tree node class (cont)

```
class Node
    Node left
    Node right
    int value

    method setValue(int newValue)
        value ^= newValue

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

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

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

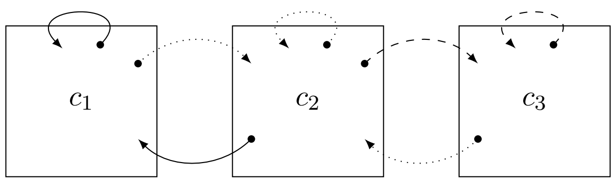


Figure 2.9: Multiple identical reference are needed for a doubly linked list implementation

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.

```
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
```

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.

```
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
```

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

```
    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 as left of  right neighbour
            delocal Cell selfCopy = nil

            local int cellIndex = index + 1
            call right::setIndex(cellIndex) // Set index in right neighbour of current
            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
```

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 [27]. 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].

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

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

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

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

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

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

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

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

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.

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

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.

$$\frac{}{\Pi \vdash_{expr} \ n : \textbf{int}}\ \text{T-C}\textsc{on} \qquad \frac{\Pi(x) = \tau}{\Pi \vdash_{expr} \ x : \tau}\ \text{T-V}\textsc{ar} \qquad \frac{\tau \neq \textbf{int}}{\Pi \vdash_{expr} \ \textbf{nil} \ : \tau}\ \text{T-N}\textsc{il}$$

$$\frac{\Pi \vdash_{expr} \ e_1 : \textbf{int} \qquad \Pi \vdash_{expr} \ e_2 : \textbf{int}}{\Pi \vdash_{expr} \ e_1 \otimes e_2 : \textbf{int}}\ \text{T-B}\textsc{in}\text{O}\textsc{p}\text{I}\textsc{nt}$$

$$\frac{\Pi \vdash_{expr} \ e_1 : c \qquad \Pi \vdash_{expr} \ e_2 : c \qquad \otimes \in \{\texttt{=}, \texttt{!=}\}}{\Pi \vdash_{expr} \ e_1 \otimes e_2 : \textbf{int}}\ \text{T-B}\textsc{in}\text{O}\textsc{p}\text{O}\textsc{bj}$$

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.

$$\frac{\Pi(x) = \tau[\,] \qquad \Pi_{expr} \vdash \ e \ : \ \textbf{int}}{\Pi \vdash_{expr} \ x[e] : \tau}\ \text{T-A}\textsc{rr}\text{E}\textsc{lem}\text{V}\textsc{ar}$$

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.

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

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

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

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

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

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

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

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

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.

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

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

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

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

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

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

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.

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

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

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

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.

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

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.

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

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

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.

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

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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

presented in [28]. 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

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

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

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

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

$$\frac{\langle l, \gamma \rangle \vdash^{\Gamma}_{stmt} \ \textbf{new} \ a[e] \ x \ : \ \mu' \rightleftharpoons \mu}{\gamma \vdash^{\Gamma}_{stmt} \ \textbf{delete} \ a[e] \ x \ : \ \mu \rightleftharpoons \mu'} \ \textsc{ArrDelete}$$

$$\frac{\gamma(x) = l \qquad \gamma(x') = l' \qquad \mu(l) = v}{\gamma \vdash^{\Gamma}_{stmt} \ \textbf{copy} \ c \ x \ x' \ : \ \mu[l' \mapsto 0] \leftrightharpoons \mu[l' \mapsto v]} \ \textsc{Copy}$$

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

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

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

uncall statement, similarly to techniques presented in [30, 28].

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.

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

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{array}{ll}
\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{array}
$$

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 [30, 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.

$$\begin{array}{ll} \mathcal{I}'[\![\mathbf{call}\ q(\ldots)]\!] \ = \mathbf{call}\ q(\ldots) & \mathcal{I}'[\![\mathbf{call}\ x :: q(\ldots)]\!] \ = \mathbf{call}\ x :: q(\ldots) \\ \mathcal{I}'[\![\mathbf{uncall}\ q(\ldots)]\!] \ = \mathbf{uncall}\ q(\ldots) & \mathcal{I}'[\![\mathbf{uncall}\ x :: q(\ldots)]\!] \ = \mathbf{uncall}\ x :: q(\ldots) \end{array}$$

$$\mathcal{I}'[\![s]\!] = \mathcal{I}[\![s]\!]$$

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 $\mathcal{T} =$

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

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 $\mathcal{T}'$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 [28]. 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 [28].

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*

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

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

A RTM simulation implemented in ROOPL by representing the set of states $\{q_1,\ \dots,\ q_n\}$ 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 [28].

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.

```
method moveRight(int symbol, Cell tapeHead)
    local Cell right = nil
    local Cell tmp = nil
    uncall tapeHead::getSymbol(symbol)   // Put symbol back in current cell
    call tapeHead::getRight(right)       // Get right neighbour

    if right = nil && symbol = BLANK then
        symbol ^= BLANK                  // Zero clear symbol
        new Cell right                   // Init new neighbour
        copy Cell right tmp              // Copy reference to self
        uncall right::getSelf(tmp)       // Store self reference
        uncall right::getLeft(tapeHead)  // Set tape head as left of new cell
        right <=> tapeHead
    else
        call right::getLeft(tmp)         // Get copy of tape head reference
        uncopy Cell tmp tapeHead         // Clear reference to tape head

        if tapeHead = nil && symbol = BLANK
            call tmp::getSelf(tapeHead) // rev: set self pointer
            uncopy Cell tmp tapeHead    // rev: new self pointer
            delete Cell tmp             // rev: new left neighbour
            symbol ^= BLANK
        else skip                       // In reverse:
        fi tmp = nil                    // Allocate new left if current is nil

        uncall right::getLeft(tmp)       // Put tape head reference back
        tapeHead <=> right
        call tapeHead::getRight(right)   // Get right of new tape head
        call tapeHead::getSymbol(symbol) // Get symbol of new tape head
    fi right = nil
    uncall tapeHead::getRight(right)    // Set right neighbour
    delocal Cell right = nil
    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 [28], 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, 29, 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 [17].

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.

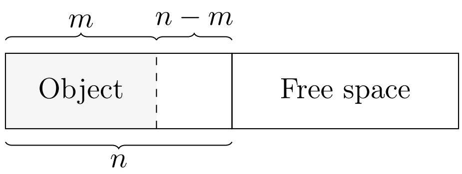


Figure 3.1a: Creation of internal fragmentation of size $n - m$ due to *over-allocation*

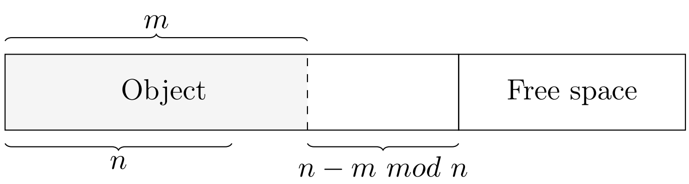


Figure 3.1b: Creation of internal fragmentation of size $n - m \; mod \; n$ due to *over-allocation*

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.

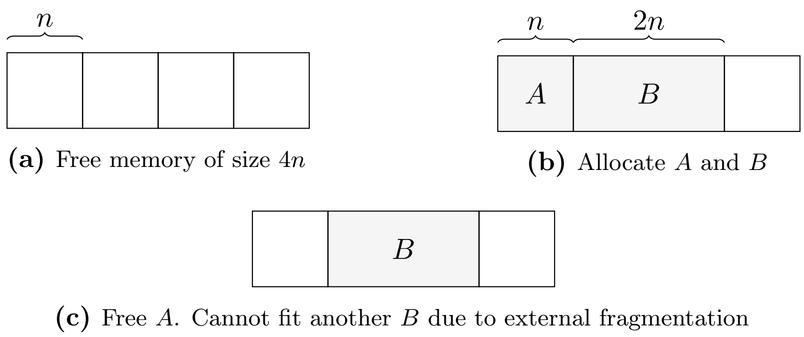
3.1.2 External Fragmentation

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.



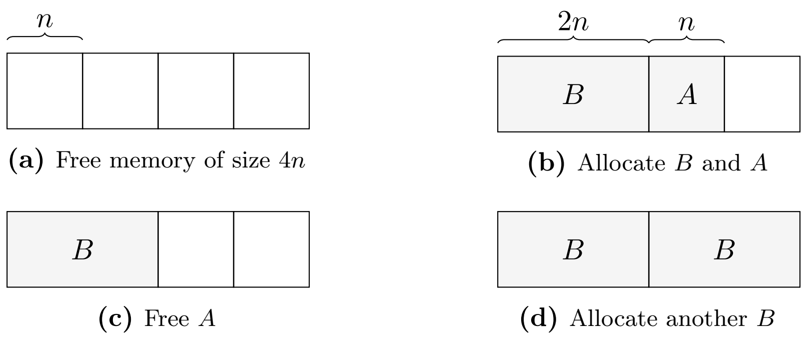
(a) Free memory of size $4n$

(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.

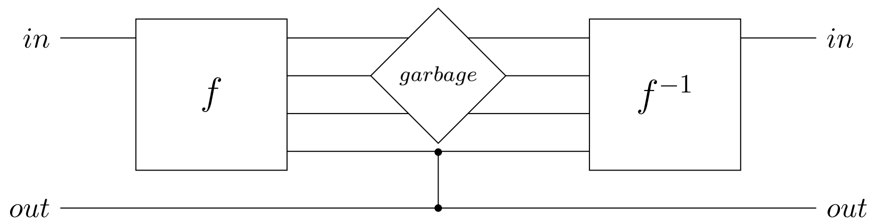


Figure 3.4: The "garbage" output of an injective function f is the input to its inverse function f^{-1}

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.

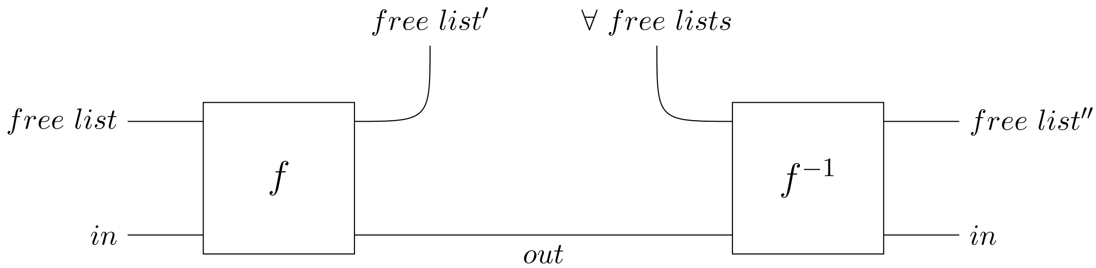


Figure 3.5: All free lists are considered equivalent "garbage" in terms of injective functions

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 $garbage_f$ (e.g. garbage in form of storing data in the heap, so the free list changes, the heap grows, etc.). Its inverse function f^{-1} must thus take the $output_f$ of f and $garbage_f$ as $input_{f^{-1}}$ to produce its output $output_{f^{-1}}$ which is the $input_f$ of f. However, in the context of reversible heaps, we must consider all free lists as of "equivalent 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 its inverse f^{-1}.

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) [14, 24, 20]. 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 46 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.

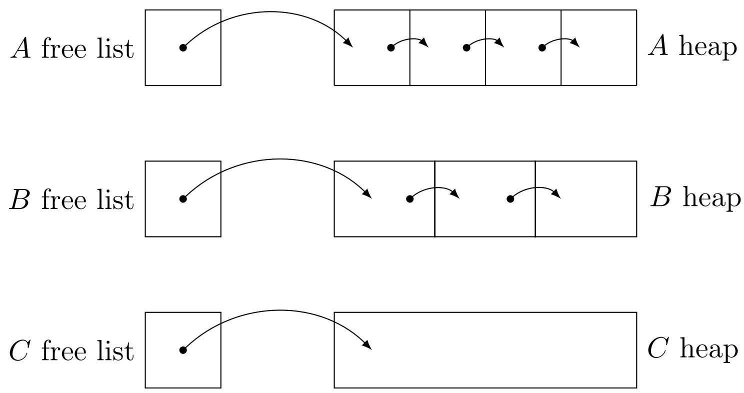


Figure 3.6: Memory layout using one heap per record size

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 [28]. 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.

```
procedure malloc(int p, int osize, int freelists[])
  local int counter = 0
  local int csize = 2
  call malloc1(p, osize, freelists, counter, csize)
  delocal int csize = 2
  delocal int counter = 0

procedure malloc1(int p, int osize, int freelists[], int counter, int csize)
  if (csize < osize) then
    counter += 1
    csize += 1
    call malloc1(p, osize, freelists, counter, csize)
    csize -= 1
    counter -= 1
  else
    p += freelists[counter]
    freelists[counter] -= p

    // Swap head of free list with next block of p
    freelists[counter] ^= M(p)
    M(p) ^= freelists[counter]
    freelists[counter] ^= M(p)
  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.

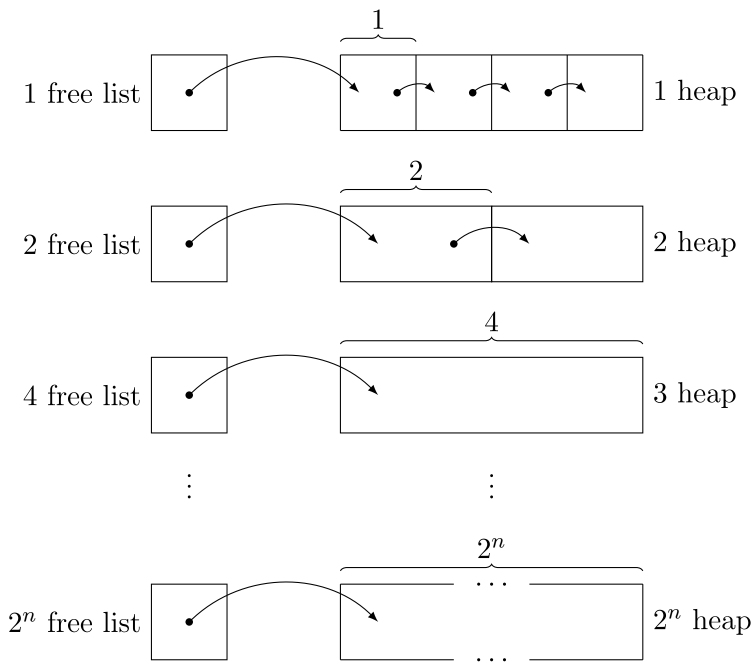


Figure 3.7: Memory layout using one heap per power-of-two

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.

```
procedure double(int target)
  local int current = target
  target += current
  delocal int current = target / 2

procedure malloc1(int p, int osize, int freelists[], int counter, int csize)
  if (csize < osize) then
      counter += 1
      call double(csize)
      call malloc1(p, osize, freelists, counter, csize)
      uncall double(csize)
      counter -= 1
  else
    if freelists[counter] != 0 then
      p += freelists[counter]
      freelists[counter] -= p

      // Swap head of free list with next block of p
      freelists[counter] ^= M(p)
      M(p) ^= freelists[counter]
      freelists[counter] ^= M(p)
    else
      counter += 1
      call double(csize)
      call malloc1(p, osize, freelists, counter, csize)
      uncall double(csize)
      counter -= 1
    fi freelists[counter] = 0 || p != freelists[counter]
  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.

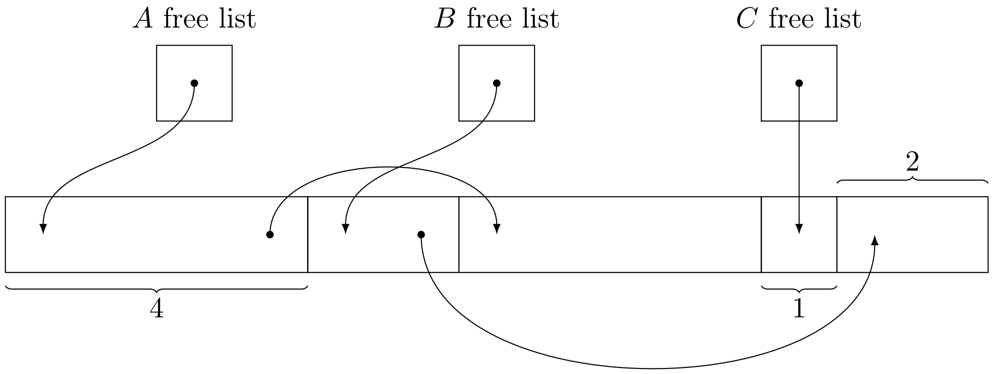
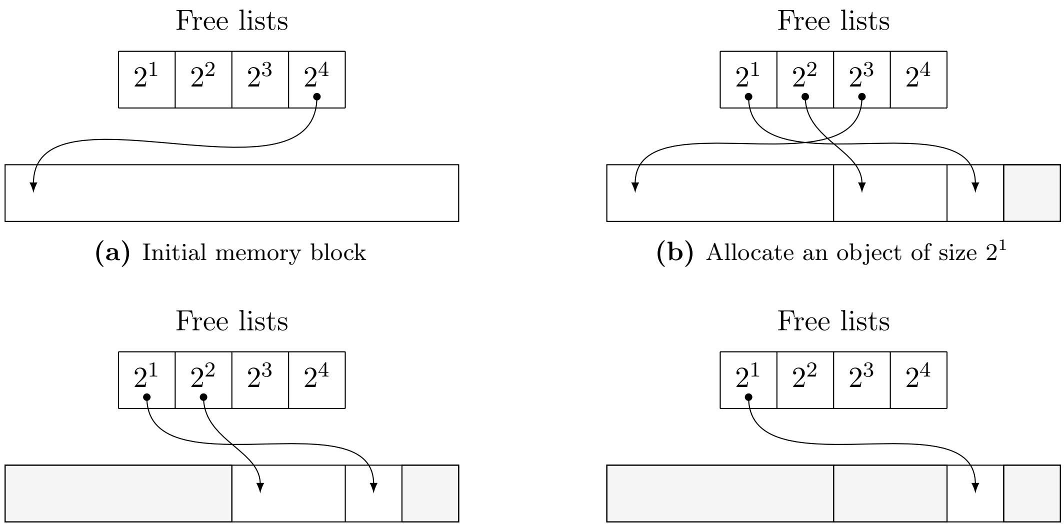


Figure 3.8: Record size-specific free lists on a shared heap (powers of two)

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.



(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.

```
procedure malloc1(int p, int osize, int freelists[], int counter, int csize)
  if (csize < osize) then
      counter += 1
      call double(csize)
      call malloc1(p, osize, freelists, counter, csize)
      uncall double(csize)
      counter -= 1
  else
      if freelists[counter] != 0 then
          p += freelists[counter]
          freelists[counter] -= p

          // Swap head of free list with next block of p
          freelists[counter] ^= M(p)
          M(p) ^= freelists[counter]
          freelists[counter] ^= M(p)
      else
          counter += 1
          call double(csize)
          call malloc1(p, osize, freelists, counter, csize)
          uncall double(csize)
          counter -= 1
          freelists[counter] += p
          p += csize
      fi freelists[counter] = 0 || p - csize != freelists[counter]
  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 [28]. 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 52. 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`.

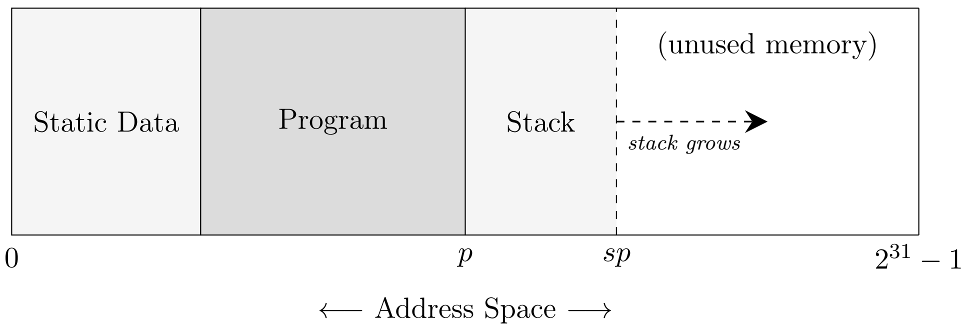


Figure 4.1: Memory layout of a ROOPL program, originally from [11]

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.

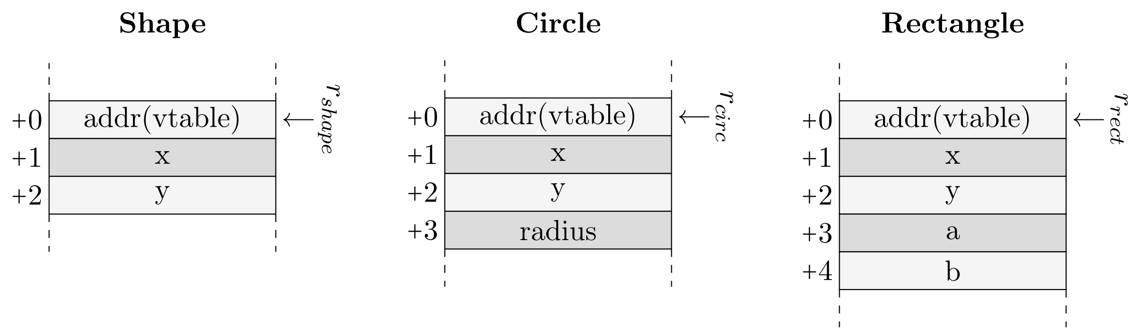


Figure 4.2: Illustration of prefixing in the memory layout of 3 ROOPL objects, originally from [11]

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.

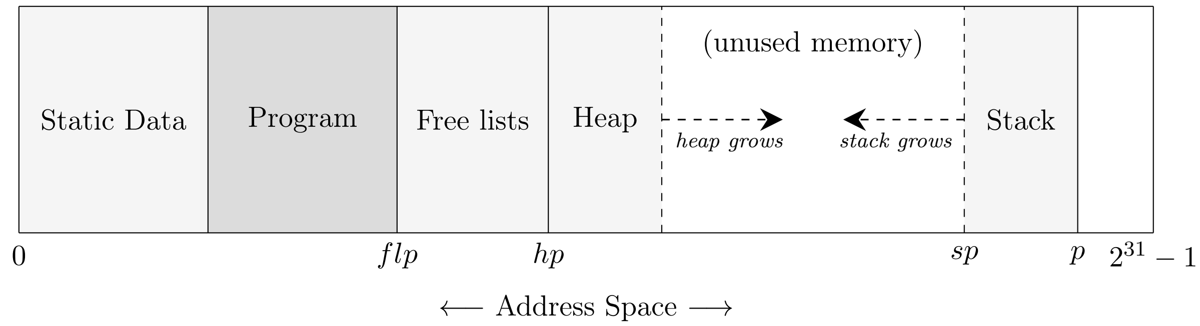


Figure 4.3: Memory layout of a ROOPL++ program

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++.

$$
\begin{aligned}
\textbf{SUBI}\quad r \quad i \quad &\stackrel{\textbf{def}}{==} \quad \textbf{ADDI}\quad r \quad -i \\
\textbf{PUSH}\quad r \quad &\stackrel{\textbf{def}}{==} \quad \left[\textbf{EXCH}\quad r \quad r_{sp}\ ,\quad \textbf{SUBI}\quad r_{sp} \quad 1\right] \\
\textbf{POP}\quad r \quad &\stackrel{\textbf{def}}{==} \quad \left[\textbf{ADDI}\quad r_{sp} \quad 1\ ,\quad \textbf{EXCH}\quad r \quad r_{sp}\right]
\end{aligned}
$$

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.

	······			; Static data declarations
	······			; Code for program class methods
start :	**START**			; Program starting point
	ADDI	r_{flps}	p	; Initialize free lists pointer
	XOR	r_{hp}	r_{flps}	; Initialize heap pointer
	ADDI	r_{hp}	$size_{fls}$	; Initialize heap pointer
	XOR	r_b	r_{hp}	; Store address of initial free memory block in r_b
	ADDI	r_{flps}	$size_{fls}$	; Index to end of free lists
	SUBI	r_{flps}	1	; Index to last element of free lists
	EXCH	rb	r_{flps}	; Store address of first block in last element of free lists
	ADDI	r_{flps}	1	; Index to end of free lists
	SUBI	r_{flps}	s	; Index to beginning of free lists
	XOR	r_{sp}	r_{hp}	; Initialize stack pointer
	ADDI	r_{sp}	$offset_{stack}$	; Initialize stack pointer
	SUBI	r_{sp}	$size_m$	; Allocate space for main object
	XOR	r_m	r_{sp}	; Store address of main object in r_m
	XORI	r_v	$label_{vt}$	; Store address of vtable in r_v
	PUSH	r_v		; Push address of vtable onto stack
	PUSH	r_m		; Push '*this*' onto stack
	BRA	$label_m$		; Call main procedure
	POP	r_m		; Pop '*this*' from stack
	POP	r_v		; Pop vtable address into r_v
	XORI	r_v	$label_{vt}$	; Clear r_v
	XOR	r_m	r_{sp}	; Clear r_m
	ADDI	r_{sp}	$size_m$	; Deallocate space of main object
	SUBI	r_{sp}	$offset_{stack}$	; Clear stack pointer
	XOR	r_{sp}	r_{hp}	; Clear stack pointer
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.

```
if (csize < osize) then
    // outer if-then
else
    if freelists[counter] != 0 then
        // inner if-then
    else
        // inner else
    fi (freelists[counter] = 0 ||
        p - csize != freelists[counter])
fi csize < osize
```

(7)		······	; Code for $r_{fl} \leftarrow addr(fl[c])$
(8)		······	; Code for $r_{block} \leftarrow [\![fl[c]]\!]$
(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
(14-21)		······	; 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 $r_{e1_i} \leftarrow [\![addr(fl[c]) \neq 0]\!]$
(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}
(30-34)		······	; 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)		······	; $r_{e2_{i1}} \leftarrow [\![p - c_{size} \neq addr(fl[c])]\!]$
(52)		······	; $r_{e2_{i2}} \leftarrow [\![addr(fl[c]) = 0]\!]$
(53)		······	; $r_{e2_{i3}} \leftarrow [\![(p - c_{size} \neq addr(fl[c])) \vee (addr(fl[c]) = 0)]\!]$
(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)**

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 55 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.

```
counter += 1
call double(csize)
call malloc1(p, osize, freelists,
             counter, csize)
uncall double(csize)
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 *malloc*1())
(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.

```
p += freelists[counter]
freelists[counter] -= p

// Swap head of free list
// with p's next block
freelists[counter] ^= M(p)
M(p) ^= freelists[counter]
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.

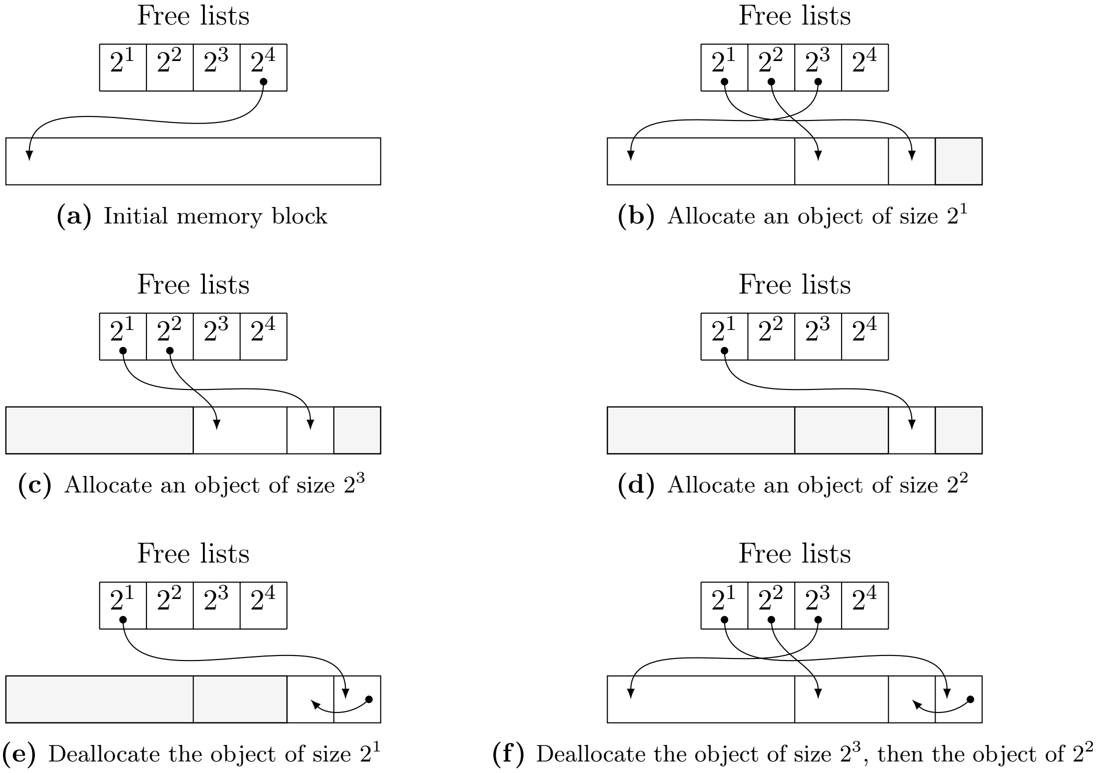
```
counter += 1
call double(csize)
call malloc1(p, osize, freelists,
             counter, csize)
uncall double(csize)
counter -= 1
freelists[counter] += p
p += csize
```

(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 *malloc*1())
(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

(e) Deallocate the object of size 2^1

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

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 55.

```
procedure malloc(int p, int osize,
                 int freelists[])
    local int counter = 0
    local int csize = 2
    call malloc1(p, osize, freelists,
                 counter, csize)
    delocal int csize = 2
    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)		$\cdots$			; 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)		$\cdots$	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)	$\cdots$			; Push registers
(2)	$\cdots$			; Code for $r_t \leftarrow x_{size}$
(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)	$\cdots$			; Inverse of **(2)**
(9)	$\cdots$			; Inverse of **(1)**
(10)	$\cdots$			; Code for $r_v \leftarrow [\![addr(x)]\!]$
(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)	$\cdots$			; Inverse of **(10)**

delete $c\ x$

(1)	$\cdots$			; Code for $r_p \leftarrow [\![addr(x)]\!]$
(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)	$\cdots$			; Push registers except r_p, r_t
(9)	$\cdots$			; Code for $r_t \leftarrow x_{size}$
(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)	$\cdots$			; Inverse of **(9)**
(16)	$\cdots$			; Inverse of **(8)**
(17)	$\cdots$			; 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 $c\ x \quad s$ **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)	$\cdots\cdots$			; Code for **new** $c\ x$
(4)	$\cdots\cdots$			; Code for statement s
(5)	$\cdots\cdots$			; 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 58.

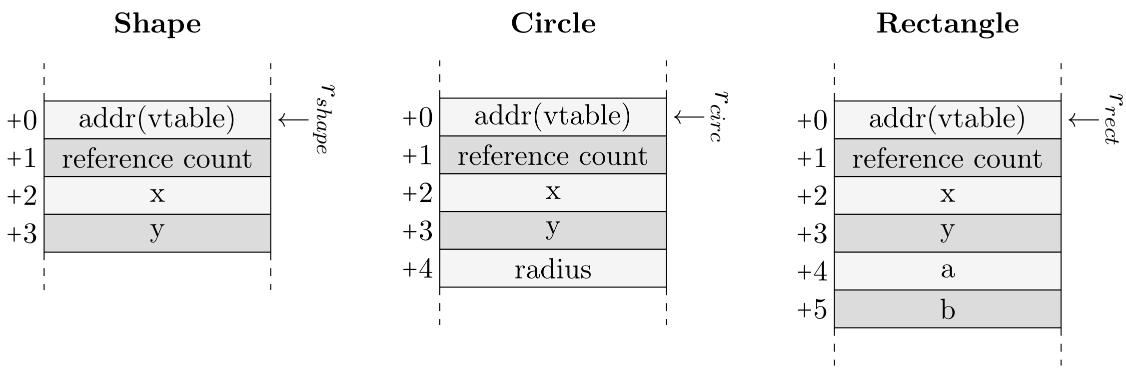


Figure 4.15: Illustration of prefixing in the memory layout of three Roopl++ objects

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 $r_p \leftarrow addr(x)$
(2)	...			; Code for $r_{cp} \leftarrow value(x')$
(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 $r_p \leftarrow addr(x)$
(2)	...			; Code for $r_{cp} \leftarrow value(x')$
(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.

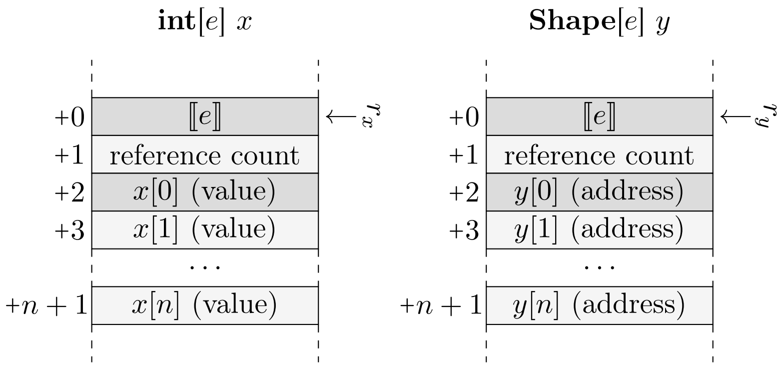


Figure 4.17: Illustration of prefixing in the memory layout of two ROOPL++ arrays

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 $a[e]\ x$

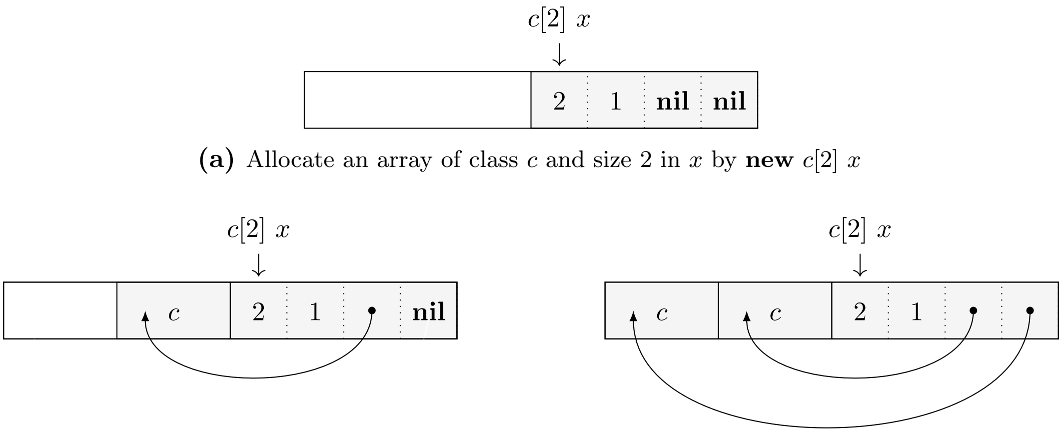
(1)	$\cdots$			; Push registers
(2)	$\cdots$			; Code for $r_t \leftarrow [\![e]\!] + 2$
(3)	**PUSH**	r_t		; Push r_t
(4)	**PUSH**	r_p		; Push r_p
(5)	**BRA**	l_{malloc}		; Allocate array
(6)	**POP**	r_p		; Inverse of **(4)**
(7)	**POP**	r_t		; Inverse of **(3)**
(9)	$\cdots$			; Inverse of **(1)**
(10)	$\cdots$			; Code for $r_v \leftarrow [\![addr(x)]\!]$
(11)	**SUBI**	r_t	2	; $r_t \leftarrow [\![e]\!]$
(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)	$\cdots$			; Inverse of **(10)**

delete $a[e]\ x$

(1)	$\cdots$			; Code for $r_p \leftarrow [\![addr(x)]\!]$
(2)	$\cdots$			; Code for $r_v \leftarrow [\![e]\!]$
(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)	$\cdots$			; 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)	$\cdots$			; Inverse of **(9)**
(18)	$\cdots$			; Inverse of **(2)**
(19)	$\cdots$			; 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 65. 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 [28] 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] Landauer, R. “Irreversibility and Heat Generation in the Computing Process”. In: *IBM Journal of Research and Development* 5.3 (1961), pp. 183–191.

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

[15] Lutz, C. “Janus: a time-reversible language”. Letter to R. Landauer. 1986.

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

[17] Mogensen, T. Æ. “Programming Language Design and Implementation (Unpublished)”.

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

[21] 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.

[22] 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.

[23] 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).

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

[25] Vieri, C. J. “Pendulum: A Reversible Computer Architecture”. Master’s Thesis. University of California at Berkeley, 1993.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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)		$\cdots\cdots$				; Code for $r_{e2_{i1}} \leftarrow [\![p - c_{size} \neq addr(freelists[counter])]\!]$
(52)		$\cdots\cdots$				; Code for $r_{e2_{i2}} \leftarrow [\![addr(freelists[counter]) = 0]\!]$
(53)		$\cdots\cdots$				; 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)		$\cdots\cdots$				; Inverse of **(53)**
(56)		$\cdots\cdots$				; Inverse of **(52)**
(57)		$\cdots\cdots$				; 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)		$\cdots\cdots$				; 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)		$\cdots\cdots$				; 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

```haskell
module AST where

import Text.Show.Pretty

{-- AST Primitives --}
type TypeName = String

type MethodName = String

data DataType = IntegerType
              | ObjectType TypeName
              | CopyType TypeName
              |Â ObjectArrayType TypeName
              | IntegerArrayType
              | ArrayType
              | ArrayElementType
              | NilType
  deriving (Show)

-- Types
instance Eq DataType where
  IntegerType == IntegerType = True
  IntegerArrayType == IntegerArrayType = True
  NilType == NilType = True
  NilType == (ObjectType _) = True
  (ObjectType _) == NilType = True
  (ObjectType t1) == (ObjectType t2) = t1 == t2
  (CopyType t1) == (CopyType t2) = t1 == t2
  (ObjectArrayType t1) == (ObjectArrayType t2) = t1 == t2
  (CopyType t1) == (ObjectType t2) = t1 == t2
  (ObjectType t1) == (CopyType t2) = t1 == t2
  ArrayType == (ObjectArrayType _) = True
  (ObjectArrayType _) == ArrayType = True
  ArrayType == IntegerArrayType = True
  IntegerArrayType == ArrayType = True
  _ == _ = False

-- Binary Operators
data BinOp = Add
           | Sub
           | Xor
           | Mul
           | Div
           | Mod
           | BitAnd
           | BitOr
           | And
           | Or
           | Lt
           | Gt
           | Eq
           | Neq
           | Lte
           | Gte
  deriving (Show, Eq, Enum)

data ModOp = ModAdd
           | ModSub
           | ModXor
  deriving (Show, Eq, Enum)

{-- Generic AST Definitions --}
--Expressions
```

```haskell
data GExpr v = Constant Integer
             | Variable v
             | ArrayElement (v, GExpr v)
             | Nil
             | Binary BinOp (GExpr v) (GExpr v)
  deriving (Show, Eq)

--Statements
data GStmt m v = Assign v ModOp (GExpr v)
               |Â AssignArrElem (v, GExpr v) ModOp (GExpr v)
               | Swap (v, Maybe (GExpr v)) (v, Maybe (GExpr v))
               | Conditional (GExpr v) [GStmt m v] [GStmt m v] (GExpr v)
               | Loop (GExpr v) [GStmt m v] [GStmt m v] (GExpr v)
               | ObjectBlock TypeName v [GStmt m v]
               | LocalBlock DataType v (GExpr v) [GStmt m v] (GExpr v)
               | LocalCall m [(v, Maybe (GExpr v))]
               | LocalUncall m [(v, Maybe (GExpr v))]
               | ObjectCall (v, Maybe (GExpr v)) MethodName [(v, Maybe (GExpr v))]
               | ObjectUncall (v, Maybe (GExpr v)) MethodName [(v, Maybe (GExpr v))]
               | ObjectConstruction TypeName (v, Maybe (GExpr v))
               | ObjectDestruction TypeName (v, Maybe (GExpr v))
               | CopyReference DataType (v, Maybe (GExpr v)) (v, Maybe (GExpr v))
               | UnCopyReference DataType (v, Maybe (GExpr v)) (v, Maybe (GExpr v))
               | ArrayConstruction (TypeName, GExpr v) v
               | ArrayDestruction (TypeName, GExpr v) v
               | Skip
  deriving (Show, Eq)

--Field/Parameter declarations
data GDecl v = GDecl DataType v
  deriving (Show, Eq)

--Method: Name, parameters, body
data GMDecl m v = GMDecl m [GDecl v] [GStmt m v]
  deriving (Show, Eq)

--Class: Name, fields, methods
data GCDecl m v = GCDecl TypeName (Maybe TypeName) [GDecl v] [GMDecl m v]
  deriving (Show, Eq)

--Program
newtype GProg m v = GProg [GCDecl m v]
  deriving (Show, Eq)

{-- Specific AST Definitions --}
--Plain AST
type Identifier = String

type Expression = GExpr Identifier

type Statement = GStmt MethodName Identifier

type VariableDeclaration = GDecl Identifier

type MethodDeclaration = GMDecl MethodName Identifier

type ClassDeclaration = GCDecl MethodName Identifier

type Program = GProg MethodName Identifier

--Scoped AST
type SIdentifier = Integer

type SExpression = GExpr SIdentifier

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

```haskell
{-# 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
```

```haskell
           | StackOffset
           | InitialMemoryBlockSize
           | ReferenceCounterIndex
           | ArrayElementOffset
    deriving (Show, Eq)

type MInstruction = GInstr Macro
type MProgram = GProg Macro

invertInstructions :: [(Maybe Label, MInstruction)] -> [(Maybe Label, MInstruction)]
invertInstructions = reverse . map (second invertInstruction . first (fmap (++ "_i")))
    where invertInstruction (ADD r1 r2) = SUB r1 r2
          invertInstruction (SUB r1 r2) = ADD r1 r2
          invertInstruction (ADDI r i) = SUBI r i
          invertInstruction (SUBI r i) = ADDI r i
          invertInstruction (RL r i) = RR r i
          invertInstruction (RLV r1 r2) = RRV r1 r2
          invertInstruction (RR r i) = RL r i
          invertInstruction (RRV r1 r2) = RLV r1 r2
          invertInstruction (BEQ r1 r2 l) = BEQ r1 r2 $ l ++ "_i"
          invertInstruction (BGEZ r l) = BGEZ r $ l ++ "_i"
          invertInstruction (BGTZ r l) = BGTZ r $ l ++ "_i"
          invertInstruction (BLEZ r l) = BLEZ r $ l ++ "_i"
          invertInstruction (BLTZ r l) = BLTZ r $ l ++ "_i"
          invertInstruction (BNE r1 r2 l) = BNE r1 r2 $ l ++ "_i"
          invertInstruction (BRA l) = BRA $ l ++ "_i"
          invertInstruction (RBRA l) = RBRA $ l ++ "_i"
          invertInstruction inst = inst

{-- Output PISA Definitions --}

type Instruction = GInstr Integer
type Program = GProg Integer

instance Show Register where
    show (Reg r) = "$" ++ show r

instance Show Instruction where
    show (ADD r1 r2) = unwords ["ADD   ", show r1, show r2]
    show (ADDI r i) = unwords ["ADDI  ", show r, show i]
    show (ANDX r1 r2 r3) = unwords ["ANDX  ", show r1, show r2, show r3]
    show (ANDIX r1 r2 i) = unwords ["ANDIX ", show r1, show r2, show i]
    show (NORX r1 r2 r3) = unwords ["NORX  ", show r1, show r2, show r3]
    show (NEG r) = unwords ["NEG   ", show r]
    show (ORX r1 r2 r3) = unwords ["ORX   ", show r1, show r2, show r3]
    show (ORIX r1 r2 i) = unwords ["ORIX  ", show r1, show r2, show i]
    show (RL r i) = unwords ["RL    ", show r, show i]
    show (RLV r1 r2) = unwords ["RLV   ", show r1, show r2]
    show (RR r i) = unwords ["RR    ", show r, show i]
    show (RRV r1 r2) = unwords ["RRV   ", show r1, show r2]
    show (SLLX r1 r2 i) = unwords ["SLLX  ", show r1, show r2, show i]
    show (SLLVX r1 r2 r3) = unwords ["SLLVX ", show r1, show r2, show r3]
    show (SRAX r1 r2 i) = unwords ["SRAX  ", show r1, show r2, show i]
    show (SRAVX r1 r2 r3) = unwords ["SRAVX ", show r1, show r2, show r3]
    show (SRLX r1 r2 i) = unwords ["SRLX  ", show r1, show r2, show i]
    show (SRLVX r1 r2 r3) = unwords ["SRLVX ", show r1, show r2, show r3]
    show (SUB r1 r2) = unwords ["SUB   ", show r1, show r2]
    show (XOR r1 r2) = unwords ["XOR   ", show r1, show r2]
    show (XORI r i) = unwords ["XORI  ", show r, show i]
    show (BEQ r1 r2 l) = unwords ["BEQ   ", show r1, show r2, l]
    show (BGEZ r l) = unwords ["BGEZ  ", show r, l]
    show (BGTZ r l) = unwords ["BGTZ  ", show r, l]
    show (BLEZ r l) = unwords ["BLEZ  ", show r, l]
    show (BLTZ r l) = unwords ["BLTZ  ", show r, l]
    show (BNE r1 r2 l) = unwords ["BNE   ", show r1, show r2, l]
    show (BRA l) = unwords ["BRA   ", l]
```

```
    show (EXCH r1 r2) = unwords ["EXCH  ", show r1, show r2]
    show (SWAPBR r) = unwords ["SWAPBR", show r]
    show (RBRA l) = unwords ["RBRA  ", l]
    show START = "START "
    show FINISH = "FINISH"
    show (DATA i) = unwords ["DATA  ", show i]
    show (SUBI r i) = unwords ["ADDI  ", show r, show $ -i] --Expand pseudo

showProgram :: Program -> String
showProgram (GProg p) = ";; pendulum pal file\n" ++ intercalate "\n" (map showLine p)
    where showLine (Nothing, i) = spaces 25 ++ show i
          showLine (Just l, i) = l ++ ":" ++ spaces (24 - length l) ++ show i
          spaces :: (Int -> String)
          spaces n = [1..n] >> " "

writeProgram :: String -> Program -> IO ()
writeProgram output p = writeFile output $ showProgram p
```

Parser.hs

```haskell
module Parser (parseString) where

import Control.Monad.Except
import Data.Functor.Identity
import Data.Bifunctor

import Text.Parsec
import Text.Parsec.String
import Text.Parsec.Expr
import Text.Parsec.Language
import qualified Text.Parsec.Token as Token

import Debug.Trace (trace, traceShow)

import AST

{-- Language Definition --}
keywords :: [String]
keywords =
    ["class",
     "inherits",
     "method",
     "call",
     "uncall",
     "construct",
     "destruct",
     "skip",
     "from",
     "do",
     "loop",
     "until",
     "int",
     "nil",
     "if",
     "then",
     "else",
     "fi",
     "local",
     "delocal",
     "new",
     "delete",
     "copy",
     "uncopy"]

--Operator precedence identical to C
operatorTable :: [[(String, BinOp)]]
operatorTable =
    [ [("*", Mul), ("/", Div), ("%", Mod)],
      [("+", Add), ("-", Sub)],
      [("<", Lt), ("<=", Lte), (">", Gt), (">=", Gte)],
      [("=", Eq), ("!=", Neq)],
      [("&", BitAnd)],
      [("^", Xor)],
      [("|", BitOr)],
      [("&&", And)],
      [("||", Or)] ]

languageDef :: Token.LanguageDef st
languageDef =
    emptyDef {
        Token.commentLine     = "//",
        Token.nestedComments  = False,
        Token.identStart      = letter,
```

```
        Token.identLetter     = alphaNum <|> oneOf "_'",
        Token.reservedOpNames = concatMap (map fst) operatorTable,
        Token.reservedNames   = keywords,
        Token.caseSensitive   = True }

tokenParser :: Token.TokenParser st
tokenParser = Token.makeTokenParser languageDef

{-- Parser Primitives --}
identifier :: Parser String
identifier = Token.identifier tokenParser

arrElemIdentifier :: Parser (String, Expression)
arrElemIdentifier = do x <- identifier
                       y <- brackets expression
                       return (x, y)

anyIdentifier :: Parser (String, Maybe Expression)
anyIdentifier = do x <- identifier
                   y <- optionMaybe $ brackets expression
                   return (x, y)

reserved :: String -> Parser ()
reserved = Token.reserved tokenParser

reservedOp :: String -> Parser ()
reservedOp = Token.reservedOp tokenParser

integer :: Parser Integer
integer = Token.integer tokenParser

symbol :: String -> Parser String
symbol = Token.symbol tokenParser

parens :: Parser a -> Parser a
parens = Token.parens tokenParser

brackets :: Parser a -> Parser a
brackets = Token.brackets tokenParser

colon :: Parser String
colon = Token.colon tokenParser

commaSep :: Parser a -> Parser [a]
commaSep = Token.commaSep tokenParser

typeName :: Parser TypeName
typeName = identifier

arrayTypeName :: Parser (TypeName, Expression)
arrayTypeName = do x <- try typeName <|> string "int"
                   y <- brackets expression
                   return (x, y)

methodName :: Parser MethodName
methodName = identifier

{-- Expression Parsers --}
constant :: Parser Expression
constant = Constant <$> integer

variable :: Parser Expression
variable = Variable <$> identifier

arrayElementVariable :: Parser Expression
arrayElementVariable = ArrayElement <$> arrElemIdentifier
```

```
nil :: Parser Expression
nil = Nil <$ reserved "nil"

expression :: Parser Expression
expression = buildExpressionParser opTable $ constant <|> try arrayElementVariable <|>
    variable <|> nil
    where binop (t, op) = Infix (Binary op <$ reservedOp t) AssocLeft
          opTable = (map . map) binop operatorTable

{-- Statement Parsers --}
modOp :: Parser ModOp
modOp = ModAdd <$ symbol "+="
    <|> ModSub <$ symbol "-="
    <|> ModXor <$ symbol "^="

assign :: Parser Statement
assign = Assign <$> identifier <*> modOp <*> expression

assignArrElem :: Parser Statement
assignArrElem = AssignArrElem <$> arrElemIdentifier <*> modOp <*> expression

swap :: Parser Statement
swap = Swap <$> anyIdentifier <* symbol "<=>" <*> anyIdentifier

conditional :: Parser Statement
conditional =
    reserved "if"
    >> Conditional
    <$> expression
    <* reserved "then"
    <*> block
    <* reserved "else"
    <*> block
    <* reserved "fi"
    <*> expression

loop :: Parser Statement
loop =
    reserved "from"
    >> Loop
    <$> expression
    <* reserved "do"
    <*> block
    <* reserved "loop"
    <*> block
    <* reserved "until"
    <*> expression

localCall :: Parser Statement
localCall =
    reserved "call"
    >> LocalCall
    <$> methodName
    <*> parens (commaSep anyIdentifier)

localUncall :: Parser Statement
localUncall =
    reserved "uncall"
    >> LocalUncall
    <$> methodName
    <*> parens (commaSep anyIdentifier)

objectCall :: Parser Statement
objectCall =
    reserved "call"
```

```
    >> ObjectCall
    <$> anyIdentifier
    <* colon
    <* colon
    <*> methodName
    <*> parens (commaSep anyIdentifier)

objectUncall :: Parser Statement
objectUncall =
    reserved "uncall"
    >> ObjectUncall
    <$> anyIdentifier
    <* colon
    <* colon
    <*> methodName
    <*> parens (commaSep anyIdentifier)

objectConstruction :: Parser Statement
objectConstruction =
    reserved "new"
    >> ObjectConstruction
    <$> typeName
    <*> anyIdentifier

objectDestruction :: Parser Statement
objectDestruction =
    reserved "delete"
    >> ObjectDestruction
    <$> typeName
    <*> anyIdentifier

localBlock :: Parser Statement
localBlock =
    reserved "local"
    >> LocalBlock
    <$> dataType
    <*> identifier
    <* symbol "="
    <*> expression
    <*> block
    <* reserved "delocal"
    <* dataType
    <* identifier
    <* symbol "="
    <*> expression

objectBlock :: Parser Statement
objectBlock =
    reserved "construct"
    >> ObjectBlock
    <$> typeName
    <*> identifier
    <*> block
    <* reserved "destruct"
    <* identifier

skip :: Parser Statement
skip = Skip <$ reserved "skip"

copyReference :: Parser Statement
copyReference =
    reserved "copy"
    >> CopyReference
    <$> dataType
    <*> anyIdentifier
    <*> anyIdentifier
```

```haskell

unCopyReference :: Parser Statement
unCopyReference =
    reserved "uncopy"
    >> UnCopyReference
    <$> dataType
    <*> anyIdentifier
    <*> anyIdentifier

arrayConstruction :: Parser Statement
arrayConstruction =
    reserved "new"
    >> ArrayConstruction
    <$> arrayTypeName
    <*> identifier

arrayDestruction :: Parser Statement
arrayDestruction =
    reserved "delete"
    >> ArrayDestruction
    <$> arrayTypeName
    <*> identifier

statement :: Parser Statement
statement = try assign
        <|> try assignArrElem <|> swap
        <|> conditional
        <|> loop
        <|> try localCall
        <|> try localUncall
        <|> objectCall
        <|> objectUncall
        <|> localBlock
        <|> objectBlock
        <|> try arrayConstruction <|> objectConstruction
        <|> try arrayDestruction <|> objectDestruction
        <|> skip
        <|> copyReference
        <|> unCopyReference

block :: Parser [Statement]
block = many1 statement

{-- Top Level Parsers --}
dataType :: Parser DataType
dataType = try (IntegerArrayType <$ reserved "int" <* symbol "[" <* symbol "]")
           <|> IntegerType <$ reserved "int"
       <|> try (ObjectArrayType <$> typeName <* symbol "[" <* symbol "]")
           <|> ObjectType <$> typeName


variableDeclaration :: Parser VariableDeclaration
variableDeclaration = GDecl <$> dataType <*> identifier

methodDeclaration :: Parser MethodDeclaration
methodDeclaration =
    reserved "method"
    >> GMDecl
    <$> methodName
    <*> parens (commaSep variableDeclaration)
    <*> block

classDeclaration :: Parser ClassDeclaration
classDeclaration =
    reserved "class"
    >> GCDecl
```

```
    <$> typeName
    <*> optionMaybe (reserved "inherits" >> typeName)
    <*> many variableDeclaration
    <*> many1 methodDeclaration

program :: Parser Program
program = spaces >> GProg <$> many1 classDeclaration <* eof

parseString :: String -> Except String Program
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 ->
```

```haskell
    when (count cs > 1) (throwError $ "Multiple definitions of class " ++ n)
    where count = length . filter ((== n) . fst)

-- | Ensures legal inheritance
checkBaseClass :: ClassDeclaration -> ClassAnalyzer ()
checkBaseClass (GCDecl _ Nothing _ _) = return ()
checkBaseClass (GCDecl n (Just b) _ _) =
    do when (n == b) (throwError $ "Class " ++ n ++ " cannot inherit from itself")
       cs <- gets classes
       when (isNothing $ lookup b cs) (throwError $ "Class " ++ n ++ " cannot inherit from unknown class " ++ b)

-- |Â Checks duplicated field declarations
checkDuplicateFields :: ClassDeclaration -> ClassAnalyzer ()
checkDuplicateFields (GCDecl n _ fs _) = mapM_ checkField fs
    where count v = length . filter (\(GDecl _ v') -> v' == v) $ fs
          checkField (GDecl _ v) = when (count v > 1) (throwError $ "Multiple declarations of field " ++ v ++ " in class " ++ n)

-- | Checks duplicated method declaration in classes
checkDuplicateMethods :: ClassDeclaration -> ClassAnalyzer ()
checkDuplicateMethods (GCDecl n _ _ ms) = mapM_ checkMethod ms'
    where ms' = map (\(GMDecl n' _ _) -> n') ms
          count m = length . filter (== m) $ ms'
          checkMethod m = when (count m > 1) (throwError $ "Multiple definitions of method " ++ m ++ " in class " ++ n)

-- | Checks cyclic inheritance
checkCyclicInheritance :: ClassDeclaration -> ClassAnalyzer ()
checkCyclicInheritance (GCDecl _ Nothing _ _) = return ()
checkCyclicInheritance (GCDecl n b _ _) = checkInheritance b [n]
    where checkInheritance Nothing _ = return ()
          checkInheritance (Just b') visited =
              do when (b' `elem` visited) (throwError $ "Cyclic inheritance involving class " ++ n)
                 next <- getBaseClass b'
                 checkInheritance next (b' : visited)

-- | Sets the main class in the Class Analyzer State
setMainClass :: ClassDeclaration -> ClassAnalyzer ()
setMainClass (GCDecl n _ _ ms) = when ("main" `elem` ms') (gets mainClass >>= set)
    where
        ms' = map (\(GMDecl n' _ _) -> n') ms
        set (Just m) = throwError $ "Method main already defined in class " ++ m ++ " but redefined in class " ++ n
        set Nothing = modify $ \s -> s {mainClass = Just n}

-- | Adds classes to the state
setClasses :: ClassDeclaration -> ClassAnalyzer ()
setClasses c@(GCDecl n _ _ _) = modify $ \s -> s {classes = (n, c) : classes s}

-- | Add subclasses to the state
setSubClasses :: ClassDeclaration -> ClassAnalyzer ()
setSubClasses (GCDecl n b _ _) = modify (\s -> s { subClasses = (n, []) : subClasses s }) >> addSubClass n b

-- | Adds a subclass to the list of subclasses
addSubClass :: TypeName -> Maybe TypeName -> ClassAnalyzer ()
addSubClass _ Nothing = return ()
addSubClass n (Just b) = gets subClasses >>= \sc ->
    case lookup b sc of
        Nothing -> modify $ \s -> s { subClasses = (b, [n]) : sc }
        (Just sc') -> modify $ \s -> s { subClasses = (b, n : sc') : delete (b, sc') sc }

-- | Sets super classes in the state
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 =
```

```haskell
    do ts <- topScope
       when (isJust $ lookup n ts) (throwError $ "Redeclaration of symbol: " ++ n)
       i <- gets symbolIndex
       modify $ \s -> s { symbolTable = (i, sym) : symbolTable s, symbolIndex = 1 + i }
       addToScope (n, i)
       return i

-- | Looks up an identifier in the scope
saLookup :: Identifier -> ScopeAnalyzer SIdentifier
saLookup n = gets scopeStack >>= \ss ->
    case listToMaybe $ mapMaybe (lookup n) ss of
        Nothing -> throwError $ "Undeclared symbol: " ++ n
        Just i -> return i

-- | Scope Analyses Expressions
saExpression :: Expression -> ScopeAnalyzer SExpression
saExpression (Constant v) = pure $ Constant v
saExpression (Variable n) = Variable <$> saLookup n
saExpression Nil = pure Nil
saExpression (ArrayElement (n, e)) =
    do n' <- saLookup n
       e' <- saExpression e
       return $ ArrayElement (n', e')
saExpression (Binary binop e1 e2) =
    Binary binop
    <$> saExpression e1
    <*> saExpression e2

-- | Scope Analyses Statements
saStatement :: Statement -> ScopeAnalyzer SStatement
saStatement s =
    case s of
        (Assign n modop e) ->
            when (elem n $ var e) (throwError "Irreversible variable assignment")
            >> Assign
            <$> saLookup n
            <*> pure modop
            <*> saExpression e

        (AssignArrElem (n, e1) modop e2) ->
            when (elem (n, e1) $ varArr e2) (throwError "Irreversible variable assignment")
            >> AssignArrElem
            <$> saArrayCell n e1
            <*> pure modop
            <*> saExpression e2

        (Swap (n1, e1) (n2, e2)) ->
            Swap
            <$> maybeArrayCell n1 e1
            <*> maybeArrayCell n2 e2

        (Conditional e1 s1 s2 e2) ->
            Conditional
            <$> saExpression e1
            <*> mapM saStatement s1
            <*> mapM saStatement s2
            <*> saExpression e2

        (Loop e1 s1 s2 e2) ->
            Loop
            <$> saExpression e1
            <*> mapM saStatement s1
            <*> mapM saStatement s2
            <*> saExpression e2

        (LocalBlock t n e1 stmt e2) ->
```

```
            do e1' <- saExpression e1
               enterScope
               n' <- saInsert (LocalVariable t n) n
               stmt' <- mapM saStatement stmt
               leaveScope
               e2' <- saExpression e2
               return $ LocalBlock t n' e1' stmt' e2'

        (LocalCall m args) ->
            LocalCall
            <$> saLookup m
            <*> localCall m args

        (LocalUncall m args) ->
            LocalUncall
            <$> saLookup m
            <*> localCall m args

        (ObjectCall (o, e) m args) ->
            do when (args /= nub args || (o, e) `elem` args) (throwError $ "Irreversible
                invocation of method " ++ m)
               >> ObjectCall
               <$> maybeArrayCell o e
               <*> pure m
               <*> saArgs args

        (ObjectUncall (o, e) m args) ->
            when (args /= nub args || (o, e) `elem` args) (throwError $ "Irreversible
                invocation of method " ++ m)
            >> ObjectUncall
            <$> maybeArrayCell o e
            <*> pure m
            <*> saArgs args

        (ObjectConstruction tp (n, e)) ->
            ObjectConstruction
            <$> pure tp
            <*> maybeArrayCell n e

        (ObjectDestruction tp (n, e)) ->
            ObjectDestruction
            <$> pure tp
            <*> maybeArrayCell n e

        (ObjectBlock tp n stmt) ->
            do enterScope
               n' <- saInsert (LocalVariable (ObjectType tp) n) n
               stmt' <- mapM saStatement stmt
               leaveScope
               return $ ObjectBlock tp n' stmt'

        Skip -> pure Skip

        (CopyReference tp (n, e1) (m, e2)) ->
            CopyReference
            <$> pure tp
            <*> maybeArrayCell n e1
            <*> maybeArrayCell m e2

        (UnCopyReference tp (n, e1) (m, e2)) ->
            UnCopyReference
            <$> pure tp
            <*> maybeArrayCell n e1
            <*> maybeArrayCell m e2

        (ArrayConstruction (tp, e) n) ->
```

```haskell
            do n' <- saLookup n
               e' <- saExpression e
               return $ ArrayConstruction (tp, e') n'

        (ArrayDestruction (tp, e) n) ->
            do n' <- saLookup n
               e' <- saExpression e
               return $ ArrayDestruction (tp, e') n'

    where var (Variable n) = [n]
          var (Binary _ e1 e2) = var e1 ++ var e2
          var _ = []

          varArr (ArrayElement (n, e)) = [(n, e)]
          varArr _ = []

          isCF ClassField{} = True
          isCF _ = False

          rlookup = flip lookup

          localCall :: MethodName -> [(Identifier, Maybe Expression)] -> ScopeAnalyzer [(SIdentifier, Maybe SExpression)]
          localCall m args =
            do when (args /= nub args) (throwError $ "Irreversible invocation of method " ++ m)
               args' <- saArgs args
               st <- gets symbolTable
               when (any isCF $ mapMaybe (rlookup st . fst) args') (throwError $ "Irreversible invocation of method " ++ m)
               return args'

          saArgs :: [(Identifier, Maybe Expression)] -> ScopeAnalyzer [(SIdentifier, Maybe SExpression)]
          saArgs args =
            do (ns, es) <- pure $ unzip args
               ns' <- mapM saLookup ns
               es' <- mapM (mapM saExpression) es
               return $ zip ns' es'

          maybeArrayCell :: Identifier -> Maybe Expression -> ScopeAnalyzer (SIdentifier, Maybe SExpression)
          maybeArrayCell n e =
            do n' <- saLookup n
               e' <- mapM saExpression e
               return (n', e')

          saArrayCell :: Identifier -> Expression -> ScopeAnalyzer (SIdentifier, SExpression)
          saArrayCell n e =
            do n' <- saLookup n
               e' <- saExpression e
               return (n', e')

-- | Set the main method in the Scope Analyzer state
setMainMethod :: SIdentifier -> ScopeAnalyzer ()
setMainMethod i = modify $ \s -> s { mainMethod = i }

-- | Scope Analyses Methods
saMethod :: (TypeName, MethodDeclaration) -> ScopeAnalyzer (TypeName, SMethodDeclaration)
saMethod (t, GMDecl m ps body) =
    do m' <- saLookup m
       when (m == "main") (setMainMethod m')
       enterScope
       ps' <- mapM insertMethodParameter ps
       body' <- mapM saStatement body
       leaveScope
```

```haskell
        return (t, GMDecl m' ps' body')
    where insertMethodParameter (GDecl tp n) = GDecl tp <$> saInsert (MethodParameter tp n) n

-- | Returns subclasses for a given type name
getSubClasses :: TypeName -> ScopeAnalyzer [ClassDeclaration]
getSubClasses n =
    do cs <- gets $ classes . caState
       sc <- gets $ subClasses . caState
       case lookup n sc of
           Nothing -> throwError $ "ICE: Unknown class " ++ n
           (Just sc') -> return $ mapMaybe (rlookup cs) sc'
    where rlookup = flip lookup

-- | Returns method name at given index
getMethodName :: SIdentifier -> ScopeAnalyzer (SIdentifier, MethodName)
getMethodName i = gets symbolTable >>= \st ->
    case lookup i st of
        (Just (Method _ m)) -> return (i, m)
        _ -> throwError $ "ICE: Invalid method index " ++ show i

-- | Prefixes the virtual table
prefixVtable :: [(SIdentifier, MethodName)] -> (SIdentifier, MethodName) -> [(SIdentifier,
    MethodName)]
prefixVtable [] m' = [m']
prefixVtable (m:ms) m' = if comp m m' then m':ms else m : prefixVtable ms m'
    where comp (_, n) (_, n') = n == n'

-- | Scope Analyses a passed class
saClass :: Offset -> [SIdentifier] -> ClassDeclaration -> ScopeAnalyzer [(TypeName,
    SMethodDeclaration)]
saClass offset pids (GCDecl c _ fs ms) =
    do enterScope
       mapM_ insertClassField $ zip [offset..] fs
       m1 <- mapM getMethodName pids
       m2 <- mapM insertMethod ms
       let m3 = map fst $ foldl prefixVtable m1 m2
           offset' = genericLength fs + offset
       modify $ \s -> s { virtualTables = (c, m3) : virtualTables s }
       sc <- getSubClasses c
       ms' <- concat <$> mapM (saClass offset' m3) sc
       ms'' <- mapM saMethod $ zip (repeat c) ms
       leaveScope
       return $ ms' ++ ms''
    where insertClassField (o, GDecl tp n) = saInsert (ClassField tp n c o) n
          insertMethod (GMDecl n ps _) = saInsert (Method (map getType ps) n) n >>=
              getMethodName
          getType (GDecl tp _) = tp

-- | Analyses Programs
saProgram :: Program -> ScopeAnalyzer SProgram
saProgram (GProg cs) = concat <$> mapM (saClass 2 []) cs

-- | Performs scope analysis on the entire program
scopeAnalysis :: (Program, CAState) -> Except String (SProgram, SAState)
scopeAnalysis (p, s) = runStateT (runSA $ saProgram p) $ initialState s

-- | Pretty prints the current Scope Analysis State Monad
printSAState :: (Show a, MonadIO m) => (t, a) -> m ()
printSAState (_, s) = pPrint s
```

TypeChecker.hs

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

module TypeChecker (typeCheck) where

import Data.List
import Data.Maybe

import Control.Monad.Reader
import Control.Monad.Except
import Control.Exception

import Debug.Trace (trace, traceShow)

import AST
import ClassAnalyzer
import ScopeAnalyzer

newtype TypeChecker a = TypeChecker { runTC :: ReaderT SAState (Except String) a }
    deriving (Functor, Applicative, Monad, MonadReader SAState, MonadError String)

getType :: SIdentifier -> TypeChecker DataType
getType i = asks symbolTable >>= \st ->
    case lookup i st of
        (Just (LocalVariable t _)) -> return t
        (Just (ClassField t _ _ _)) -> return t
        (Just (MethodParameter t _)) -> return t
        _ -> throwError $ "ICE: Invalid index " ++ show i

getParameterTypes :: SIdentifier -> TypeChecker [DataType]
getParameterTypes i = asks symbolTable >>= \st ->
    case lookup i st of
        (Just (Method ps _)) -> return ps
        _ -> throwError $ "ICE: Invalid index " ++ show i

expectType :: DataType -> DataType -> TypeChecker ()
expectType t1 t2 = unless (t1 == t2) (throwError $ "Expected type: " ++ show t1 ++ "\nActual type: " ++ show t2)

getClassMethods :: TypeName -> TypeChecker [MethodDeclaration]
getClassMethods n = asks (classMethods . caState) >>= \cm ->
    case lookup n cm of
        Nothing -> throwError $ "ICE: Unknown class " ++ n
        (Just ms) -> return ms

getDynamicParameterTypes :: TypeName -> MethodName -> TypeChecker [DataType]
getDynamicParameterTypes n m = getClassMethods n >>= \ms ->
    case find (\(GMDecl m' _ _) -> m == m') ms of
        Nothing -> throwError $ "Class " ++ n ++ " does not support method " ++ m
        (Just (GMDecl _ ps _)) -> return $ map (\(GDecl tp _) -> tp) ps

getArrayType :: DataType -> DataType
getArrayType tp = case tp of
                    IntegerArrayType -> IntegerType
                    ObjectArrayType t -> ObjectType t

checkCall :: [(SIdentifier, Maybe SExpression)] -> [DataType] -> TypeChecker ()
checkCall args ps =
    when (la /= lp) (throwError err)
    >> mapM (mapM tcExpression . snd) args
    >> mapM (getType . fst) args
    >>= \as -> mapM_ checkArgument (zip as ps)
    where la = length args
          lp = length ps
```

```
        err = "Passed " ++ show la ++ " argument(s) to method expecting " ++ show lp ++ " argument(s)"

checkArgument :: (DataType, DataType) -> TypeChecker ()
checkArgument (ObjectType ca, ObjectType cp) = asks (superClasses . caState) >>= \sc ->
    unless (ca == cp || maybe False (elem cp) (lookup ca sc)) (throwError $ "Class " ++ ca ++ " not a subtype of class " ++ cp)
checkArgument (ObjectType ca, ObjectArrayType cp) = asks (superClasses . caState) >>= \sc ->
    unless (ca == cp || maybe False (elem cp) (lookup ca sc)) (throwError $ "Class " ++ ca ++ " not a subtype of class " ++ cp)
checkArgument (ObjectArrayType ca, ObjectType cp) = asks (superClasses . caState) >>= \sc ->
    unless (ca == cp || maybe False (elem cp) (lookup ca sc)) (throwError $ "Class " ++ ca ++ " not a subtype of class " ++ cp)
checkArgument (IntegerArrayType, tp) = expectType (getArrayType IntegerArrayType) tp
checkArgument (ta, IntegerArrayType) = expectType (getArrayType IntegerArrayType) ta
checkArgument (ta, tp) = expectType tp ta

tcExpression :: SExpression -> TypeChecker DataType
tcExpression (Constant _) = pure IntegerType
tcExpression (Variable n) = getType n
tcExpression Nil = pure NilType
tcExpression (ArrayElement (n, e)) =
    do t <- getType n
       expectType ArrayType t
       e' <- tcExpression e
       expectType IntegerType e'
       return $ getArrayType t
tcExpression (Binary binop e1 e2)
    | binop == Eq || binop == Neq =
        do t1 <- tcExpression e1
           t2 <- tcExpression e2
           expectType t1 t2
           pure IntegerType
    | otherwise =
        do t1 <- tcExpression e1
           t2 <- tcExpression e2
           expectType t1 IntegerType
           expectType t2 IntegerType
           pure IntegerType

tcStatement :: SStatement -> TypeChecker ()
tcStatement s =
    case s of
        (Assign n _ e) ->
            getType n
            >>= expectType IntegerType
            >> tcExpression e
            >>= expectType IntegerType

        (AssignArrElem (n, e1) _ e2) ->
            getType n
            >>= expectType IntegerArrayType
            >> tcExpression e1
            >>= expectType IntegerType
            >> tcExpression e2
            >>= expectType IntegerType

        (Swap (n1, e1) (n2, e2)) ->
            do t1 <- getType n1
               t2 <- getType n2
               if isNothing e1 /= isNothing e2
                 then catchError (checkArgument (t1, t2)) (\_ -> checkArgument (t2, t1))
                 else expectType (if isNothing e1 then t1 else getArrayType t1) (if isNothing e2 then t2 else getArrayType t2)

        (Conditional e1 s1 s2 e2) ->
```

```
            tcExpression e1
            >>= expectType IntegerType
            >> mapM_ tcStatement s1
            >> mapM_ tcStatement s2
            >> tcExpression e2
            >>= expectType IntegerType

        (Loop e1 s1 s2 e2) ->
            tcExpression e1
            >>= expectType IntegerType
            >> mapM_ tcStatement s1
            >> mapM_ tcStatement s2
            >> tcExpression e2
            >>= expectType IntegerType

        (ObjectBlock _ _ stmt) ->
            mapM_ tcStatement stmt

        (LocalBlock t n e1 stmt e2) ->
            getType n
            >> tcExpression e1
            >>= expectType (if t == IntegerType then IntegerType else NilType)
            >> mapM_ tcStatement stmt
            >> tcExpression e2
            >>= expectType (if t == IntegerType then IntegerType else NilType)

        (LocalCall m args) ->
            getParameterTypes m
            >>= checkCall args

        (LocalUncall m args) ->
            getParameterTypes m
            >>= checkCall args

        (ObjectCall (o, e) m args) ->
            do t <- getType o
               e' <- mapM tcExpression e
               case t of
                    (ObjectType tn) -> getDynamicParameterTypes tn m >>= checkCall args
                    (ObjectArrayType tn) ->
                     case e' of
                         Nothing -> throwError $ "Non-object type " ++ show t ++ " does not support method invocation"
                         _ -> getDynamicParameterTypes tn m >>= checkCall args
                    _ -> throwError $ "Non-object type " ++ show t ++ " does not support method invocation"

        (ObjectUncall (o, e) m args) ->
            do t <- getType o
               e' <- mapM tcExpression e
               case t of
                    (ObjectType tn) -> getDynamicParameterTypes tn m >>= checkCall args
                    (ObjectArrayType tn) ->
                     case e' of
                         Nothing -> throwError $ "Non-object type " ++ show t ++ " does not support method invocation"
                         _ -> getDynamicParameterTypes tn m >>= checkCall args
                    _ -> throwError $ "Non-object type " ++ show t ++ " does not support method invocation"

        Skip -> pure ()

        (ObjectConstruction tp (n, e)) ->
            do t <- getType n
               e' <- mapM tcExpression e
               case e' of
```

```
                    Nothing -> expectType t (ObjectType tp)
                    _       -> checkArgument (ObjectType tp, t)

        (ObjectDestruction tp (n, e)) ->
            do t <- getType n
               _ <- mapM tcExpression e
               case t of
                (ObjectType _) -> expectType t (ObjectType tp)
                (ObjectArrayType _) -> checkArgument (ObjectType tp, t)
                _ -> throwError $ "Expected type: " ++ show (ObjectType tp) ++ " Actual type:
                    " ++ show t

        -- Allow copying with a copy type
        CopyReference _ (n, e1) (m, e2) ->
            do t1 <- getType n
               t2 <- getType m
               e1' <- mapM tcExpression e1
               e2' <- mapM tcExpression e2
               when (t1 == IntegerType || t2 == IntegerType) (throwError "Integer types does
                   not support reference copying")
               if isNothing e1 /= isNothing e2
                 then catchError (checkArgument (t1, t2)) (\_ -> checkArgument (t2, t1))
                 else expectType (if isNothing e1 then t1 else getArrayType t1) (if isNothing
                     e2 then t2 else getArrayType t2)

        -- Allow uncopying with two identical copies
        UnCopyReference _ (n, e1) (m, e2) ->
            do t1 <- getType n
               t2 <- getType m
               e1' <- mapM tcExpression e1
               e2' <- mapM tcExpression e2
               when (t1 == IntegerType || t2 == IntegerType) (throwError "Integer types does
                   not support reference copying")
               if isNothing e1 /= isNothing e2
                 then catchError (checkArgument (t1, t2)) (\_ -> checkArgument (t2, t1))
                 else expectType (if isNothing e1 then t1 else getArrayType t1) (if isNothing
                     e2 then t2 else getArrayType t2)


        (ArrayConstruction (tp, e) n) ->
            do t <- getType n
               _ <- tcExpression e
               case tp of
                 "int" -> expectType t IntegerArrayType
                 _     -> expectType t (ObjectArrayType tp)

        (ArrayDestruction (tp, e) n) ->
            do t <- getType n
               _ <- tcExpression e
               case tp of
                 "int" -> expectType t IntegerArrayType
                 _     -> checkArgument (ObjectArrayType tp, t)

getMethodName :: SIdentifier -> TypeChecker Identifier
getMethodName i = asks symbolTable >>= \st ->
    case lookup i st of
        (Just (Method _ n)) -> return n
        _ -> throwError $ "ICE: Invalid index " ++ show i

tcMethod :: (TypeName, SMethodDeclaration) -> TypeChecker ()
tcMethod (_, GMDecl _ [] body) = mapM_ tcStatement body
tcMethod (_, GMDecl i (_:_) body) = getMethodName i >>= \n ->
    when (n == "main") (throwError "Method main has invalid signature")
    >> mapM_ tcStatement body

tcProgram :: SProgram -> TypeChecker (SProgram, SAState)
```

```
tcProgram p = (,) p <$> (mapM_ tcMethod p >> ask)

typeCheck :: (SProgram, SAState) -> Except String (SProgram, SAState)
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
registerRO :: Register
registerRO = Reg 2

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

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

-- | Register containing Heap pointer
```

```haskell
registerHP :: Register
registerHP = Reg 5

-- | Pushes a new register to the register stack
pushRegister :: SIdentifier -> CodeGenerator Register
pushRegister i = do ri <- gets registerIndex
                    modify $ \s -> s { registerIndex = 1 + ri, registerStack = (i, Reg ri) : registerStack s }
                    return $ Reg ri

-- | Pop a register from the register stack
popRegister :: CodeGenerator ()
popRegister = modify $ \s -> s { registerIndex = (-1) + registerIndex s, registerStack = drop 1 $ registerStack s }

-- | Reserve a tmp register
tempRegister :: CodeGenerator Register
tempRegister =
    do ri <- gets registerIndex
       modify $ \s -> s { registerIndex = 1 + ri }
       return $ Reg ri

-- | Clear reverved tmp register
popTempRegister :: CodeGenerator ()
popTempRegister = modify $ \s -> s { registerIndex = (-1) + registerIndex s }

-- | Lookup register of given identifier
lookupRegister :: SIdentifier -> CodeGenerator Register
lookupRegister i = gets registerStack >>= \rs ->
    case lookup i rs of
        Nothing -> throwError $ "ICE: No register reserved for index " ++ show i
        (Just r) -> return r

-- | Returns the method name of a valid method identifier
getMethodName :: SIdentifier -> CodeGenerator MethodName
getMethodName i = gets (symbolTable . saState) >>= \st ->
    case lookup i st of
        (Just (Method _ n)) -> return n
        _ -> throwError $ "ICE: Invalid method index " ++ show i

-- | Inserts a unique method label in the label table for a given method identifier
insertMethodLabel :: SIdentifier -> CodeGenerator ()
insertMethodLabel m =
    do n <- getMethodName m
       i <- gets labelIndex
       modify $ \s -> s { labelIndex = 1 + i, labelTable = (m, "l_" ++ n ++ "_" ++ show i) : labelTable s }

-- | Returns the Method label for a method identifier
getMethodLabel :: SIdentifier -> CodeGenerator Label
getMethodLabel m = gets labelTable >>= \lt ->
    case lookup m lt of
        (Just l) -> return l
        Nothing -> insertMethodLabel m >> getMethodLabel m

-- | Returns a unique label by appending the label index to a passed label type
getUniqueLabel :: Label -> CodeGenerator Label
getUniqueLabel l =
    do i <- gets labelIndex
       modify $ \s -> s { labelIndex = 1 + i }
       return $ l ++ "_" ++ show i

-- | Returns the address to the variable of a given identifier
loadVariableAddress :: SIdentifier -> CodeGenerator (Register, [(Maybe Label, MInstruction)], CodeGenerator ())
loadVariableAddress n = gets (symbolTable . saState) >>= \st ->
```

```haskell
    case lookup n st of
        (Just (ClassField _ _ _ o)) -> tempRegister >>= \r -> return (r, [(Nothing, ADD r registerThis), (Nothing, ADDI r $ Immediate o)], popTempRegister)
        (Just (LocalVariable _ _)) -> lookupRegister n >>= \r -> return (r, [], return ())
        (Just (MethodParameter _ _)) -> lookupRegister n >>= \r -> return (r, [], return ())
        _ -> throwError $ "ICE: Invalid variable index " ++ show n

-- | Returns the value of a variable of given identifier
loadVariableValue :: SIdentifier -> CodeGenerator (Register, [(Maybe Label, MInstruction)], CodeGenerator ())
loadVariableValue n =
    do (ra, la, ua) <- loadVariableAddress n
       rv <- tempRegister
       return (rv, la ++ [(Nothing, EXCH rv ra)] ++ invertInstructions la, popTempRegister >> ua)

-- | Returns address an array element
loadArrayElementVariableAddress :: SIdentifier -> SExpression -> CodeGenerator (Register, [(Maybe Label, MInstruction)], CodeGenerator ())
loadArrayElementVariableAddress n e =
    do (ra, la, ua) <- loadVariableAddress n
       (re, le, ue) <- cgExpression e
       rv <- tempRegister
       rt <- tempRegister
       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)

-- | Returns the value of an array element
loadArrayElementVariableValue :: SIdentifier -> SExpression -> CodeGenerator (Register, [(Maybe Label, MInstruction)], CodeGenerator ())
loadArrayElementVariableValue n e =
    do (ra, la, ua) <- loadArrayElementVariableAddress n e
       rv <- tempRegister
       return (rv, la ++ [(Nothing, EXCH rv ra)] ++ invertInstructions la , popTempRegister >> ua)

--Â | Returns pointer to free list at given index
loadFreeListAddress :: Register -> CodeGenerator (Register, [(Maybe Label, MInstruction)], CodeGenerator ())
loadFreeListAddress index = tempRegister >>= \rt -> return (rt, [(Nothing, XOR rt registerFLPs), (Nothing, ADD rt index)], popTempRegister)

-- |Â Returns a copy of the pointer to the head of the free list at the given register
loadHeadAtFreeList :: Register -> CodeGenerator (Register, [(Maybe Label, MInstruction)], CodeGenerator ())
loadHeadAtFreeList rFreeList =
    do rv <- tempRegister
       rt <- tempRegister
       let copyAddress = [(Nothing, EXCH rt rFreeList),
                          (Nothing, XOR rv rt),
                          (Nothing, EXCH rt rFreeList)]
       return (rv, copyAddress, popTempRegister >> popTempRegister)

-- | Code generation for binary operators
cgBinOp :: BinOp -> Register -> Register -> CodeGenerator (Register, [(Maybe Label, MInstruction)], CodeGenerator ())
cgBinOp Add r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, XOR rt r1), (Nothing, ADD rt r2)], popTempRegister)
cgBinOp Sub r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, XOR rt r1), (Nothing, SUB rt r2)], popTempRegister)
cgBinOp Xor r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, XOR rt r1), (Nothing, XOR rt r2)], popTempRegister)
cgBinOp BitAnd r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, ANDX rt r1 r2)], popTempRegister)
```

```
cgBinOp BitOr r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, ORX rt r1 r2)],
    popTempRegister)
cgBinOp Lt r1 r2 =
    do rt <- tempRegister
       rc <- tempRegister
       l_top <- getUniqueLabel "cmp_top"
       l_bot <- getUniqueLabel "cmp_bot"
       let cmp = [(Nothing, XOR rt r1),
                  (Nothing, SUB rt r2),
                  (Just l_top, BGEZ rt l_bot),
                  (Nothing, XORI rc $ Immediate 1),
                  (Just l_bot, BGEZ rt l_top)]
       return (rc, cmp, popTempRegister >> popTempRegister)
cgBinOp Gt r1 r2 =
    do rt <- tempRegister
       rc <- tempRegister
       l_top <- getUniqueLabel "cmp_top"
       l_bot <- getUniqueLabel "cmp_bot"
       let cmp = [(Nothing, XOR rt r1),
                  (Nothing, SUB rt r2),
                  (Just l_top, BLEZ rt l_bot),
                  (Nothing, XORI rc $ Immediate 1),
                  (Just l_bot, BLEZ rt l_top)]
       return (rc, cmp, popTempRegister >> popTempRegister)
cgBinOp Eq r1 r2 =
    do rt <- tempRegister
       l_top <- getUniqueLabel "cmp_top"
       l_bot <- getUniqueLabel "cmp_bot"
       let cmp = [(Just l_top, BNE r1 r2 l_bot),
                  (Nothing, XORI rt $ Immediate 1),
                  (Just l_bot, BNE r1 r2 l_top)]
       return (rt, cmp, popTempRegister)
cgBinOp Neq r1 r2 =
    do rt <- tempRegister
       l_top <- getUniqueLabel "cmp_top"
       l_bot <- getUniqueLabel "cmp_bot"
       let cmp = [(Just l_top, BEQ r1 r2 l_bot),
                  (Nothing, XORI rt $ Immediate 1),
                  (Just l_bot, BEQ r1 r2 l_top)]
       return (rt, cmp, popTempRegister)
cgBinOp Lte r1 r2 =
    do rt <- tempRegister
       rc <- tempRegister
       l_top <- getUniqueLabel "cmp_top"
       l_bot <- getUniqueLabel "cmp_bot"
       let cmp = [(Nothing, XOR rt r1),
                  (Nothing, SUB rt r2),
                  (Just l_top, BGTZ rt l_bot),
                  (Nothing, XORI rc $ Immediate 1),
                  (Just l_bot, BGTZ rt l_top)]
       return (rc, cmp, popTempRegister >> popTempRegister)
cgBinOp Gte r1 r2 =
    do rt <- tempRegister
       rc <- tempRegister
       l_top <- getUniqueLabel "cmp_top"
       l_bot <- getUniqueLabel "cmp_bot"
       let cmp = [(Nothing, XOR rt r1),
                  (Nothing, SUB rt r2),
                  (Just l_top, BLTZ rt l_bot),
                  (Nothing, XORI rc $ Immediate 1),
                  (Just l_bot, BLTZ rt l_top)]
       return (rc, cmp, popTempRegister >> popTempRegister)
cgBinOp _ _ _ = throwError "ICE: Binary operator not implemented"

-- | Code generation for expressions
```

```haskell
cgExpression :: SExpression -> CodeGenerator (Register, [(Maybe Label, MInstruction)], CodeGenerator ())
cgExpression (Constant 0) = return (registerZero, [], return ())
cgExpression (Constant n) = tempRegister >>= \rt -> return (rt, [(Nothing, XORI rt $ Immediate n)], popTempRegister)
cgExpression (Variable i) = loadVariableValue i
cgExpression (ArrayElement (n, e)) = loadArrayElementVariableValue n e
cgExpression Nil = return (registerZero, [], return ())
cgExpression (Binary op e1 e2) =
    do (r1, l1, u1) <- cgExpression e1
       (r2, l2, u2) <- cgExpression e2
       (ro, lo, uo) <- cgBinOp op r1 r2
       return (ro, l1 ++ l2 ++ lo, uo >> u2 >> u1)

-- | Code generation for binary expressions
cgBinaryExpression :: SExpression -> CodeGenerator (Register, [(Maybe Label, MInstruction)], CodeGenerator ())
cgBinaryExpression e =
    do (re, le, ue) <- cgExpression e
       rt <- tempRegister
       l_top <- getUniqueLabel "f_top"
       l_bot <- getUniqueLabel "f_bot"
       let flatten = [(Just l_top, BEQ re registerZero l_bot),
                      (Nothing, XORI rt $ Immediate 1),
                      (Just l_bot, BEQ re registerZero l_top)]
       return (rt, le ++ flatten, popTempRegister >> ue)

-- | Code generation for assignments
cgAssign :: SIdentifier -> ModOp -> SExpression -> CodeGenerator [(Maybe Label, MInstruction)]
cgAssign n modop e =
    do (rt, lt, ut) <- loadVariableValue n
       (re, le, ue) <- cgExpression e
       ue >> ut
       return $ lt ++ le ++ [(Nothing, cgModOp modop rt re)] ++ invertInstructions (lt ++ le)
    where cgModOp ModAdd = ADD
          cgModOp ModSub = SUB
          cgModOp ModXor = XOR

-- | Code generation for assignments
cgAssignArrElem :: (SIdentifier, SExpression) -> ModOp -> SExpression -> CodeGenerator [(Maybe Label, MInstruction)]
cgAssignArrElem (n, e1) modop e2 =
    do (rt, lt, ut) <- loadArrayElementVariableValue n e1
       (re, le, ue) <- cgExpression e2
       l <- getUniqueLabel "assArrElem"
       ue >> ut
       return $ lt ++ le ++ [(Just l, cgModOp modop rt re)] ++ invertInstructions (lt ++ le)
    where cgModOp ModAdd = ADD
          cgModOp ModSub = SUB
          cgModOp ModXor = XOR

-- | Ensures correct loads for swapping
loadForSwap :: (SIdentifier, Maybe SExpression) -> CodeGenerator (Register, [(Maybe Label, MInstruction)], CodeGenerator ())
loadForSwap (n, x) = gets (symbolTable . saState) >>= \st ->
    case lookup n st of
        (Just (ClassField IntegerArrayType _ _ _)) -> case x of
                                                        Just x' -> loadArrayElementVariableValue n x'
                                                        _ -> loadVariableValue n
        (Just (ClassField (ObjectArrayType _) _ _ _)) -> case x of
                                                          Just x' -> loadArrayElementVariableValue n x'
                                                          _ -> loadVariableValue n
        (Just ClassField {}) -> loadVariableValue n
        (Just (LocalVariable IntegerType _)) -> loadVariableValue n
```

```
        (Just (LocalVariable (ObjectType _) _)) -> loadVariableValue n
        (Just (LocalVariable (CopyType _) _)) -> loadVariableValue n
        (Just (LocalVariable IntegerArrayType _)) -> case x of
                                                      Just x' ->
                                                          loadArrayElementVariableValue n x'
                                                      _ -> loadVariableValue n
        (Just (LocalVariable (ObjectArrayType _) _)) -> case x of
                                                       Just x' ->
                                                           loadArrayElementVariableValue n x'
                                                       _ -> loadVariableValue n
        (Just (MethodParameter IntegerType _)) -> loadVariableValue n
        (Just (MethodParameter (ObjectType _) _)) -> loadVariableValue n
        (Just (MethodParameter (CopyType _) _)) -> loadVariableValue n
        (Just (MethodParameter IntegerArrayType _)) -> case x of
                                                        Just x' ->
                                                            loadArrayElementVariableValue n x
                                                            '
                                                        _ -> loadVariableValue n
        (Just (MethodParameter (ObjectArrayType _) _)) -> case x of
                                                           Just x' ->
                                                               loadArrayElementVariableValue
                                                               n x'
                                                           _ -> loadVariableValue n
        _ -> throwError $ "ICE: Invalid variable index " ++ show n

-- | Code generation for swaps
cgSwap :: (SIdentifier, Maybe SExpression) -> (SIdentifier, Maybe SExpression) ->
    CodeGenerator [(Maybe Label, MInstruction)]
cgSwap n1 n2 = if n1 == n2 then return [] else
    do (r1, l1, u1) <- loadForSwap n1
       (r2, l2, u2) <- loadForSwap n2
       u2 >> u1
       l <- getUniqueLabel "swap"
       let swap = [(Just l, XOR r1 r2), (Nothing, XOR r2 r1), (Nothing, XOR r1 r2)]
       return $ l1 ++ l2 ++ swap ++ invertInstructions (l1 ++ l2)

-- | Code generation for conditionals
cgConditional :: SExpression -> [SStatement] -> [SStatement] -> SExpression -> CodeGenerator
    [(Maybe Label, MInstruction)]
cgConditional e1 s1 s2 e2 =
    do l_test <- getUniqueLabel "test"
       l_assert_t <- getUniqueLabel "assert_true"
       l_test_f <- getUniqueLabel "test_false"
       l_assert <- getUniqueLabel "assert"
       rt <- tempRegister
       (re1, le1, ue1) <- cgBinaryExpression e1
       ue1
       s1' <- concat <$> mapM cgStatement s1
       s2' <- concat <$> mapM cgStatement s2
       (re2, le2, ue2) <- cgBinaryExpression e2
       ue2 >> popTempRegister --rt
       return $ le1 ++ [(Nothing, XOR rt re1)] ++ invertInstructions le1 ++
                  [(Just l_test, BEQ rt registerZero l_test_f), (Nothing, XORI rt $ Immediate 1)
                      ] ++
                  s1' ++ [(Nothing, XORI rt $ Immediate 1), (Just l_assert_t, BRA l_assert), (
                      Just l_test_f, BRA l_test)] ++
                  s2' ++ [(Just l_assert, BNE rt registerZero l_assert_t)] ++
                  le2 ++ [(Nothing, XOR rt re2)] ++ invertInstructions le2

-- | Code generation for loops
cgLoop :: SExpression -> [SStatement] -> [SStatement] -> SExpression -> CodeGenerator [(Maybe
    Label, MInstruction)]
cgLoop e1 s1 s2 e2 =
    do l_entry <- getUniqueLabel "entry"
       l_test <- getUniqueLabel "test"
       l_assert <- getUniqueLabel "assert"
```

```haskell
        l_exit <- getUniqueLabel "exit"
        rt <- tempRegister
        (re1, le1, ue1) <- cgBinaryExpression e1
        ue1
        s1' <- concat <$> mapM cgStatement s1
        s2' <- concat <$> mapM cgStatement s2
        (re2, le2, ue2) <- cgBinaryExpression e2
        ue2 >> popTempRegister --rt
        return $ [(Nothing, XORI rt $ Immediate 1), (Just l_entry, BEQ rt registerZero l_assert)] ++
                 le1 ++ [(Nothing, XOR rt re1)] ++ invertInstructions le1 ++
                 s1' ++ le2 ++ [(Nothing, XOR rt re2)] ++ invertInstructions le2 ++
                 [(Just l_test, BNE rt registerZero l_exit)] ++ s2' ++
                 [(Just l_assert, BRA l_entry), (Just l_exit, BRA l_test), (Nothing, XORI rt $ Immediate 1)]

-- | Code generation for object blocks
cgObjectBlock :: TypeName -> SIdentifier -> [SStatement] -> CodeGenerator [(Maybe Label, MInstruction)]
cgObjectBlock tp n stmt =
    do rn <- pushRegister n
       let push = [(Nothing, XOR rn registerSP), (Nothing, SUBI registerSP $ Immediate 1)]
       malloc <- cgObjectConstruction tp (n, Nothing)
       stmt' <- concat <$> mapM cgStatement stmt
       free <- cgObjectDestruction tp (n, Nothing)
       popRegister
       return $ push ++ malloc ++ stmt' ++ free ++ invertInstructions push

-- | Code generation for local blocks
cgLocalBlock :: SIdentifier -> SExpression -> [SStatement] -> SExpression -> CodeGenerator [(Maybe Label, MInstruction)]
cgLocalBlock n e1 stmt e2 =
    do rn <- pushRegister n
       (re1, le1, ue1) <- cgExpression e1
       rt1 <- tempRegister
       popTempRegister >> ue1
       stmt' <- concat <$> mapM cgStatement stmt
       (re2, le2, ue2) <- cgExpression e2
       rt2 <- tempRegister
       popTempRegister >> ue2
       popRegister --rn
       l <- getUniqueLabel "localBlock"
       let create re rt = [(Just l, XOR rn registerSP),
                           (Nothing, XOR rt re),
                           (Nothing, EXCH rt registerSP),
                           (Nothing, SUBI registerSP $ Immediate 1)]
           load = le1 ++ create re1 rt1 ++ invertInstructions le1
           clear = le2 ++ invertInstructions (create re2 rt2) ++ invertInstructions le2
       return $ load ++ stmt' ++ clear

-- | Code generation for calls
cgCall :: [(SIdentifier, Maybe SExpression)] -> [(Maybe Label, MInstruction)] -> Register -> CodeGenerator [(Maybe Label, MInstruction)]
cgCall args jump this =
    do (ra, la, ua) <- unzip3 <$> mapM loadAddr args
       sequence_ ua
       rs <- gets registerStack
       let rr = (registerThis : map snd rs) \\ (this : ra)
           store = concatMap push $ rr ++ ra ++ [this]
       return $ concat la ++ store ++ jump ++ invertInstructions store ++ invertInstructions (concat la)
    where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]
          loadAddr (n, e) =
            case e of
                Nothing -> loadVariableAddress n
                Just e' -> loadArrayElementVariableAddress n e'
```

```
-- | Code generation for local calling
cgLocalCall :: SIdentifier -> [(SIdentifier, Maybe SExpression)]-> CodeGenerator [(Maybe Label
    , MInstruction)]
cgLocalCall m args = getMethodLabel m >>= \l_m -> cgCall args [(Nothing, BRA l_m)]
    registerThis

-- | Code generation for local uncalling
cgLocalUncall :: SIdentifier -> [(SIdentifier, Maybe SExpression)] -> CodeGenerator [(Maybe
    Label, MInstruction)]
cgLocalUncall m args = getMethodLabel m >>= \l_m -> cgCall args [(Nothing, RBRA l_m)]
    registerThis

-- | Returns the type associated with a given identifier
getType :: SIdentifier -> CodeGenerator TypeName
getType i = gets (symbolTable . saState) >>= \st ->
    case lookup i st of
        (Just (LocalVariable (ObjectType tp) _)) -> return tp
        (Just (ClassField (ObjectType tp) _ _ _)) -> return tp
        (Just (MethodParameter (ObjectType tp) _)) -> return tp
        (Just (LocalVariable (ObjectArrayType tp) _)) -> return tp
        (Just (ClassField (ObjectArrayType tp) _ _ _)) -> return tp
        (Just (MethodParameter (ObjectArrayType tp) _)) -> return tp
        _ -> throwError $ "ICE: Invalid object variable index " ++ show i

-- | Load the return offset for methods
loadMethodAddress :: (SIdentifier, Register) -> MethodName -> CodeGenerator (Register, [(Maybe
     Label, MInstruction)])
loadMethodAddress (o, ro) m =
    do rv <- tempRegister
       rt <- tempRegister
       rtgt <- tempRegister
       offsetMacro <- OffsetMacro <$> getType o <*> pure m
       l <- getUniqueLabel "loadMetAdd"
       let load = [(Just l, EXCH rv ro),
                   (Nothing, ADDI rv offsetMacro),
                   (Nothing, EXCH rt rv),
                   (Nothing, XOR rtgt rt),
                   (Nothing, EXCH rt rv),
                   (Nothing, SUBI rv offsetMacro),
                   (Nothing, EXCH rv ro)]
       return (rtgt, load)

-- | Load address or value needed for calls
loadForCall :: (SIdentifier, Maybe SExpression) -> CodeGenerator (Register, [(Maybe Label,
    MInstruction)], CodeGenerator ())
loadForCall (n, e) = gets (symbolTable . saState) >>= \st ->
    case lookup n st of
        (Just (ClassField (ObjectArrayType _) _ _ _)) ->
            case e of
                Just x' -> loadArrayElementVariableValue n x'
                _ -> throwError $ "ICE: Invalid variable index " ++ show n
        (Just ClassField {}) -> loadVariableValue n
        (Just (LocalVariable (ObjectType _) _)) -> loadVariableValue n
        (Just (LocalVariable (CopyType _) _)) -> loadVariableValue n
        (Just (LocalVariable (ObjectArrayType _) _)) ->
            case e of
                Just x' -> loadArrayElementVariableValue n x'
                _ -> throwError $ "ICE: Invalid variable index " ++ show n
        (Just _) -> loadVariableValue n
        _ -> throwError $ "ICE: Invalid variable index " ++ show n

-- | Code generation for object calls
cgObjectCall :: (SIdentifier, Maybe SExpression) -> MethodName -> [(SIdentifier, Maybe
    SExpression)] -> CodeGenerator [(Maybe Label, MInstruction)]
cgObjectCall (o, e) m args =
```

```haskell
    do (ro, lo, uo) <- loadForCall (o, e)
       rt <- tempRegister
       (rtgt, loadAddress) <- loadMethodAddress (o, rt) m
       l_jmp <- getUniqueLabel "l_jmp"
       let jp = [(Nothing, SUBI rtgt $ AddressMacro l_jmp),
                 (Just l_jmp, SWAPBR rtgt),
                 (Nothing, NEG rtgt),
                 (Nothing, ADDI rtgt $ AddressMacro l_jmp)]
       call <- cgCall args jp rt
       popTempRegister >> popTempRegister >> popTempRegister -- rv, rt & rtgt from loadMethodAddr
       popTempRegister >> uo
       let load = lo ++ [(Nothing, XOR rt ro)] ++ loadAddress ++ invertInstructions lo
       return $ load ++ call ++ invertInstructions load

-- | Code generation for object uncalls
cgObjectUncall :: (SIdentifier, Maybe SExpression) -> MethodName -> [(SIdentifier, Maybe SExpression)] -> CodeGenerator [(Maybe Label, MInstruction)]
cgObjectUncall (o, e) m args =
    do (ro, lo, uo) <- loadForCall (o, e)
       rt <- tempRegister
       (rtgt, loadAddress) <- loadMethodAddress (o, rt) m
       l_jmp <- getUniqueLabel "l_jmp"
       l_rjmp_top <- getUniqueLabel "l_rjmp_top"
       l_rjmp_bot <- getUniqueLabel "l_rjmp_bot"
       let jp = [(Nothing, SUBI rtgt $ AddressMacro l_jmp),
                 (Just l_rjmp_top, RBRA l_rjmp_bot),
                 (Just l_jmp, SWAPBR rtgt),
                 (Nothing, NEG rtgt),
                 (Just l_rjmp_bot, BRA l_rjmp_top),
                 (Nothing, ADDI rtgt $ AddressMacro l_jmp)]
       call <- cgCall args jp rt
       popTempRegister >> popTempRegister >> popTempRegister -- rv, rt & rtgt from loadMethodAddr
       popTempRegister >> uo
       let load = lo ++ [(Nothing, XOR rt ro)] ++ loadAddress ++ invertInstructions lo
       return $ load ++ call ++ invertInstructions load

-- | Code generation for object construction
cgObjectConstruction :: TypeName -> (SIdentifier, Maybe SExpression) -> CodeGenerator [(Maybe Label, MInstruction)]
cgObjectConstruction tp (n, e) =
    do (rv, lv, uv) <- case e of
                         Nothing -> loadVariableAddress n
                         Just e' -> loadArrayElementVariableAddress n e'
       rp <- tempRegister
       rt <- tempRegister
       popTempRegister >> popTempRegister
       l  <- getUniqueLabel "obj_con"
       rs <- gets registerStack
       let rr = (registerThis : map snd rs) \\ [rp, rt]
           store = concatMap push rr
           malloc = [(Just l, ADDI rt $ SizeMacro tp)] ++ push rt ++ push rp
           lb = l ++ "_bot"
           setVtable = [(Nothing, XORI rt $ AddressMacro $ "l_" ++ tp ++ "_vt"),
                        (Nothing, EXCH rt rp),
                        (Nothing, ADDI rp ReferenceCounterIndex),
                        (Nothing, XORI rt $ Immediate 1),
                        (Nothing, EXCH rt rp),
                        (Just lb, SUBI rp ReferenceCounterIndex),
                        (Nothing, EXCH rp rv)]
       uv
       return $ store ++ malloc ++ [(Nothing, BRA "l_malloc")] ++ invertInstructions malloc ++ invertInstructions store ++ lv ++ setVtable ++ invertInstructions lv
    where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]

```

```haskell
-- | Code generation for object destruction
cgObjectDestruction :: TypeName -> (SIdentifier, Maybe SExpression) -> CodeGenerator [(Maybe
    Label, MInstruction)]
cgObjectDestruction tp (n, e) =
    do (rp, la, ua) <- case e of
                         Nothing -> loadVariableValue n
                         Just e' -> loadArrayElementVariableValue n e'
       rt <- tempRegister
       l  <- getUniqueLabel "obj_des"
       popTempRegister >> ua
       rs <- gets registerStack
       let removeVtable = [(Just lt, EXCH rt rp),
                           (Nothing, XORI rt $ AddressMacro $ "l_" ++ tp ++ "_vt"),
                           (Nothing, ADDI rp ReferenceCounterIndex),
                           (Nothing, EXCH rt rp),
                           (Nothing, XORI rt $ Immediate 1),
                           (Nothing, SUBI rp ReferenceCounterIndex)]
           rr = (registerThis : map snd rs) \\ [rp, rt]
           store = concatMap push rr
           free = [(Just l, ADDI rt $ SizeMacro tp)] ++ push rt ++ push rp
           lt = l ++ "_top"
       return $ la ++ removeVtable ++ store ++ free ++ [(Nothing, RBRA "l_malloc")] ++
           invertInstructions (la ++ store ++ free)
    where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]

-- | Code generation for reference construction
cgCopyReference :: (SIdentifier, Maybe SExpression) -> (SIdentifier, Maybe SExpression) ->
    CodeGenerator [(Maybe Label, MInstruction)]
cgCopyReference (n, e1) (m, e2) =
    do (rcp, lp, up) <- case e2 of
                          Nothing -> loadVariableValue m
                          Just e2' -> loadArrayElementVariableValue m e2'
       (rp, la, ua) <- case e1 of
                         Nothing -> loadVariableValue n
                         Just e1' -> loadArrayElementVariableValue n e1'
       rt <- tempRegister
       up >> ua >> popTempRegister
       l <- getUniqueLabel "copy"
       let reference = [(Just l, XOR rcp rp),
                        (Nothing, ADDI rp ReferenceCounterIndex),
                        (Nothing, EXCH rt rp),
                        (Nothing, ADDI rt $ Immediate 1),
                        (Nothing, EXCH rt rp),
                        (Nothing, SUBI rp ReferenceCounterIndex)]
       return $ lp ++ la ++ reference ++ invertInstructions (lp ++ la)

-- | Code generation for reference destruction
cgUnCopyReference :: (SIdentifier, Maybe SExpression) -> (SIdentifier, Maybe SExpression) ->
    CodeGenerator [(Maybe Label, MInstruction)]
cgUnCopyReference (n, e1) (m, e2) =
    do (rcp, la1, ua1) <- case e2 of
                            Nothing -> loadVariableValue m
                            Just e2' -> loadArrayElementVariableValue m e2'
       (rp, la2, ua2) <- case e1 of
                           Nothing -> loadVariableValue n
                           Just e1' -> loadArrayElementVariableValue n e1'
       rt <- tempRegister
       l <- getUniqueLabel "uncopy"
       ua1 >> ua2 >> popTempRegister
       let reference = [(Just l, XOR rcp rp),
                        (Nothing, ADDI rp ReferenceCounterIndex),
                        (Nothing, EXCH rt rp),
                        (Nothing, SUBI rt $ Immediate 1),
                        (Nothing, EXCH rt rp),
                        (Nothing, SUBI rp ReferenceCounterIndex)]
    --    removeRegister (m, rcp)
```

```haskell
       return $ la1 ++ la2 ++ reference ++ invertInstructions (la1 ++ la2)

-- | Code generation for array construction
cgArrayConstruction :: SExpression -> SIdentifier -> CodeGenerator [(Maybe Label, MInstruction)]
cgArrayConstruction e n =
    do (ra, la, ua) <- loadVariableAddress n
       (re, le, ue) <- cgExpression e
       rp <- tempRegister
       rt <- tempRegister
       popTempRegister >> popTempRegister
       l  <- getUniqueLabel "arr_con"
       rs <- gets registerStack
       let rr = (registerThis : map snd rs) \\ [rp, rt]
           store = le ++ [(Just l, ADDI rt ArrayElementOffset), (Nothing, ADD rt re)] ++ invertInstructions le ++ concatMap push rr
           malloc = push rt ++ push rp
           lb = l ++ "_bot"
           initArray = la ++ le ++
                       [(Nothing, XOR rt re),
                        (Nothing, EXCH rt rp),
                        (Nothing, ADDI rp ReferenceCounterIndex),
                        (Nothing, XORI rt $ Immediate 1),
                        (Nothing, EXCH rt rp),
                        (Nothing, SUBI rp ReferenceCounterIndex),
                        (Just lb, EXCH rp ra)] ++
                       invertInstructions (la ++ le)
       ue >> ua
       return $ store ++ malloc ++ [(Nothing, BRA "l_malloc")] ++ invertInstructions (store ++ malloc) ++ initArray
    where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]

-- | Code generation for array destruction
cgArrayDestruction :: SExpression -> SIdentifier -> CodeGenerator [(Maybe Label, MInstruction)]
cgArrayDestruction e n =
    do (rp, lp, up) <- loadVariableValue n
       (re, le, ue) <- cgExpression e
       rt <- tempRegister
       l  <- getUniqueLabel "obj_des"
       popTempRegister >> ue >> up
       rs <- gets registerStack
       let removeArray = [(Just lt, EXCH rt rp),
                          (Nothing, XOR rt re),
                          (Nothing, ADDI rp ReferenceCounterIndex),
                          (Nothing, EXCH rt rp),
                          (Nothing, XORI rt $ Immediate 1),
                          (Nothing, SUBI rp ReferenceCounterIndex)]
           rr = (registerThis : map snd rs) \\ [rp, rt]
           store = concatMap push rr
           free = [(Just l, ADDI rt ArrayElementOffset), (Nothing, ADD rt re)] ++ push rt ++ push rp
           lt = l ++ "_top"
       return $ lp ++ le ++ removeArray ++ store ++ free ++ [(Nothing, RBRA "l_malloc")] ++ invertInstructions (lp ++ le ++ store ++ free)
    where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]

-- | Code generation for statements
cgStatement :: SStatement -> CodeGenerator [(Maybe Label, MInstruction)]
cgStatement (Assign n modop e) = cgAssign n modop e
cgStatement (AssignArrElem (n, e1) modop e2) = cgAssignArrElem (n, e1) modop e2
cgStatement (Swap (n1, e1) (n2, e2)) = cgSwap (n1, e1) (n2, e2)
cgStatement (Conditional e1 s1 s2 e2) = cgConditional e1 s1 s2 e2
cgStatement (Loop e1 s1 s2 e2) = cgLoop e1 s1 s2 e2
cgStatement (ObjectBlock tp n stmt) = cgObjectBlock tp n stmt
cgStatement (LocalBlock _ n e1 stmt e2) = cgLocalBlock n e1 stmt e2
```

```
cgStatement (LocalCall m args) = cgLocalCall m args
cgStatement (LocalUncall m args) = cgLocalUncall m args
cgStatement (ObjectCall o m args) = cgObjectCall o m args
cgStatement (ObjectUncall o m args) = cgObjectUncall o m args
cgStatement (ObjectConstruction tp n) = cgObjectConstruction tp n
cgStatement (ObjectDestruction tp n)  = cgObjectDestruction tp n
cgStatement Skip = return []
cgStatement (CopyReference _ n m) = cgCopyReference n m
cgStatement (UnCopyReference _ n m)  = cgUnCopyReference n m
cgStatement (ArrayConstruction (_, e) n) = cgArrayConstruction e n
cgStatement (ArrayDestruction (_, e) n) = cgArrayDestruction e n

-- | Code generation for methods
cgMethod :: (TypeName, SMethodDeclaration) -> CodeGenerator [(Maybe Label, MInstruction)]
cgMethod (_, GMDecl m ps body) =
    do l <- getMethodLabel m
       rs <- addParameters
       body' <- concat <$> mapM cgStatement body
       clearParameters
       let lt = l ++ "_top"
           lb = l ++ "_bot"
           mp = [(Just lt, BRA lb),
                 (Nothing, ADDI registerSP $ Immediate 1),
                 (Nothing, EXCH registerRO registerSP)]
                 ++ concatMap pushParameter rs ++
                [(Nothing, EXCH registerThis registerSP),
                 (Nothing, SUBI registerSP $ Immediate 1),
                 (Just l, SWAPBR registerRO),
                 (Nothing, NEG registerRO),
                 (Nothing, ADDI registerSP $ Immediate 1),
                 (Nothing, EXCH registerThis registerSP)]
                 ++ invertInstructions (concatMap pushParameter rs) ++
                [(Nothing, EXCH registerRO registerSP),
                 (Nothing, SUBI registerSP $ Immediate 1)]
       return $ mp ++ body' ++ [(Just lb, BRA lt)]
    where addParameters = mapM (pushRegister . (\(GDecl _ p) -> p)) ps
          clearParameters = replicateM_ (length ps) popRegister
          pushParameter r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $
              Immediate 1)]

cgMalloc1 :: CodeGenerator [(Maybe Label, MInstruction)]
cgMalloc1 =
    do -- Temp registers needed for malloc
       r_p <- tempRegister -- Pointer to new obj
       r_object_size <- tempRegister -- Object size
       r_counter  <- tempRegister -- Free list index
       r_csize  <- tempRegister -- Current cell size
       rt <- tempRegister
       rt2 <- tempRegister
       r_tmp <- tempRegister

       -- Expressions and sub routines
       (r_e1_outer, l_e1_outer, u_e1_outer) <- cgBinOp Lt r_csize r_object_size
       (r_e2_outer, l_e2_outer, u_e2_outer) <- cgBinOp Lt r_csize r_object_size
       (r_fl, l_fl, u_fl) <- loadFreeListAddress r_counter
       (r_block, l_block, u_block) <- loadHeadAtFreeList r_fl
       (r_e1_inner, l_e1_inner, u_e1_inner) <- cgBinOp Neq r_block registerZero
       (r_e2_i1, l_e2_i1, u_e2_i1) <- cgBinOp Neq r_p r_tmp
       (r_e2_i2, l_e2_i2, u_e2_i2) <- cgBinOp Eq r_tmp registerZero
       (r_e2_i3, l_e2_i3, u_e2_i3) <- cgBinOp BitOr r_e2_i1 r_e2_i2

       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]

       -- Update state after evaluating expressions and subroutines
       u_e2_i3 >> u_e2_i2 >> u_e2_i1 >> u_e1_inner
```

```
    u_block >> u_fl >> u_e2_outer >> u_e1_outer
    popTempRegister >> popTempRegister >> popTempRegister >> popTempRegister >> popTempRegister >> popTempRegister >> popTempRegister

    let l_o_test = "l_o_test"
        l_o_assert_t = "l_o_assert_true"
        l_o_test_f = "l_o_test_false"
        l_o_assert = "l_o_assert"
        l_i_test = "l_i_test"
        l_i_assert_t = "l_i_assert_true"
        l_i_test_f = "l_i_test_false"
        l_i_assert = "l_i_assert"
        l_m_top = "l_malloc1_top"
        l_m_bot = "l_malloc1_bot"
        l_m_entry = "l_malloc1"
        malloc = [(Just l_m_top, BRA l_m_bot),                --
                  (Nothing, ADDI registerSP $ Immediate 1),
                  (Nothing, EXCH registerRO registerSP)]      -- Pop return offset from stack
                  ++ invertInstructions l_fl ++
                 [(Just l_m_entry, SWAPBR registerRO),        -- Malloc1 entry/exit point
                  (Nothing, NEG registerRO),                  -- Restore return offset
                  (Nothing, EXCH registerRO registerSP),      -- Push return offset to stack
                  (Nothing, SUBI registerSP $ Immediate 1)]
                 ++ l_fl
                 ++ l_block
                 ++ l_e1_outer                                -- Set r_e1 -> c_size < obj_size
                 ++ [(Nothing, XOR rt r_e1_outer)]            -- r_t = r_e1_o
                 ++ invertInstructions l_e1_outer ++          -- Clear r_e1_o
                 [(Just l_o_test, BEQ rt registerZero l_o_test_f),
                  (Nothing, XORI rt $ Immediate 1),           -- S1_outer start
                  (Nothing, ADDI r_counter $ Immediate 1)]    -- counter++
                  ++ invertInstructions l_block ++
                  [(Nothing, RL r_csize $ Immediate 1)]        -- call double(csize)
                 ++ concatMap pushRegisterToStack tmpRegisterList
                 ++
                 [(Nothing, BRA l_m_entry)]                   -- call malloc1()
                 ++ invertInstructions(concatMap pushRegisterToStack tmpRegisterList)
                 ++
                 [(Nothing, RR r_csize $ Immediate 1),        -- uncall double(csize)
                  (Nothing, SUBI r_counter $ Immediate 1),    -- counter++
                  (Nothing, XORI rt $ Immediate 1),           -- S1_outer end
                  (Just l_o_assert_t, BRA l_o_assert),
                  (Just l_o_test_f, BRA l_o_test)]
                  ++ l_e1_inner ++                            -- Set r_e1_i -> r_block != 0 (S2_OUTER)
                 [(Nothing, XOR rt2 r_e1_inner)]              -- Set rt2 -> r_e1_i
                  ++ invertInstructions l_e1_inner ++         -- Clear r_e1_i
                 [(Just l_i_test, BEQ rt2 registerZero l_i_test_f),
                  (Nothing, XORI rt2 $ Immediate 1),          -- S1_inner start
                  (Nothing, ADD r_p r_block),                 -- Set address of p to said block
                  (Nothing, SUB r_block r_p),                 -- Clear r_block
                  (Nothing, EXCH r_tmp r_p),                  -- Load address of next block
                  (Nothing, EXCH r_tmp r_fl),                 -- Set address of next block as head of current free list
                  (Nothing, XOR r_tmp r_p),                   -- Clear address of next block
                  (Nothing, XORI rt2 $ Immediate 1),          -- S1_inner end
                  (Just l_i_assert_t, BRA l_i_assert),
                  (Just l_i_test_f, BRA l_i_test),
                  (Nothing, ADDI r_counter $ Immediate 1),    -- S2_inner start
                  (Nothing, RL r_csize $ Immediate 1)]        -- call double(csize)
                 ++ concatMap pushRegisterToStack tmpRegisterList ++
                 [(Nothing, BRA l_m_entry)]                   -- call malloc1()
                 ++ invertInstructions(concatMap pushRegisterToStack tmpRegisterList) ++
```

```
                    [(Nothing, RR r_csize $ Immediate 1),       -- uncall double(csize)
                     (Nothing, SUBI r_counter $ Immediate 1),   -- counter -= 1
                     (Nothing, XOR r_tmp r_p),                  -- Copy current address of p
                     (Nothing, EXCH r_tmp r_fl),                -- Store address in current
                         free list
                     (Nothing, ADD r_p r_csize),                -- Set p to other half of the
                         block we're splitting
                     (Just l_i_assert, BNE rt2 registerZero l_i_assert_t),
                     (Nothing, EXCH r_tmp r_fl),
                     (Nothing, SUB r_p r_csize)]
                     ++ l_e2_i1                                 -- set r_e2_i1 <- p - csize !=
                          free_list[counter]
                     ++ l_e2_i2                                 -- set r_e2_i2 <- free_list[
                         counter] = 0
                     ++ l_e2_i3                                 -- set r_e2_i3 <- r_e2_i1 ||
                         Â r_e2_i2
                     ++ [(Nothing, XOR rt2 r_e2_i3)]            -- Set rt2 -> r_i_2
                     ++ invertInstructions l_e2_i3              -- Clear r_i_2
                     ++ invertInstructions l_e2_i2
                     ++ invertInstructions l_e2_i1 ++
                     [(Nothing, ADD r_p r_csize),
                      (Nothing, EXCH r_tmp r_fl),               -- S2_outer end
                      (Just l_o_assert, BNE rt registerZero l_o_assert_t)]
                     ++ l_e2_outer                              -- Set r_e2 -> c_size <
                         obj_size
                     ++ [(Nothing, XOR rt r_e2_outer)]          -- r_t = r_e1_o
                     ++ invertInstructions l_e2_outer           -- Clear r_e1_o
                     ++ [(Just l_m_bot, BRA l_m_top)]           -- Go to top
       return malloc
    where pushRegisterToStack r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $
          Immediate 1)]

cgMalloc :: CodeGenerator [(Maybe Label, MInstruction)]
cgMalloc =
    do rp <- tempRegister -- Pointer to new obj
       ros <- tempRegister -- Object size
       rc  <- tempRegister -- Free list index
       rs  <- tempRegister -- Current cell size
       popTempRegister >> popTempRegister >> popTempRegister >> popTempRegister
       let malloc = [(Just "l_malloc_top", BRA "l_malloc_bot")]
                    ++
                    [(Just "l_malloc", SWAPBR registerRO),
                     (Nothing, NEG registerRO),
                     (Nothing, ADDI rs $ Immediate 2),
                     (Nothing, XOR rc registerZero)]
                    ++ concatMap pop [rp, ros]
                    ++ push registerRO ++
                    [(Nothing, BRA "l_malloc1")]
                    ++ pop registerRO
                    ++ concatMap push [ros, rp] ++
                    [(Nothing, XOR rc registerZero),
                     (Nothing, SUBI rs $ Immediate 2),
                     (Just "l_malloc_bot", BRA "l_malloc_top")]
       return malloc
    where pop r = [(Nothing, ADDI registerSP $ Immediate 1), (Nothing, EXCH r registerSP)]
          push r = invertInstructions (pop r)

-- | Code generation for virtual tables
cgVirtualTables :: CodeGenerator [(Maybe Label, MInstruction)]
cgVirtualTables = concat <$> (gets (virtualTables . saState) >>= mapM vtInstructions)
    where vtInstructions (n, ms) = zip (vtLabel n) <$> mapM vtData ms
          vtData m = DATA . AddressMacro <$> getMethodLabel m
          vtLabel n = (Just $ "l_" ++ n ++ "_vt") : repeat Nothing

-- | Returns the main class label
getMainLabel :: CodeGenerator Label
```

```haskell
getMainLabel = gets (mainMethod . saState) >>= getMethodLabel

-- | Fetches the main class from the class analysis state
getMainClass :: CodeGenerator TypeName
getMainClass = gets (mainClass . caState . saState) >>= \mc ->
    case mc of
        (Just tp) -> return tp
        Nothing -> throwError "ICE: No main method defined"

-- | Fetches the field of a given type name
getFields :: TypeName -> CodeGenerator [VariableDeclaration]
getFields tp =
    do cs <- gets (classes . caState . saState)
       case lookup tp cs of
           (Just (GCDecl _ _ fs _)) -> return fs
           Nothing -> throwError $ "ICE: Unknown class " ++ tp

-- | Code generation for output
cgOutput :: TypeName -> CodeGenerator ([(Maybe Label, MInstruction)], [(Maybe Label, MInstruction)])
cgOutput tp =
    do mfs <- getFields tp
       co <- concat <$> mapM cgCopyOutput (zip [1..] mfs)
       return (map cgStatic mfs, co)
    where cgStatic (GDecl _ n) = (Just $ "l_r_" ++ n, DATA $ Immediate 0)
          cgCopyOutput(o, GDecl _ n) =
              do rt <- tempRegister
                 ra <- tempRegister
                 popTempRegister >> popTempRegister
                 let copy = [ADDI registerThis ReferenceCounterIndex,
                             ADDI registerThis $ Immediate o,
                             EXCH rt registerThis,
                             XORI ra $ AddressMacro $ "l_r_" ++ n,
                             EXCH rt ra,
                             XORI ra $ AddressMacro $ "l_r_" ++ n,
                             SUBI registerThis $ Immediate o,
                             SUBI registerThis ReferenceCounterIndex]
                 return $ zip (repeat Nothing) copy

-- | Generates code for the program entry point
cgProgram :: SProgram -> CodeGenerator PISA.MProgram
cgProgram p =
    do vt <- cgVirtualTables
       malloc <- cgMalloc
       malloc1 <- cgMalloc1
       rv <- tempRegister -- V table register
       rb <- tempRegister -- Memory block register
       popTempRegister >> popTempRegister
       ms <- concat <$> mapM cgMethod p
       l_main <- getMainLabel
       mtp <- getMainClass
       (out, co) <- cgOutput mtp
       let mvt = "l_" ++ mtp ++ "_vt"
           mn = [(Just "start", BRA "top"),
                 (Nothing, START),
                 (Nothing, ADDI registerFLPs ProgramSize),    -- Init free list pointer list
                 (Nothing, XOR registerHP registerFLPs),      -- Init heap pointer
                 (Nothing, ADDI registerHP FreeListsSize),    -- Init space for FLPs list
                 (Nothing, XOR rb registerHP),                -- Store address of initial memory block in rb
                 (Nothing, ADDI registerFLPs FreeListsSize),  -- Index to end of free lists
                 (Nothing, SUBI registerFLPs $ Immediate 1),  -- Index to last element of free lists
                 (Nothing, EXCH rb registerFLPs),             -- Store address of first block in last element of free lists
                 (Nothing, ADDI registerFLPs $ Immediate 1),  -- Index to end of free lists
```

```
                     (Nothing, SUBI registerFLPs FreeListsSize),  -- Index to beginning of free
                         lists
                     (Nothing, XOR registerSP registerHP),         -- Init stack pointer 1/2
                     (Nothing, ADDI registerSP StackOffset),       -- Init stack pointer 2/2
                     (Nothing, SUBI registerSP $ SizeMacro mtp),   -- Allocate space for main on
                         stack
                     (Nothing, XOR registerThis registerSP),       -- Store address of main object
                     (Nothing, XORI rv $ AddressMacro mvt),        -- Store address of vtable in rv
                     (Nothing, EXCH rv registerThis),              -- Add address of vtable to
                         stack
                     (Nothing, SUBI registerSP $ Immediate 1),     -- Add address of vtable to
                         stack
                     (Nothing, EXCH registerThis registerSP),      -- Push 'this' to stack
                     (Nothing, SUBI registerSP $ Immediate 1),     -- Push 'this' to stack
                     (Nothing, BRA l_main),                        -- Execute main
                     (Nothing, ADDI registerSP $ Immediate 1),     -- Pop 'this'
                     (Nothing, EXCH registerThis registerSP)]      -- Pop 'this'
                      ++ co ++
                    [(Nothing, ADDI registerSP $ Immediate 1),     -- Pop vtable address
                     (Nothing, EXCH rv registerThis),              -- Pop vtable address
                     (Nothing, XORI rv $ AddressMacro mvt),        -- Clear rv
                     (Nothing, XOR registerThis registerSP),       -- Clear 'this'
                     (Nothing, ADDI registerSP $ SizeMacro mtp),   -- Deallocate space for main
                     (Nothing, SUBI registerSP StackOffset),       -- Clear stack pointer
                     (Nothing, XOR registerSP registerHP),         -- Clear stack pointer
                     (Nothing, SUBI registerHP FreeListsSize),     -- Reset Heap pointer (For
                         pretty output)
                     (Nothing, XOR registerHP registerFLPs),       -- Reset Heap pointer (For
                         pretty output)
                     (Nothing, SUBI registerFLPs ProgramSize),     -- Reset Free lists pointer (For
                          pretty output)
                     (Just "finish", FINISH)]
        return $ PISA.GProg $ [(Just "top", BRA "start")] ++ out ++ vt ++ malloc ++ malloc1 ++
            ms ++ mn


-- | Generates code for a program
generatePISA :: (SProgram, SAState) -> Except String (PISA.MProgram, SAState)
generatePISA (p, s) = second saState <$> runStateT (runCG $ cgProgram p) (initialState s)

showPISAProgram :: (Show a, MonadIO m) => (t, a) -> m ()
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

```

```haskell
-- | Returns the address of a given label
getAddress :: Label -> MacroExpander Address
getAddress l = asks addressTable >>= \at ->
    case lookup l at of
        (Just i) -> return i
        Nothing -> throwError $ "ICE: Unknown label " ++ l

-- | Returns the size of a given class name
getSize :: TypeName -> MacroExpander Size
getSize tn = asks sizeTable >>= \st ->
    case lookup tn st of
        (Just s) -> return s
        Nothing -> throwError $ "ICE: Unknown type " ++ tn

-- | Returns the off set of a method for a given class name and method
getOffset :: TypeName -> MethodName -> MacroExpander Offset
getOffset tn mn = asks offsetTable >>= \ot ->
    case lookup tn ot of
        Nothing -> throwError $ "ICE: Unknown type " ++ tn
        (Just mo) -> case lookup mn mo of
                        Nothing -> throwError $ "ICE: Unknown method " ++ mn
                        (Just o) -> return o

-- | Macro definitions
meMacro :: Macro -> MacroExpander Integer
meMacro (Immediate i) = return i
meMacro (AddressMacro l) = getAddress l
meMacro (SizeMacro tn) = getSize tn
meMacro (OffsetMacro tn mn) = getOffset tn mn
meMacro ProgramSize = asks programSize
meMacro FreeListsSize = asks freeListsSize
meMacro StackOffset = asks stackOffset
meMacro InitialMemoryBlockSize = asks initialMemoryBlockSize
meMacro ReferenceCounterIndex = asks referenceCounterIndex
meMacro ArrayElementOffset = asks arrayElementOffset

-- | Macro instructions
meInstruction :: MInstruction -> MacroExpander Instruction
meInstruction (ADD r1 r2) = return $ ADD r1 r2
meInstruction (ADDI r m) = ADDI r <$> meMacro m
meInstruction (ANDX r1 r2 r3) = return $ ANDX r1 r2 r3
meInstruction (ANDIX r1 r2 m) = ANDIX r1 r2 <$> meMacro m
meInstruction (NORX r1 r2 r3) = return $ NORX r1 r2 r3
meInstruction (NEG r) = return $ NEG r
meInstruction (ORX r1 r2 r3) = return $ ORX r1 r2 r3
meInstruction (ORIX r1 r2 m) = ORIX r1 r2 <$> meMacro m
meInstruction (RL r m) = RL r <$> meMacro m
meInstruction (RLV r1 r2 ) = return $ RLV r1 r2
meInstruction (RR r m) = RR r <$> meMacro m
meInstruction (RRV r1 r2 ) = return $ RRV r1 r2
meInstruction (SLLX r1 r2 m) = SLLX r1 r2 <$> meMacro m
meInstruction (SLLVX r1 r2 r3) = return $ SLLVX r1 r2 r3
meInstruction (SRAX r1 r2 m) = SRAX r1 r2 <$> meMacro m
meInstruction (SRAVX r1 r2 r3) = return $ SRAVX r1 r2 r3
meInstruction (SRLX r1 r2 m) = SRLX r1 r2 <$> meMacro m
meInstruction (SRLVX r1 r2 r3) = return $ SRLVX r1 r2 r3
meInstruction (SUB r1 r2) = return $ SUB r1 r2
meInstruction (XOR r1 r2) = return $ XOR r1 r2
meInstruction (XORI r m) = XORI r <$> meMacro m
meInstruction (BEQ r1 r2 l) = return $ BEQ r1 r2 l
meInstruction (BGEZ r l) = return $ BGEZ r l
meInstruction (BGTZ r l) = return $ BGTZ r l
meInstruction (BLEZ r l) = return $ BLEZ r l
meInstruction (BLTZ r l) = return $ BLTZ r l
meInstruction (BNE r1 r2 l) = return $ BNE r1 r2 l
meInstruction (BRA l) = return $ BRA l
```

```haskell
meInstruction (EXCH r1 r2) = return $ EXCH r1 r2
meInstruction (SWAPBR r) = return $ SWAPBR r
meInstruction (RBRA l) = return $ RBRA l
meInstruction START = return START
meInstruction FINISH = return FINISH
meInstruction (DATA m) = DATA <$> meMacro m
meInstruction (SUBI r m) = SUBI r <$> meMacro m

-- | Macro Expand a program
meProgram :: MProgram -> MacroExpander Program
meProgram (GProg p) = GProg <$> mapM expandPair p
    where expandPair (l, i) = (,) l <$> meInstruction i

-- | Interface for starting the macro extension process
expandMacros :: (MProgram, SAState) -> Except String Program
expandMacros (p, s) = runReaderT (runME $ meProgram p) $ initialState p s
```

ROOPLPPC.hs

```
import Control.Monad.Except
import System.IO
import System.Environment

import PISA
import Parser
import ClassAnalyzer
import ScopeAnalyzer
import TypeChecker
import CodeGenerator
import MacroExpander

import Data.List.Split

type Error = String

main :: IO ()
main =
    do args <- getArgs
       when (null args) (error "Supply input filename.\nUsage: ROOPLPPC input.rplpp output.pal
           \n")
       when (length args > 2) (error "Too many arguments.\nUsage: ROOPLPPC input.rplpp output.
           pal\n")
       handle <- openFile (head args) ReadMode
       input <- hGetContents handle
       let output =  if length args == 2 then last args else head (splitOn "." (head args)) ++
            ".pal"
       either (hPutStrLn stderr) (writeProgram output) $ compileProgram input

compileProgram :: String -> Either Error PISA.Program
compileProgram s =
    runExcept $
    parseString s
    >>= classAnalysis
    >>= scopeAnalysis
    >>= typeCheck
    >>= generatePISA
    >>= 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
```

```
        else skip
        fi cell != nil & head = nil

        if cell != nil & head = nil then
            cell <=> head               // Set head = cell. Cell is nil after execution
        else skip
        fi cell = nil & head != nil

        listLength += 1                 // Increment length

    method length(int result)
        result ^= listLength

class Program
    LinkedList linkedList
    int sumResult
    int listLength

    method main()
        new LinkedList linkedList           // Init new linked linkedList
        listLength += 10

        local int x = 0
        from x = 0 do
            skip
        loop
            local Cell cell = nil
            new Cell cell                       // Instantiate new cell
            call cell::constructor(x)           // Set value of cell
            call linkedList::appendCell(cell)   // Append it to the linkedList
            delocal Cell cell = nil
            x += 1
        until x = listLength
        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
```

```
                        XORI   $10 1
cmp_top_28_i:           BEQ    $9 $0 cmp_bot_29_i
                        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
cmp_top_38:             BEQ    $9 $0 cmp_bot_39
                        XORI   $10 1
cmp_bot_39:             BEQ    $9 $0 cmp_top_38
f_top_40:               BEQ    $10 $0 f_bot_41
                        XORI   $11 1
f_bot_41:               BEQ    $10 $0 f_top_40
                        XOR    $7 $11
f_bot_41_i:             BEQ    $10 $0 f_top_40_i
                        XORI   $11 1
f_top_40_i:             BEQ    $10 $0 f_bot_41_i
cmp_bot_39_i:           BEQ    $9 $0 cmp_top_38_i
                        XORI   $10 1
cmp_top_38_i:           BEQ    $9 $0 cmp_bot_39_i
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
test_34:                BEQ    $7 $0 test_false_36
                        XORI   $7 1
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
                        XOR    $10 $9
loadMetAdd_42:          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 -397
l_jmp_43:               SWAPBR $13
                        NEG    $13
                        ADDI   $13 397
                        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_42_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_35:         BRA    assert_37
test_false_36:          BRA    test_34
assert_37:              BNE    $7 $0 assert_true_35
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
cmp_top_44:             BEQ    $9 $0 cmp_bot_45
                        XORI   $10 1
cmp_bot_45:             BEQ    $9 $0 cmp_top_44
f_top_46:               BEQ    $10 $0 f_bot_47
                        XORI   $11 1
f_bot_47:               BEQ    $10 $0 f_top_46
                        XOR    $7 $11
f_bot_47_i:             BEQ    $10 $0 f_top_46_i
                        XORI   $11 1
f_top_46_i:             BEQ    $10 $0 f_bot_47_i
cmp_bot_45_i:           BEQ    $9 $0 cmp_top_44_i
                        XORI   $10 1
cmp_top_44_i:           BEQ    $9 $0 cmp_bot_45_i
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
l_append_6_bot:         BRA    l_append_6_top
l_insertHead_1_top:     BRA
```

```
    l_insertHead_1_bot
                          ADDI   $1 1
                          EXCH   $2 $1
                          EXCH   $6 $1
                          ADDI   $1 -1
                          EXCH   $3 $1
                          ADDI   $1 -1
l_insertHead_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_52:               BNE    $9 $0
    cmp_bot_53
                          XORI   $10 1
cmp_bot_53:               BNE    $9 $0
    cmp_top_52
                          EXCH   $11 $6
cmp_top_54:               BEQ    $11 $0
    cmp_bot_55
                          XORI   $12 1
cmp_bot_55:               BEQ    $11 $0
    cmp_top_54
                          ANDX   $13 $10 $12
f_top_56:                 BEQ    $13 $0
    f_bot_57
                          XORI   $14 1
f_bot_57:                 BEQ    $13 $0
    f_top_56
                          XOR    $7 $14
f_bot_57_i:               BEQ    $13 $0
    f_top_56_i
                          XORI   $14 1
f_top_56_i:               BEQ    $13 $0
    f_bot_57_i
                          ANDX   $13 $10 $12
cmp_bot_55_i:             BEQ    $11 $0
    cmp_top_54_i
                          XORI   $12 1
cmp_top_54_i:             BEQ    $11 $0
    cmp_bot_55_i
                          EXCH   $11 $6
cmp_bot_53_i:             BNE    $9 $0
    cmp_top_52_i
                          XORI   $10 1
cmp_top_52_i:             BNE    $9 $0
    cmp_bot_53_i
                          ADD    $8 $3
                          ADDI   $8 2
                          EXCH   $9 $8
                          ADDI   $8 -2
                          SUB    $8 $3
test_48:                  BEQ    $7 $0
    test_false_50
                          XORI   $7 1
                          ADD    $8 $3
                          ADDI   $8 2
                          EXCH   $9 $8
                          ADDI   $8 -2
                          SUB    $8 $3
                          EXCH   $10 $6
swap_58:                  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_49:           BRA    assert_51
test_false_50:            BRA    test_48
assert_51:                BNE    $7 $0
    assert_true_49
                          ADD    $8 $3
                          ADDI   $8 2
                          EXCH   $9 $8
                          ADDI   $8 -2
                          SUB    $8 $3
cmp_top_59:               BEQ    $9 $0
    cmp_bot_60
                          XORI   $10 1
cmp_bot_60:               BEQ    $9 $0
    cmp_top_59
                          EXCH   $11 $6
cmp_top_61:               BNE    $11 $0
    cmp_bot_62
                          XORI   $12 1
cmp_bot_62:               BNE    $11 $0
    cmp_top_61
                          ANDX   $13 $10 $12
f_top_63:                 BEQ    $13 $0
    f_bot_64
                          XORI   $14 1
f_bot_64:                 BEQ    $13 $0
    f_top_63
                          XOR    $7 $14
f_bot_64_i:               BEQ    $13 $0
    f_top_63_i
                          XORI   $14 1
f_top_63_i:               BEQ    $13 $0
    f_bot_64_i
                          ANDX   $13 $10 $12
cmp_bot_62_i:             BNE    $11 $0
    cmp_top_61_i
                          XORI   $12 1
cmp_top_61_i:             BNE    $11 $0
    cmp_bot_62_i
                          EXCH   $11 $6
cmp_bot_60_i:             BEQ    $9 $0
    cmp_top_59_i
                          XORI   $10 1
cmp_top_59_i:             BEQ    $9 $0
    cmp_bot_60_i
                          ADD    $8 $3
                          ADDI   $8 2
                          EXCH   $9 $8
                          ADDI   $8 -2
                          SUB    $8 $3
l_insertHead_1_bot:       BRA
    l_insertHead_1_top
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
```

```
                         XORI   $11 1
f_top_77_i:              BEQ    $10 $0
    f_bot_78_i
cmp_bot_76_i:            BEQ    $9 $0
    cmp_top_75_i
                         XORI   $10 1
cmp_top_75_i:            BEQ    $9 $0
    cmp_bot_76_i
                         ADD    $8 $3
                         ADDI   $8 2
                         EXCH   $9 $8
                         ADDI   $8 -2
                         SUB    $8 $3
                         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_appendCell_2_bot:      BRA
    l_appendCell_2_top
l_prependCell_3_top:     BRA
    l_prependCell_3_bot
                         ADDI   $1 1
                         EXCH   $2 $1
                         EXCH   $6 $1
                         ADDI   $1 -1
                         EXCH   $3 $1
                         ADDI   $1 -1
l_prependCell_3:         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_1
                         ADDI   $1 1
                         EXCH   $3 $1
                         ADDI   $1 1
                         EXCH   $6 $1
                         EXCH   $8 $6
cmp_top_83:              BEQ    $8 $0
    cmp_bot_84
                         XORI   $9 1
cmp_bot_84:              BEQ    $8 $0
    cmp_top_83
                         ADD    $10 $3
                         ADDI   $10 2
                         EXCH   $11 $10
                         ADDI   $10 -2
                         SUB    $10 $3
cmp_top_85:              BEQ    $11 $0
    cmp_bot_86
                         XORI   $12 1
cmp_bot_86:              BEQ    $11 $0
    cmp_top_85
                         ANDX   $13 $9 $12
f_top_87:                BEQ    $13 $0
    f_bot_88
                         XORI   $14 1
f_bot_88:                BEQ    $13 $0
    f_top_87
                         XOR    $7 $14
f_bot_88_i:              BEQ    $13 $0
    f_top_87_i
                         XORI   $14 1
f_top_87_i:              BEQ    $13 $0
    f_bot_88_i
                         ANDX   $13 $9 $12
cmp_bot_86_i:            BEQ    $11 $0
    cmp_top_85_i
                         XORI   $12 1
cmp_top_85_i:            BEQ    $11 $0
    cmp_bot_86_i
                         ADD    $10 $3
                         ADDI   $10 2
                         EXCH   $11 $10
                         ADDI   $10 -2
                         SUB    $10 $3
cmp_bot_84_i:            BEQ    $8 $0
    cmp_top_83_i
                         XORI   $9 1
cmp_top_83_i:            BEQ    $8 $0
    cmp_bot_84_i
                         EXCH   $8 $6
test_79:                 BEQ    $7 $0
    test_false_81
                         XORI   $7 1
                         EXCH   $8 $6
                         XOR    $9 $8
loadMetAdd_89:           EXCH   $10 $9
                         ADDI   $10 1
                         EXCH   $11 $10
                         XOR    $12 $11
                         EXCH   $11 $10
                         ADDI   $10 -1
                         EXCH   $10 $9
                         EXCH   $8 $6
                         ADD    $13 $3
                         ADDI   $13 2
                         EXCH   $3 $1
                         ADDI   $1 -1
                         EXCH   $6 $1
                         ADDI   $1 -1
                         EXCH   $13 $1
                         ADDI   $1 -1
                         EXCH   $9 $1
                         ADDI   $1 -1
                         ADDI   $12 -777
l_jmp_90:                SWAPBR $12
                         NEG    $12
                         ADDI   $12 777
                         ADDI   $1 1
                         EXCH   $9 $1
                         ADDI   $1 1
                         EXCH   $13 $1
                         ADDI   $1 1
                         EXCH   $6 $1
                         ADDI   $1 1
```

```
                                    EXCH   $3 $1
                                    ADDI   $13 -2
                                    SUB    $13 $3
                                    EXCH   $8 $6
                                    EXCH   $10 $9
                                    ADDI   $10 1
                                    EXCH   $11 $10
                                    XOR    $12 $11
                                    EXCH   $11 $10
                                    ADDI   $10 -1
loadMetAdd_89_i:                    EXCH   $10 $9
                                    XOR    $9 $8
                                    EXCH   $8 $6
                                    XORI   $7 1
assert_true_80:                     BRA    assert_82
test_false_81:                      BRA    test_79
assert_82:                          BNE    $7 $0
    assert_true_80
                                    EXCH   $8 $6
cmp_top_91:                         BEQ    $8 $0
    cmp_bot_92
                                    XORI   $9 1
cmp_bot_92:                         BEQ    $8 $0
    cmp_top_91
                                    ADD    $10 $3
                                    ADDI   $10 2
                                    EXCH   $11 $10
                                    ADDI   $10 -2
                                    SUB    $10 $3
cmp_top_93:                         BNE    $11 $0
    cmp_bot_94
                                    XORI   $12 1
cmp_bot_94:                         BNE    $11 $0
    cmp_top_93
                                    ANDX   $13 $9 $12
f_top_95:                           BEQ    $13 $0
    f_bot_96
                                    XORI   $14 1
f_bot_96:                           BEQ    $13 $0
    f_top_95
                                    XOR    $7 $14
f_bot_96_i:                         BEQ    $13 $0
    f_top_95_i
                                    XORI   $14 1
f_top_95_i:                         BEQ    $13 $0
    f_bot_96_i
                                    ANDX   $13 $9 $12
cmp_bot_94_i:                       BNE    $11 $0
    cmp_top_93_i
                                    XORI   $12 1
cmp_top_93_i:                       BNE    $11 $0
    cmp_bot_94_i
                                    ADD    $10 $3
                                    ADDI   $10 2
                                    EXCH   $11 $10
                                    ADDI   $10 -2
                                    SUB    $10 $3
cmp_bot_92_i:                       BEQ    $8 $0
    cmp_top_91_i
                                    XORI   $9 1
cmp_top_91_i:                       BEQ    $8 $0
    cmp_bot_92_i
                                    EXCH   $8 $6
                                    EXCH   $8 $6
cmp_top_101:                        BEQ    $8 $0
    cmp_bot_102
                                    XORI   $9 1
cmp_bot_102:                        BEQ    $8 $0
    cmp_top_101
                                    ADD    $10 $3
                                    ADDI   $10 2
                                    EXCH   $11 $10
                                    ADDI   $10 -2
                                    SUB    $10 $3
cmp_top_103:                        BNE    $11 $0
    cmp_bot_104
                                    XORI   $12 1
cmp_bot_104:                        BNE    $11 $0
    cmp_top_103
                                    ANDX   $13 $9 $12
f_top_105:                          BEQ    $13 $0
    f_bot_106
                                    XORI   $14 1
f_bot_106:                          BEQ    $13 $0
    f_top_105
                                    XOR    $7 $14
f_bot_106_i:                        BEQ    $13 $0
    f_top_105_i
                                    XORI   $14 1
f_top_105_i:                        BEQ    $13 $0
    f_bot_106_i
                                    ANDX   $13 $9 $12
cmp_bot_104_i:                      BNE    $11 $0
    cmp_top_103_i
                                    XORI   $12 1
cmp_top_103_i:                      BNE    $11 $0
    cmp_bot_104_i
                                    ADD    $10 $3
                                    ADDI   $10 2
                                    EXCH   $11 $10
                                    ADDI   $10 -2
                                    SUB    $10 $3
cmp_bot_102_i:                      BEQ    $8 $0
    cmp_top_101_i
                                    XORI   $9 1
cmp_top_101_i:                      BEQ    $8 $0
    cmp_bot_102_i
                                    EXCH   $8 $6
test_97:                            BEQ    $7 $0
    test_false_99
                                    XORI   $7 1
                                    EXCH   $8 $6
                                    ADD    $9 $3
                                    ADDI   $9 2
                                    EXCH   $10 $9
                                    ADDI   $9 -2
                                    SUB    $9 $3
swap_107:                           XOR    $8 $10
                                    XOR    $10 $8
                                    XOR    $8 $10
                                    ADD    $9 $3
                                    ADDI   $9 2
                                    EXCH   $10 $9
                                    ADDI   $9 -2
                                    SUB    $9 $3
                                    EXCH   $8 $6
                                    XORI   $7 1
assert_true_98:                     BRA    assert_100
test_false_99:                      BRA    test_97
assert_100:                         BNE    $7 $0
    assert_true_98
                                    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
```

```
                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 -1279
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
```

```
            root <=> node
        else skip
        fi root != nil & node = nil

        if root != nil then
            call root::insertNode(node, value)
        else skip
        fi root != nil

    method sum(int result)
        if root != nil then
            call root::getSum(result)
        else skip
        fi root != nil

    method mirror()
        if root != nil then
            call root::mirror()
        else skip
        fi root != nil

class Program
    int sumResult
    Tree tree
    int nodeCount

    method main()
        new Tree tree
        nodeCount += 3

        local int x = 0
        from x = 0 do
            skip
        loop
            local Node node = nil
            new Node node                   // Init new node
            call node::setValue(x)          // Set node value
            call tree::insertNode(node, x)  // Insert node in tree
            delocal Node node = nil
            x += 1
        until x = nodeCount
        delocal int x = nodeCount

        call tree::sum(sumResult)
        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
```

```
                        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_setValue_4_top:       BRA     l_setValue_4_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        EXCH    $3 $1
                        ADDI    $1 -1
l_setValue_4:           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 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
```

```
                            ADDI   $10 -2
                            SUB    $10 $3
                            XORI   $9 1
assert_true_27:             BRA    assert_29
test_false_28:              BRA    test_26
assert_29:                  BNE    $9 $0 assert_true_27
                            ADD    $10 $3
                            ADDI   $10 2
                            EXCH   $11 $10
                            ADDI   $10 -2
                            SUB    $10 $3
cmp_top_37:                 BEQ    $11 $0 cmp_bot_38
                            XORI   $12 1
cmp_bot_38:                 BEQ    $11 $0 cmp_top_37
                            EXCH   $13 $6
cmp_top_39:                 BNE    $13 $0 cmp_bot_40
                            XORI   $14 1
cmp_bot_40:                 BNE    $13 $0 cmp_top_39
                            ANDX   $15 $12 $14
f_top_41:                   BEQ    $15 $0 f_bot_42
                            XORI   $16 1
f_bot_42:                   BEQ    $15 $0 f_top_41
                            XOR    $9 $16
f_bot_42_i:                 BEQ    $15 $0 f_top_41_i
                            XORI   $16 1
f_top_41_i:                 BEQ    $15 $0 f_bot_42_i
                            ANDX   $15 $12 $14
cmp_bot_40_i:               BNE    $13 $0 cmp_top_39_i
                            XORI   $14 1
cmp_top_39_i:               BNE    $13 $0 cmp_bot_40_i
                            EXCH   $13 $6
cmp_bot_38_i:               BEQ    $11 $0 cmp_top_37_i
                            XORI   $12 1
cmp_top_37_i:               BEQ    $11 $0 cmp_bot_38_i
                            ADD    $10 $3
                            ADDI   $10 2
                            EXCH   $11 $10
                            ADDI   $10 -2
                            SUB    $10 $3
                            ADD    $10 $3
                            ADDI   $10 2
                            EXCH   $11 $10
                            ADDI   $10 -2
                            SUB    $10 $3
cmp_top_47:                 BEQ    $11 $0 cmp_bot_48
                            XORI   $12 1
cmp_bot_48:                 BEQ    $11 $0 cmp_top_47
f_top_49:                   BEQ    $12 $0 f_bot_50
                            XORI   $13 1
f_bot_50:                   BEQ    $12 $0 f_top_49
                            XOR    $9 $13
f_bot_50_i:                 BEQ    $12 $0 f_top_49_i
                            XORI   $13 1
f_top_49_i:                 BEQ    $12 $0 f_bot_50_i
cmp_bot_48_i:               BEQ    $11 $0 cmp_top_47_i
                            XORI   $12 1
cmp_top_47_i:               BEQ    $11 $0 cmp_bot_48_i
                            ADD    $10 $3
                            ADDI   $10 2
                            EXCH   $11 $10
                            ADDI   $10 -2
                            SUB    $10 $3
test_43:                    BEQ    $9 $0 test_false_45
                            XORI   $9 1
                            ADD    $10 $3
                            ADDI   $10 2
                            EXCH   $11 $10
                            ADDI   $10 -2
                            SUB    $10 $3
                            XOR    $12 $11
loadMetAdd_51:              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 2
                            EXCH   $11 $10
                            ADDI   $10 -2
                            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 -435
l_jmp_52:                   SWAPBR $15
                            NEG    $15
                            ADDI   $15 435
                            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 2
                            EXCH   $11 $10
                            ADDI   $10 -2
                            SUB    $10 $3
                            EXCH   $13 $12
                            ADDI   $13 1
                            EXCH   $14 $13
```

```
                            XOR    $15 $14
                            EXCH   $14 $13
                            ADDI   $13 -1
loadMetAdd_51_i:            EXCH   $13 $12
                            XOR    $12 $11
                            ADD    $10 $3
                            ADDI   $10 2
                            EXCH   $11 $10
                            ADDI   $10 -2
                            SUB    $10 $3
                            XORI   $9 1
assert_true_44:             BRA    assert_46
test_false_45:              BRA    test_43
assert_46:                  BNE    $9 $0
    assert_true_44
                            ADD    $10 $3
                            ADDI   $10 2
                            EXCH   $11 $10
                            ADDI   $10 -2
                            SUB    $10 $3
cmp_top_53:                 BEQ    $11 $0
    cmp_bot_54
                            XORI   $12 1
cmp_bot_54:                 BEQ    $11 $0
    cmp_top_53
f_top_55:                   BEQ    $12 $0
    f_bot_56
                            XORI   $13 1
f_bot_56:                   BEQ    $12 $0
    f_top_55
                            XOR    $9 $13
f_bot_56_i:                 BEQ    $12 $0
    f_top_55_i
                            XORI   $13 1
f_top_55_i:                 BEQ    $12 $0
    f_bot_56_i
cmp_bot_54_i:               BEQ    $11 $0
    cmp_top_53_i
                            XORI   $12 1
cmp_top_53_i:               BEQ    $11 $0
    cmp_bot_54_i
                            ADD    $10 $3
                            ADDI   $10 2
                            EXCH   $11 $10
                            ADDI   $10 -2
                            SUB    $10 $3
                            XORI   $8 1
assert_true_19:             BRA    assert_21
test_false_20:              BRA    test_18
                            ADD    $10 $3
                            ADDI   $10 3
                            EXCH   $11 $10
                            ADDI   $10 -3
                            SUB    $10 $3
cmp_top_61:                 BNE    $11 $0
    cmp_bot_62
                            XORI   $12 1
cmp_bot_62:                 BNE    $11 $0
    cmp_top_61
                            EXCH   $13 $6
cmp_top_63:                 BEQ    $13 $0
    cmp_bot_64
                            XORI   $14 1
cmp_bot_64:                 BEQ    $13 $0
    cmp_top_63
                            ANDX   $15 $12 $14
f_top_65:                   BEQ    $15 $0
    f_bot_66
                            XORI   $16 1
f_bot_66:                   BEQ    $15 $0
    f_top_65
                            XOR    $9 $16
f_bot_66_i:                 BEQ    $15 $0
    f_top_65_i
                            XORI   $16 1
f_top_65_i:                 BEQ    $15 $0
    f_bot_66_i
                            ANDX   $15 $12 $14
cmp_bot_64_i:               BEQ    $13 $0
    cmp_top_63_i
                            XORI   $14 1
cmp_top_63_i:               BEQ    $13 $0
    cmp_bot_64_i
                            EXCH   $13 $6
cmp_bot_62_i:               BNE    $11 $0
    cmp_top_61_i
                            XORI   $12 1
cmp_top_61_i:               BNE    $11 $0
    cmp_bot_62_i
                            ADD    $10 $3
                            ADDI   $10 3
                            EXCH   $11 $10
                            ADDI   $10 -3
                            SUB    $10 $3
test_57:                    BEQ    $9 $0
    test_false_59
                            XORI   $9 1
                            ADD    $10 $3
                            ADDI   $10 3
                            EXCH   $11 $10
                            ADDI   $10 -3
                            SUB    $10 $3
                            EXCH   $12 $6
swap_67:                    XOR    $11 $12
                            XOR    $12 $11
                            XOR    $11 $12
                            EXCH   $12 $6
                            ADD    $10 $3
                            ADDI   $10 3
                            EXCH   $11 $10
                            ADDI   $10 -3
                            SUB    $10 $3
                            XORI   $9 1
assert_true_58:             BRA    assert_60
test_false_59:              BRA    test_57
assert_60:                  BNE    $9 $0
    assert_true_58
                            ADD    $10 $3
                            ADDI   $10 3
                            EXCH   $11 $10
                            ADDI   $10 -3
                            SUB    $10 $3
cmp_top_68:                 BEQ    $11 $0
    cmp_bot_69
                            XORI   $12 1
cmp_bot_69:                 BEQ    $11 $0
    cmp_top_68
                            EXCH   $13 $6
cmp_top_70:                 BNE    $13 $0
    cmp_bot_71
                            XORI   $14 1
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
                        XOR     $12 $11
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
```

```
                        XORI   $12 1
cmp_bot_85:             BEQ    $11 $0 cmp_top_84
f_top_86:               BEQ    $12 $0 f_bot_87
                        XORI   $13 1
f_bot_87:               BEQ    $12 $0 f_top_86
                        XOR    $9 $13
f_bot_87_i:             BEQ    $12 $0 f_top_86_i
                        XORI   $13 1
f_top_86_i:             BEQ    $12 $0 f_bot_87_i
cmp_bot_85_i:           BEQ    $11 $0 cmp_top_84_i
                        XORI   $12 1
cmp_top_84_i:           BEQ    $11 $0 cmp_bot_85_i
                        ADD    $10 $3
                        ADDI   $10 3
                        EXCH   $11 $10
                        ADDI   $10 -3
                        SUB    $10 $3
assert_21:              BNE    $8 $0 assert_true_19
                        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_88:             BGEZ   $12 cmp_bot_89
                        XORI   $13 1
cmp_bot_89:             BGEZ   $12 cmp_top_88
f_top_90:               BEQ    $13 $0 f_bot_91
                        XORI   $14 1
f_bot_91:               BEQ    $13 $0 f_top_90
                        XOR    $8 $14
f_bot_91_i:             BEQ    $13 $0 f_top_90_i
                        XORI   $14 1
f_top_90_i:             BEQ    $13 $0 f_bot_91_i
cmp_bot_89_i:           BGEZ   $12 cmp_top_88_i
                        XORI   $13 1
cmp_top_88_i:           BGEZ   $12 cmp_bot_89_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
l_insertNode_5_bot:     BRA    l_insertNode_5_top
l_getSum_6_top:         BRA    l_getSum_6_bot
                        ADDI   $1 1
                        EXCH   $2 $1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $3 $1
                        ADDI   $1 -1
l_getSum_6:             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 4
                        EXCH   $9 $8
                        ADDI   $8 -4
                        SUB    $8 $3
                        ADD    $7 $9
                        ADD    $8 $3
                        ADDI   $8 4
                        EXCH   $9 $8
                        ADDI   $8 -4
                        SUB    $8 $3
                        EXCH   $7 $6
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
cmp_top_96:             BEQ    $9 $0 cmp_bot_97
                        XORI   $10 1
cmp_bot_97:             BEQ    $9 $0 cmp_top_96
f_top_98:               BEQ    $10 $0 f_bot_99
                        XORI   $11 1
f_bot_99:               BEQ    $10 $0 f_top_98
                        XOR    $7 $11
f_bot_99_i:             BEQ    $10 $0 f_top_98_i
                        XORI   $11 1
f_top_98_i:             BEQ    $10 $0 f_bot_99_i
cmp_bot_97_i:           BEQ    $9 $0 cmp_top_96_i
                        XORI   $10 1
cmp_top_96_i:           BEQ    $9 $0 cmp_bot_97_i
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
test_92:                BEQ    $7 $0 test_false_94
                        XORI   $7 1
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
                        XOR    $10 $9
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
```

```
                                ADDI   $7 -3
                                SUB    $7 $3
                                XOR    $9 $8
loadMetAdd_143:                 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 3
                                EXCH   $8 $7
                                ADDI   $7 -3
                                SUB    $7 $3
                                EXCH   $3 $1
                                ADDI   $1 -1
                                EXCH   $9 $1
                                ADDI   $1 -1
                                ADDI   $12 -1142
l_jmp_144:                      SWAPBR $12
                                NEG    $12
                                ADDI   $12 1142
                                ADDI   $1 1
                                EXCH   $9 $1
                                ADDI   $1 1
                                EXCH   $3 $1
                                ADD    $7 $3
                                ADDI   $7 3
                                EXCH   $8 $7
                                ADDI   $7 -3
                                SUB    $7 $3
                                EXCH   $10 $9
                                ADDI   $10 3
                                EXCH   $11 $10
                                XOR    $12 $11
                                EXCH   $11 $10
                                ADDI   $10 -3
loadMetAdd_143_i:               EXCH   $10 $9
                                XOR    $9 $8
                                ADD    $7 $3
                                ADDI   $7 3
                                EXCH   $8 $7
                                ADDI   $7 -3
                                SUB    $7 $3
assert_138:                     BNE    $6 $0
    assert_true_136
                                ADD    $7 $3
                                ADDI   $7 3
                                EXCH   $8 $7
                                ADDI   $7 -3
                                SUB    $7 $3
cmp_top_145:                    BNE    $8 $0
    cmp_bot_146
                                XORI   $9 1
cmp_bot_146:                    BNE    $8 $0
    cmp_top_145
f_top_147:                      BEQ    $9 $0
    f_bot_148
                                XORI   $10 1
f_bot_148:                      BEQ    $9 $0
    f_top_147
                                XOR    $6 $10
f_bot_148_i:                    BEQ    $9 $0
    f_top_147_i
                                XORI   $10 1
f_top_147_i:                    BEQ    $9 $0
    f_bot_148_i
cmp_bot_146_i:                  BNE    $8 $0
    cmp_top_145_i
                                XORI   $9 1
cmp_top_145_i:                  BNE    $8 $0
    cmp_bot_146_i
                                ADD    $7 $3
                                ADDI   $7 3
                                EXCH   $8 $7
                                ADDI   $7 -3
                                SUB    $7 $3
l_mirror_7_bot:                 BRA    l_mirror_7_top
l_insertNode_1_top:             BRA
    l_insertNode_1_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_1:                 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
                                ADD    $9 $3
                                ADDI   $9 2
                                EXCH   $10 $9
                                ADDI   $9 -2
                                SUB    $9 $3
cmp_top_153:                    BNE    $10 $0
    cmp_bot_154
                                XORI   $11 1
cmp_bot_154:                    BNE    $10 $0
    cmp_top_153
                                EXCH   $12 $6
cmp_top_155:                    BEQ    $12 $0
    cmp_bot_156
                                XORI   $13 1
cmp_bot_156:                    BEQ    $12 $0
    cmp_top_155
                                ANDX   $14 $11 $13
f_top_157:                      BEQ    $14 $0
    f_bot_158
                                XORI   $15 1
f_bot_158:                      BEQ    $14 $0
    f_top_157
                                XOR    $8 $15
f_bot_158_i:                    BEQ    $14 $0
    f_top_157_i
                                XORI   $15 1
f_top_157_i:                    BEQ    $14 $0
    f_bot_158_i
                                ANDX   $14 $11 $13
cmp_bot_156_i:                  BEQ    $12 $0
    cmp_top_155_i
                                XORI   $13 1
cmp_top_155_i:                  BEQ    $12 $0
```

```
      cmp_bot_156_i
                          EXCH   $12 $6
cmp_bot_154_i:            BNE    $10 $0
      cmp_top_153_i
                          XORI   $11 1
cmp_top_153_i:            BNE    $10 $0
      cmp_bot_154_i
                          ADD    $9 $3
                          ADDI   $9 2
                          EXCH   $10 $9
                          ADDI   $9 -2
                          SUB    $9 $3
test_149:                 BEQ    $8 $0
      test_false_151
                          XORI   $8 1
                          ADD    $9 $3
                          ADDI   $9 2
                          EXCH   $10 $9
                          ADDI   $9 -2
                          SUB    $9 $3
                          EXCH   $11 $6
swap_159:                 XOR    $10 $11
                          XOR    $11 $10
                          XOR    $10 $11
                          EXCH   $11 $6
                          ADD    $9 $3
                          ADDI   $9 2
                          EXCH   $10 $9
                          ADDI   $9 -2
                          SUB    $9 $3
                          XORI   $8 1
assert_true_150:          BRA    assert_152
test_false_151:           BRA    test_149
assert_152:               BNE    $8 $0
      assert_true_150
                          ADD    $9 $3
                          ADDI   $9 2
                          EXCH   $10 $9
                          ADDI   $9 -2
                          SUB    $9 $3
cmp_top_160:              BEQ    $10 $0
      cmp_bot_161
                          XORI   $11 1
cmp_bot_161:              BEQ    $10 $0
      cmp_top_160
                          EXCH   $12 $6
cmp_top_162:              BNE    $12 $0
      cmp_bot_163
                          XORI   $13 1
cmp_bot_163:              BNE    $12 $0
      cmp_top_162
                          ANDX   $14 $11 $13
f_top_164:                BEQ    $14 $0
      f_bot_165
                          XORI   $15 1
f_bot_165:                BEQ    $14 $0
      f_top_164
                          XOR    $8 $15
f_bot_165_i:              BEQ    $14 $0
      f_top_164_i
                          XORI   $15 1
f_top_164_i:              BEQ    $14 $0
      f_bot_165_i
                          ANDX   $14 $11 $13
cmp_bot_163_i:            BNE    $12 $0
      cmp_top_162_i
                          XORI   $13 1
cmp_top_162_i:            BNE    $12 $0
      cmp_bot_163_i
                          EXCH   $12 $6
cmp_bot_161_i:            BEQ    $10 $0
      cmp_top_160_i
                          XORI   $11 1
cmp_top_160_i:            BEQ    $10 $0
      cmp_bot_161_i
                          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
cmp_top_170:              BEQ    $10 $0
      cmp_bot_171
                          XORI   $11 1
cmp_bot_171:              BEQ    $10 $0
      cmp_top_170
f_top_172:                BEQ    $11 $0
      f_bot_173
                          XORI   $12 1
f_bot_173:                BEQ    $11 $0
      f_top_172
                          XOR    $8 $12
f_bot_173_i:              BEQ    $11 $0
      f_top_172_i
                          XORI   $12 1
f_top_172_i:              BEQ    $11 $0
      f_bot_173_i
cmp_bot_171_i:            BEQ    $10 $0
      cmp_top_170_i
                          XORI   $11 1
cmp_top_170_i:            BEQ    $10 $0
      cmp_bot_171_i
                          ADD    $9 $3
                          ADDI   $9 2
                          EXCH   $10 $9
                          ADDI   $9 -2
                          SUB    $9 $3
test_166:                 BEQ    $8 $0
      test_false_168
                          XORI   $8 1
                          ADD    $9 $3
                          ADDI   $9 2
                          EXCH   $10 $9
                          ADDI   $9 -2
                          SUB    $9 $3
                          XOR    $11 $10
loadMetAdd_174:           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    $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
```

```
                        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 -1472
l_jmp_189:              SWAPBR $13
                        NEG    $13
                        ADDI   $13 1472
                        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_188_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_181:        BRA    assert_183
test_false_182:         BRA    test_180
assert_183:             BNE    $7 $0
    assert_true_181
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
cmp_top_190:            BEQ    $9 $0
    cmp_bot_191
                        XORI   $10 1
cmp_bot_191:            BEQ    $9 $0
    cmp_top_190
f_top_192:              BEQ    $10 $0
    f_bot_193
                        XORI   $11 1
f_bot_193:              BEQ    $10 $0
    f_top_192
                        XOR    $7 $11
f_bot_193_i:            BEQ    $10 $0
    f_top_192_i
                        XORI   $11 1
f_top_192_i:            BEQ    $10 $0
    f_bot_193_i
cmp_bot_191_i:          BEQ    $9 $0
    cmp_top_190_i
                        XORI   $10 1
cmp_top_190_i:          BEQ    $9 $0
    cmp_bot_191_i
                        ADD    $8 $3
                        ADDI   $8 2
                        EXCH   $9 $8
                        ADDI   $8 -2
                        SUB    $8 $3
l_sum_2_bot:            BRA    l_sum_2_top
l_mirror_3_top:         BRA    l_mirror_3_bot
                        ADDI   $1 1
                        EXCH   $2 $1
                        EXCH   $3 $1
                        ADDI   $1 -1
l_mirror_3:             SWAPBR $2
                        NEG    $2
                        ADDI   $1 1
                        EXCH   $3 $1
                        EXCH   $2 $1
                        ADDI   $1 -1
                        ADD    $7 $3
                        ADDI   $7 2
                        EXCH   $8 $7
                        ADDI   $7 -2
                        SUB    $7 $3
cmp_top_198:            BEQ    $8 $0
    cmp_bot_199
                        XORI   $9 1
cmp_bot_199:            BEQ    $8 $0
    cmp_top_198
f_top_200:              BEQ    $9 $0
    f_bot_201
                        XORI   $10 1
f_bot_201:              BEQ    $9 $0
    f_top_200
                        XOR    $6 $10
f_bot_201_i:            BEQ    $9 $0
    f_top_200_i
                        XORI   $10 1
f_top_200_i:            BEQ    $9 $0
    f_bot_201_i
cmp_bot_199_i:          BEQ    $8 $0
    cmp_top_198_i
                        XORI   $9 1
cmp_top_198_i:          BEQ    $8 $0
    cmp_bot_199_i
                        ADD    $7 $3
                        ADDI   $7 2
                        EXCH   $8 $7
                        ADDI   $7 -2
                        SUB    $7 $3
test_194:               BEQ    $6 $0
    test_false_196
                        XORI   $6 1
                        ADD    $7 $3
                        ADDI   $7 2
                        EXCH   $8 $7
                        ADDI   $7 -2
                        SUB    $7 $3
```

```
                     XOR    $9 $8
loadMetAdd_202:      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 -1586
l_jmp_203:           SWAPBR $12
                     NEG    $12
                     ADDI   $12 1586
                     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_202_i:    EXCH   $10 $9
                     XOR    $9 $8
                     ADD    $7 $3
                     ADDI   $7 2
                     EXCH   $8 $7
                     ADDI   $7 -2
                     SUB    $7 $3
                     XORI   $6 1
assert_true_195:     BRA    assert_197
test_false_196:      BRA    test_194
assert_197:          BNE    $6 $0 assert_true_195
                     ADD    $7 $3
                     ADDI   $7 2
                     EXCH   $8 $7
                     ADDI   $7 -2
                     SUB    $7 $3
cmp_top_204:         BEQ    $8 $0 cmp_bot_205
                     XORI   $9 1
cmp_bot_205:         BEQ    $8 $0 cmp_top_204
f_top_206:           BEQ    $9 $0 f_bot_207
                     XORI   $10 1
f_bot_207:           BEQ    $9 $0 f_top_206
                     XOR    $6 $10
f_bot_207_i:         BEQ    $9 $0 f_top_206_i
                     XORI   $10 1
f_top_206_i:         BEQ    $9 $0 f_bot_207_i
cmp_bot_205_i:       BEQ    $8 $0 cmp_top_204_i
                     XORI   $9 1
cmp_top_204_i:       BEQ    $8 $0 cmp_bot_205_i
                     ADD    $7 $3
                     ADDI   $7 2
                     EXCH   $8 $7
                     ADDI   $7 -2
                     SUB    $7 $3
l_mirror_3_bot:      BRA    l_mirror_3_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_208:         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_208_i:       ADDI   $8 -4
                     ADDI   $1 1
                     EXCH   $3 $1
                     ADD    $6 $3
                     ADDI   $6 3
                     XORI   $8 5
                     EXCH   $8 $7
                     ADDI   $7 1
                     XORI   $8 1
                     EXCH   $8 $7
obj_con_208_bot:     ADDI   $7 -1
                     EXCH   $7 $6
                     ADDI   $6 -3
                     SUB    $6 $3
                     ADD    $6 $3
                     ADDI   $6 4
                     EXCH   $7 $6
                     ADDI   $6 -4
                     SUB    $6 $3
                     XORI   $8 3
                     ADD    $7 $8
                     XORI   $8 3
                     ADD    $6 $3
                     ADDI   $6 4
                     EXCH   $7 $6
                     ADDI   $6 -4
                     SUB    $6 $3
localBlock_227:      XOR    $6 $1
                     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_212
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
```

```
    int listLength

    method main()
        new DoublyLinkedList list
        listLength += 10

        local int x = 0
        from x = 0 do skip
        loop
            local Cell cell = nil
                new Cell cell

                local Cell cellCopy = nil
                copy Cell cell cellCopy
                call cell::setSelf(cellCopy)
                delocal Cell cellCopy = nil

                call cell::setData(x)
                call list::appendCell(cell)
            delocal Cell cell = nil
            x += 1
        until x = listLength
        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
```

```
                          ADDI   $7 -2
                          SUB    $7 $3
                          EXCH   $9 $6
                          XOR    $8 $9
                          EXCH   $9 $6
                          ADD    $7 $3
                          ADDI   $7 2
                          EXCH   $8 $7
                          ADDI   $7 -2
                          SUB    $7 $3
l_setData_2_bot:          BRA    l_setData_2_top
l_setIndex_3_top:         BRA    l_setIndex_3_bot
                          ADDI   $1 1
                          EXCH   $2 $1
                          EXCH   $6 $1
                          ADDI   $1 -1
                          EXCH   $3 $1
                          ADDI   $1 -1
l_setIndex_3:             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_setIndex_3_bot:         BRA    l_setIndex_3_top
l_setLeft_4_top:          BRA    l_setLeft_4_bot
                          ADDI   $1 1
                          EXCH   $2 $1
                          EXCH   $6 $1
                          ADDI   $1 -1
                          EXCH   $3 $1
                          ADDI   $1 -1
l_setLeft_4:              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 4
                          EXCH   $8 $7
                          ADDI   $7 -4
                          SUB    $7 $3
                          EXCH   $9 $6
swap_18:                  XOR    $8 $9
                          XOR    $9 $8
                          XOR    $8 $9
                          EXCH   $9 $6
                          ADD    $7 $3
                          ADDI   $7 4
                          EXCH   $8 $7
                          ADDI   $7 -4
                          SUB    $7 $3
l_setLeft_4_bot:          BRA    l_setLeft_4_top
l_setRight_5_top:         BRA    l_setRight_5_bot
                          ADDI   $1 1
                          EXCH   $2 $1
                          EXCH   $6 $1
                          ADDI   $1 -1
                          EXCH   $3 $1
                          ADDI   $1 -1
l_setRight_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 5
                          EXCH   $8 $7
                          ADDI   $7 -5
                          SUB    $7 $3
                          EXCH   $9 $6
swap_19:                  XOR    $8 $9
                          XOR    $9 $8
                          XOR    $8 $9
                          EXCH   $9 $6
                          ADD    $7 $3
                          ADDI   $7 5
                          EXCH   $8 $7
                          ADDI   $7 -5
                          SUB    $7 $3
l_setRight_5_bot:         BRA    l_setRight_5_top
l_setSelf_6_top:          BRA    l_setSelf_6_bot
                          ADDI   $1 1
                          EXCH   $2 $1
                          EXCH   $6 $1
                          ADDI   $1 -1
                          EXCH   $3 $1
                          ADDI   $1 -1
l_setSelf_6:              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 6
                          EXCH   $8 $7
                          ADDI   $7 -6
                          SUB    $7 $3
                          EXCH   $9 $6
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
```

```
                      ADDI   $9 -3
                      SUB    $9 $3
                      XORI   $7 1
assert_true_22:       BRA    assert_24
test_false_23:        BRA    test_21
assert_24:            BNE    $7 $0     assert_true_22
                      ADD    $8 $3
                      ADDI   $8 5
                      EXCH   $9 $8
                      ADDI   $8 -5
                      SUB    $8 $3
cmp_top_39:           BEQ    $9 $0     cmp_bot_40
                      XORI   $10 1
cmp_bot_40:           BEQ    $9 $0     cmp_top_39
                      EXCH   $11 $6
cmp_top_41:           BNE    $11 $0    cmp_bot_42
                      XORI   $12 1
cmp_bot_42:           BNE    $11 $0    cmp_top_41
                      ANDX   $13 $10 $12
f_top_43:             BEQ    $13 $0    f_bot_44
                      XORI   $14 1
f_bot_44:             BEQ    $13 $0    f_top_43
                      XOR    $7 $14
f_bot_44_i:           BEQ    $13 $0    f_top_43_i
                      XORI   $14 1
f_top_43_i:           BEQ    $13 $0    f_bot_44_i
                      ANDX   $13 $10 $12
cmp_bot_42_i:         BNE    $11 $0    cmp_top_41_i
                      XORI   $12 1
cmp_top_41_i:         BNE    $11 $0    cmp_bot_42_i
                      EXCH   $11 $6
cmp_bot_40_i:         BEQ    $9 $0     cmp_top_39_i
                      XORI   $10 1
cmp_top_39_i:         BEQ    $9 $0     cmp_bot_40_i
                      ADD    $8 $3
                      ADDI   $8 5
                      EXCH   $9 $8
                      ADDI   $8 -5
                      SUB    $8 $3
                      ADD    $8 $3
                      ADDI   $8 5
                      EXCH   $9 $8
                      ADDI   $8 -5
                      SUB    $8 $3
cmp_top_49:           BEQ    $9 $0     cmp_bot_50
                      XORI   $10 1
cmp_bot_50:           BEQ    $9 $0     cmp_top_49
f_top_51:             BEQ    $10 $0    f_bot_52
                      XORI   $11 1
f_bot_52:             BEQ    $10 $0    f_top_51
                      XOR    $7 $11
f_bot_52_i:           BEQ    $10 $0    f_top_51_i
                      XORI   $11 1
f_top_51_i:           BEQ    $10 $0    f_bot_52_i
cmp_bot_50_i:         BEQ    $9 $0     cmp_top_49_i
                      XORI   $10 1
cmp_top_49_i:         BEQ    $9 $0     cmp_bot_50_i
                      ADD    $8 $3
                      ADDI   $8 5
                      EXCH   $9 $8
                      ADDI   $8 -5
                      SUB    $8 $3
test_45:              BEQ    $7 $0     test_false_47
                      XORI   $7 1
                      ADD    $8 $3
                      ADDI   $8 5
                      EXCH   $9 $8
                      ADDI   $8 -5
                      SUB    $8 $3
                      XOR    $10 $9
loadMetAdd_53:        EXCH   $11 $10
                      ADDI   $11 5
                      EXCH   $12 $11
                      XOR    $13 $12
                      EXCH   $12 $11
                      ADDI   $11 -5
                      EXCH   $11 $10
                      ADD    $8 $3
                      ADDI   $8 5
                      EXCH   $9 $8
                      ADDI   $8 -5
                      SUB    $8 $3
                      EXCH   $3 $1
                      ADDI   $1 -1
                      EXCH   $6 $1
                      ADDI   $1 -1
                      EXCH   $10 $1
                      ADDI   $1 -1
                      ADDI   $13 -697
l_jmp_54:             SWAPBR $13
                      NEG    $13
                      ADDI   $13 697
                      ADDI   $1 1
                      EXCH   $10 $1
                      ADDI   $1 1
                      EXCH   $6 $1
                      ADDI   $1 1
                      EXCH   $3 $1
                      ADD    $8 $3
                      ADDI   $8 5
                      EXCH   $9 $8
                      ADDI   $8 -5
                      SUB    $8 $3
                      EXCH   $11 $10
                      ADDI   $11 5
                      EXCH   $12 $11
                      XOR    $13 $12
                      EXCH   $12 $11
                      ADDI   $11 -5
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
```

```
cmp_top_70:         BEQ    $9 $0 cmp_bot_71
                    XORI   $10 1
cmp_bot_71:         BEQ    $9 $0 cmp_top_70
                    EXCH   $11 $6
cmp_top_72:         BNE    $11 $0 cmp_bot_73
                    XORI   $12 1
cmp_bot_73:         BNE    $11 $0 cmp_top_72
                    ANDX   $13 $10 $12
f_top_74:           BEQ    $13 $0 f_bot_75
                    XORI   $14 1
f_bot_75:           BEQ    $13 $0 f_top_74
                    XOR    $7 $14
f_bot_75_i:         BEQ    $13 $0 f_top_74_i
                    XORI   $14 1
f_top_74_i:         BEQ    $13 $0 f_bot_75_i
                    ANDX   $13 $10 $12
cmp_bot_73_i:       BNE    $11 $0 cmp_top_72_i
                    XORI   $12 1
cmp_top_72_i:       BNE    $11 $0 cmp_bot_73_i
                    EXCH   $11 $6
cmp_bot_71_i:       BEQ    $9 $0 cmp_top_70_i
                    XORI   $10 1
cmp_top_70_i:       BEQ    $9 $0 cmp_bot_71_i
                    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
cmp_top_80:         BEQ    $9 $0 cmp_bot_81
                    XORI   $10 1
cmp_bot_81:         BEQ    $9 $0 cmp_top_80
f_top_82:           BEQ    $10 $0 f_bot_83
                    XORI   $11 1
f_bot_83:           BEQ    $10 $0 f_top_82
                    XOR    $7 $11
f_bot_83_i:         BEQ    $10 $0 f_top_82_i
                    XORI   $11 1
f_top_82_i:         BEQ    $10 $0 f_bot_83_i
cmp_bot_81_i:       BEQ    $9 $0 cmp_top_80_i
                    XORI   $10 1
cmp_top_80_i:       BEQ    $9 $0 cmp_bot_81_i
                    ADD    $8 $3
                    ADDI   $8 2
                    EXCH   $9 $8
                    ADDI   $8 -2
                    SUB    $8 $3
test_76:            BEQ    $7 $0 test_false_78
                    XORI   $7 1
                    ADD    $8 $3
                    ADDI   $8 2
                    EXCH   $9 $8
                    ADDI   $8 -2
                    SUB    $8 $3
                    XOR    $10 $9
loadMetAdd_84:      EXCH   $11 $10
                    ADDI   $11 5
                    EXCH   $12 $11
                    XOR    $13 $12
                    EXCH   $12 $11
                    ADDI   $11 -5
                    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 -904
l_jmp_85:           SWAPBR $13
                    NEG    $13
                    ADDI   $13 904
                    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 5
                    EXCH   $12 $11
                    XOR    $13 $12
                    EXCH   $12 $11
                    ADDI   $11 -5
loadMetAdd_84_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_77:     BRA    assert_79
test_false_78:      BRA    test_76
assert_79:          BNE    $7 $0 assert_true_77
                    ADD    $8 $3
```

```
                            ADDI   $8 2
                            EXCH   $9 $8
                            ADDI   $8 -2
                            SUB    $8 $3
cmp_top_86:                 BEQ    $9 $0 cmp_bot_87
                            XORI   $10 1
cmp_bot_87:                 BEQ    $9 $0 cmp_top_86
f_top_88:                   BEQ    $10 $0 f_bot_89
                            XORI   $11 1
f_bot_89:                   BEQ    $10 $0 f_top_88
                            XOR    $7 $11
f_bot_89_i:                 BEQ    $10 $0 f_top_88_i
                            XORI   $11 1
f_top_88_i:                 BEQ    $10 $0 f_bot_89_i
cmp_bot_87_i:               BEQ    $9 $0 cmp_top_86_i
                            XORI   $10 1
cmp_top_86_i:               BEQ    $9 $0 cmp_bot_87_i
                            ADD    $8 $3
                            ADDI   $8 2
                            EXCH   $9 $8
                            ADDI   $8 -2
                            SUB    $8 $3
                            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_appendCell_1_bot:         BRA    l_appendCell_1_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_90:                 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_90_i:               ADDI   $8 -4
                            ADDI   $1 1
                            EXCH   $3 $1
                            ADD    $6 $3
                            ADDI   $6 2
                            XORI   $8 4
                            EXCH   $8 $7
                            ADDI   $7 1
                            XORI   $8 1
                            EXCH   $8 $7
obj_con_90_bot:             ADDI   $7 -1
                            EXCH   $7 $6
                            ADDI   $6 -2
                            SUB    $6 $3
                            ADD    $6 $3
                            ADDI   $6 3
                            EXCH   $7 $6
                            ADDI   $6 -3
                            SUB    $6 $3
                            XORI   $8 10
                            ADD    $7 $8
                            XORI   $8 10
                            ADD    $6 $3
                            ADDI   $6 3
                            EXCH   $7 $6
                            ADDI   $6 -3
                            SUB    $6 $3
localBlock_113:             XOR    $6 $1
                            XOR    $7 $0
                            EXCH   $7 $1
                            ADDI   $1 -1
                            XORI   $7 1
entry_91:                   BEQ    $7 $0 assert_93
                            EXCH   $8 $6
cmp_top_95:                 BNE    $8 $0 cmp_bot_96
                            XORI   $9 1
cmp_bot_96:                 BNE    $8 $0 cmp_top_95
f_top_97:                   BEQ    $9 $0 f_bot_98
                            XORI   $10 1
f_bot_98:                   BEQ    $9 $0 f_top_97
                            XOR    $7 $10
f_bot_98_i:                 BEQ    $9 $0 f_top_97_i
                            XORI   $10 1
f_top_97_i:                 BEQ    $9 $0 f_bot_98_i
cmp_bot_96_i:               BNE    $8 $0 cmp_top_95_i
                            XORI   $9 1
cmp_top_95_i:               BNE    $8 $0 cmp_bot_96_i
                            EXCH   $8 $6
                            EXCH   $8 $6
                            ADD    $9 $3
                            ADDI   $9 3
                            EXCH   $10 $9
                            ADDI   $9 -3
                            SUB    $9 $3
cmp_top_109:                BNE    $8 $10 cmp_bot_110
                            XORI   $11 1
```

```
cmp_bot_110:            BNE    $8 $10 cmp_top_109
f_top_111:              BEQ    $11 $0 f_bot_112
                        XORI   $12 1
f_bot_112:              BEQ    $11 $0 f_top_111
                        XOR    $7 $12
f_bot_112_i:            BEQ    $11 $0 f_top_111_i
                        XORI   $12 1
f_top_111_i:            BEQ    $11 $0 f_bot_112_i
cmp_bot_110_i:          BNE    $8 $10 cmp_top_109_i
                        XORI   $11 1
cmp_top_109_i:          BNE    $8 $10 cmp_bot_110_i
                        ADD    $9 $3
                        ADDI   $9 3
                        EXCH   $10 $9
                        ADDI   $9 -3
                        SUB    $9 $3
                        EXCH   $8 $6
test_92:                BNE    $7 $0 exit_94
localBlock_108:         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_99:             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_99_i:           ADDI   $10 -8
                        ADDI   $1 1
                        EXCH   $6 $1
                        ADDI   $1 1
                        EXCH   $8 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        XORI   $10 5
                        EXCH   $10 $9
                        ADDI   $9 1
                        XORI   $10 1
                        EXCH   $10 $9
obj_con_99_bot:         ADDI   $9 -1
                        EXCH   $9 $8
localBlock_103:         XOR    $9 $1
                        XOR    $10 $0
                        EXCH   $10 $1
                        ADDI   $1 -1
                        EXCH   $10 $9
                        EXCH   $11 $8
copy_100:               XOR    $10 $11
                        ADDI   $11 1
                        EXCH   $12 $11
                        ADDI   $12 1
                        EXCH   $12 $11
                        ADDI   $11 -1
                        EXCH   $11 $8
                        EXCH   $10 $9
                        EXCH   $10 $8
                        XOR    $11 $10
loadMetAdd_101:         EXCH   $12 $11
                        ADDI   $12 4
                        EXCH   $13 $12
                        XOR    $14 $13
                        EXCH   $13 $12
                        ADDI   $12 -4
                        EXCH   $12 $11
                        EXCH   $10 $8
                        EXCH   $3 $1
                        ADDI   $1 -1
                        EXCH   $8 $1
                        ADDI   $1 -1
                        EXCH   $6 $1
                        ADDI   $1 -1
                        EXCH   $9 $1
                        ADDI   $1 -1
                        EXCH   $11 $1
                        ADDI   $1 -1
                        ADDI   $14 -1138
l_jmp_102:              SWAPBR $14
                        NEG    $14
                        ADDI   $14 1138
                        ADDI   $1 1
                        EXCH   $11 $1
                        ADDI   $1 1
                        EXCH   $9 $1
                        ADDI   $1 1
                        EXCH   $6 $1
                        ADDI   $1 1
                        EXCH   $8 $1
                        ADDI   $1 1
                        EXCH   $3 $1
                        EXCH   $10 $8
                        EXCH   $12 $11
                        ADDI   $12 4
                        EXCH   $13 $12
                        XOR    $14 $13
                        EXCH   $13 $12
                        ADDI   $12 -4
loadMetAdd_101_i:       EXCH   $12 $11
                        XOR    $11 $10
                        EXCH   $10 $8
                        ADDI   $1 1
                        EXCH   $10 $1
                        XOR    $10 $0
localBlock_103_i:       XOR    $9 $1
                        EXCH   $9 $8
                        XOR    $10 $9
loadMetAdd_104:         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 -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
loadMetAdd_106_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_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
```

```
                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    $1 1
                EXCH    $6 $3
                XORI    $6 3
                XOR     $3 $1
                ADDI    $1 4
                ADDI    $1 -2048
                XOR     $1 $5
                ADDI    $5 -10
                XOR     $5 $4
                ADDI    $4 -1338
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

```

```
        q1[2] += 3
        s1[2] += 0
        s2[2] += 1
        q2[2] += 4

        q1[3] += 3
        s1[3] += 1
        s2[3] += 0
        q2[3] += 2

        q1[4] += 3
        s1[4] += BLANK
        s2[4] += BLANK
        q2[4] += 4

        q1[5] += 4
        s1[5] += SLASH
        s2[5] += LEFT
        q2[5] += 5

        q1[6] += 5
        s1[6] += 0
        s2[6] += 0
        q2[6] += 4

        q1[7] += 5
        s1[7] += BLANK
        s2[7] += BLANK
        q2[7] += 6

    method initTape()
        local Cell cell0 = nil
        local Cell cell1 = nil
        local Cell cell2 = nil
        local Cell cell3 = nil
        local Cell cell4 = nil

        // Init cells
        new Cell cell0
        new Cell cell1
        new Cell cell2
        new Cell cell3
        new Cell cell4

        // Write 1 1 0 1 on tape
        symbol += BLANK
        uncall cell0::getSymbol(symbol)
        symbol += 1
        uncall cell1::getSymbol(symbol)
        symbol += 1
        uncall cell2::getSymbol(symbol)
        symbol += 1
        uncall cell4::getSymbol(symbol)

        // Set tape head
        tapeHead <=> cell0

        // Set self pointers
        copy Cell tapeHead cell0
        uncall tapeHead::getSelf(cell0)
        copy Cell cell1 cell0
        uncall cell1::getSelf(cell0)
        copy Cell cell2 cell0
        uncall cell2::getSelf(cell0)
        copy Cell cell3 cell0
        uncall cell3::getSelf(cell0)
```

```
        copy Cell cell4 cell0
        uncall cell4::getSelf(cell0)

        // Link cell 3 and 4
        copy Cell cell3 cell0
        uncall cell4::getLeft(cell0)
        uncall cell3::getRight(cell4)

        // Link cell 2 and 3
        copy Cell cell2 cell0
        uncall cell3::getLeft(cell0)
        uncall cell2::getRight(cell3)

        // Link cell1 and cell 2
        copy Cell cell1 cell0
        uncall cell2::getLeft(cell0)
        uncall cell1::getRight(cell2)

        // Link tapeHead and cell 1
        copy Cell tapeHead cell0
        uncall cell1::getLeft(cell0)
        uncall tapeHead::getRight(cell1)

        delocal Cell cell4 = nil
        delocal Cell cell3 = nil
        delocal Cell cell2 = nil
        delocal Cell cell1 = nil
        delocal Cell cell0 = nil

    method init()
        // Prepare for simulation
        call initLiterals()
        call initRules()
        call initTape()

        // Init pc, start and finishing state
        state += 1
        Qs += 1
        Qf += 6

        // Start simulation
        call simulate()

    method simulate()
        from state = Qs do
            call tapeHead::getSymbol(symbol)    // Fetch current symbol
            call inst()
            uncall tapeHead::getSymbol(symbol)  // Zero-clear symbol
            pc += 1                             // Increment pc

            if pc = PC_MAX then                 // Reset pc
                pc ^= PC_MAX
            else skip
            fi pc = 0
        loop skip
        until state = Qf

    method inst()
        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]
        else skip
        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()                    // Move tape head right
            else skip
            fi s2[pc] = RIGHT
            if s2[pc] = LEFT then
                uncall moveRight()                  // Move tape head left
            else skip
            fi s2[pc] = LEFT
        else skip
        fi state = q2[pc] & s1[pc] = SLASH

    method moveRight()
        local Cell right = nil
        local Cell tmp = nil
        uncall tapeHead::getSymbol(symbol)   // Put symbol back in current cell
        call tapeHead::getRight(right)       // Get right neighbour

        if right = nil & symbol = BLANK then
            symbol ^= BLANK                  // Zero clear symbol
            new Cell right                   // Init new neighbour
            copy Cell right tmp              // Copy reference to self
            uncall right::getSelf(tmp)       // Store self reference
            uncall right::getLeft(tapeHead)  // Set tape head as left of new cell
            right <=> tapeHead
        else
            call right::getLeft(tmp)         // Get copy of tape head reference
            uncopy Cell tmp tapeHead         // Clear reference to tape head

            if tapeHead = nil & symbol = BLANK then
                call tmp::getSelf(tapeHead)  // rev: set self pointer
                uncopy Cell tmp tapeHead     // rev: new self pointer
                delete Cell tmp              // rev: new left neighbour
                symbol ^= BLANK
            else skip                        // In reverse:
            fi tmp = nil                     // Allocate new left if current is nil

            uncall right::getLeft(tmp)       // Put tape head reference back
            tapeHead <=> right
            call tapeHead::getRight(right)   // Get right of new tape head
            call tapeHead::getSymbol(symbol) // Get symbol of new tape head
        fi right = nil

        uncall tapeHead::getRight(right)     // Set right neighbour
        delocal Cell right = nil
        delocal Cell tmp = nil

class Program
    RTM bni

    method main()
        // This program contains a RTM implementing
        // incrementation of a non-negative n-bit binary number by 1 (modulo 2n).
        // The tape is initialized with | b | 1Â | 1 | 0 | 0 | and after execution,
        // the tape is left with | bÂ | 0 | 0Â | 1 | 1 |
        new RTM bni
        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
```

```
                         ADD    $7 $3
                         ADDI   $7 3
                         EXCH   $8 $7
                         ADDI   $7 -3
                         SUB    $7 $3
                         EXCH   $9 $6
swap_22:                 XOR    $8 $9
                         XOR    $9 $8
                         XOR    $8 $9
                         EXCH   $9 $6
                         ADD    $7 $3
                         ADDI   $7 3
                         EXCH   $8 $7
                         ADDI   $7 -3
                         SUB    $7 $3
l_getLeft_8_bot:         BRA    l_getLeft_8_top
l_getRight_9_top:        BRA    l_getRight_9_bot
                         ADDI   $1 1
                         EXCH   $2 $1
                         EXCH   $6 $1
                         ADDI   $1 -1
                         EXCH   $3 $1
                         ADDI   $1 -1
l_getRight_9:            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 4
                         EXCH   $8 $7
                         ADDI   $7 -4
                         SUB    $7 $3
                         EXCH   $9 $6
swap_23:                 XOR    $8 $9
                         XOR    $9 $8
                         XOR    $8 $9
                         EXCH   $9 $6
                         ADD    $7 $3
                         ADDI   $7 4
                         EXCH   $8 $7
                         ADDI   $7 -4
                         SUB    $7 $3
l_getRight_9_bot:        BRA    l_getRight_9_top
l_getSelf_10_top:        BRA    l_getSelf_10_bot
                         ADDI   $1 1
                         EXCH   $2 $1
                         EXCH   $6 $1
                         ADDI   $1 -1
                         EXCH   $3 $1
                         ADDI   $1 -1
l_getSelf_10:            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
                         ADDI   $7 -2
                         SUB    $7 $3
                         EXCH   $9 $6
swap_24:                 XOR    $8 $9
                         XOR    $9 $8
                         XOR    $8 $9
                         EXCH   $9 $6
                         ADD    $7 $3
                         ADDI   $7 2
                         EXCH   $8 $7
                         ADDI   $7 -2
                         SUB    $7 $3
l_getSelf_10_bot:        BRA    l_getSelf_10_top
l_getSymbol_11_top:      BRA    l_getSymbol_11_bot
                         ADDI   $1 1
                         EXCH   $2 $1
                         EXCH   $6 $1
                         ADDI   $1 -1
                         EXCH   $3 $1
                         ADDI   $1 -1
l_getSymbol_11:          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 5
                         EXCH   $9 $8
                         ADDI   $8 -5
                         SUB    $8 $3
swap_25:                 XOR    $7 $9
                         XOR    $9 $7
                         XOR    $7 $9
                         ADD    $8 $3
                         ADDI   $8 5
                         EXCH   $9 $8
                         ADDI   $8 -5
                         SUB    $8 $3
                         EXCH   $7 $6
l_getSymbol_11_bot:      BRA    l_getSymbol_11_top
l_initLiterals_1_top:    BRA    l_initLiterals_1_bot
                         ADDI   $1 1
                         EXCH   $2 $1
                         EXCH   $3 $1
                         ADDI   $1 -1
l_initLiterals_1:        SWAPBR $2
                         NEG    $2
                         ADDI   $1 1
                         EXCH   $3 $1
                         EXCH   $2 $1
                         ADDI   $1 -1
                         ADD    $6 $3
                         ADDI   $6 7
                         EXCH   $7 $6
                         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
```

```
                                        SUB     $7 $0
                                        ADDI    $7 -2
                                        EXCH    $8 $6
                                        XOR     $7 $8
                                        EXCH    $8 $6
                                        ADDI    $6 -5
                                        SUB     $6 $3
                                        ADD     $10 $3
                                        ADDI    $10 10
                                        EXCH    $11 $10
                                        ADDI    $10 -10
                                        SUB     $10 $3
assArrElem_31:                          ADD     $9 $11
                                        ADD     $10 $3
                                        ADDI    $10 10
                                        EXCH    $11 $10
                                        ADDI    $10 -10
                                        SUB     $10 $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
                                        SUB     $7 $0
                                        ADDI    $7 -2
                                        EXCH    $8 $6
                                        XOR     $7 $8
                                        EXCH    $8 $6
                                        ADDI    $6 -5
                                        SUB     $6 $3
                                        ADD     $6 $3
                                        ADDI    $6 6
                                        EXCH    $8 $6
                                        XOR     $7 $8
                                        EXCH    $8 $6
                                        ADDI    $7 2
                                        ADD     $7 $0
                                        ADDI    $6 -6
                                        SUB     $6 $3
                                        EXCH    $9 $7
                                        ADD     $6 $3
                                        ADDI    $6 6
                                        SUB     $7 $0
                                        ADDI    $7 -2
                                        EXCH    $8 $6
                                        XOR     $7 $8
                                        EXCH    $8 $6
                                        ADDI    $6 -6
                                        SUB     $6 $3
                                        ADD     $10 $3
                                        ADDI    $10 10
                                        EXCH    $11 $10
                                        ADDI    $10 -10
                                        SUB     $10 $3
assArrElem_32:                          ADD     $9 $11
                                        ADD     $10 $3
                                        ADDI    $10 10
                                        EXCH    $11 $10
                                        ADDI    $10 -10
                                        SUB     $10 $3
                                        ADD     $6 $3
                                        ADDI    $6 6
                                        EXCH    $8 $6
                                        XOR     $7 $8
                                        EXCH    $8 $6
                                        ADDI    $7 2
                                        ADD     $7 $0
                                        ADDI    $6 -6
                                        SUB     $6 $3
                                        EXCH    $9 $7
                                        ADD     $6 $3
                                        ADDI    $6 6
                                        SUB     $7 $0
                                        ADDI    $7 -2
                                        EXCH    $8 $6
                                        XOR     $7 $8
                                        EXCH    $8 $6
                                        ADDI    $6 -6
                                        SUB     $6 $3
                                        ADD     $6 $3
                                        ADDI    $6 4
                                        EXCH    $8 $6
                                        XOR     $7 $8
                                        EXCH    $8 $6
                                        ADDI    $7 2
                                        ADD     $7 $0
                                        ADDI    $6 -4
                                        SUB     $6 $3
                                        EXCH    $9 $7
                                        ADD     $6 $3
                                        ADDI    $6 4
                                        SUB     $7 $0
                                        ADDI    $7 -2
                                        EXCH    $8 $6
                                        XOR     $7 $8
                                        EXCH    $8 $6
                                        ADDI    $6 -4
                                        SUB     $6 $3
                                        XORI    $10 2
assArrElem_33:                          ADD     $9 $10
                                        XORI    $10 2
                                        ADD     $6 $3
                                        ADDI    $6 4
                                        EXCH    $8 $6
                                        XOR     $7 $8
                                        EXCH    $8 $6
                                        ADDI    $7 2
                                        ADD     $7 $0
                                        ADDI    $6 -4
                                        SUB     $6 $3
                                        EXCH    $9 $7
                                        ADD     $6 $3
                                        ADDI    $6 4
                                        SUB     $7 $0
                                        ADDI    $7 -2
                                        EXCH    $8 $6
                                        XOR     $7 $8
                                        EXCH    $8 $6
                                        ADDI    $6 -4
                                        SUB     $6 $3
                                        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
                        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
                        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 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
                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
```

```
                    XORI   $7 2
                    ADDI   $6 -5
                    SUB    $6 $3
                    EXCH   $10 $8
                    ADD    $6 $3
                    ADDI   $6 5
                    XORI   $7 2
                    SUB    $8 $7
                    ADDI   $8 -2
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    XORI   $7 2
                    ADDI   $6 -5
                    SUB    $6 $3
assArrElem_39:      ADD    $10 $0
                    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 -5
                    SUB    $6 $3
                    EXCH   $10 $8
                    ADD    $6 $3
                    ADDI   $6 5
                    XORI   $7 2
                    SUB    $8 $7
                    ADDI   $8 -2
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    XORI   $7 2
                    ADDI   $6 -5
                    SUB    $6 $3
                    ADD    $6 $3
                    ADDI   $6 6
                    XORI   $7 2
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    ADDI   $8 2
                    ADD    $8 $7
                    XORI   $7 2
                    ADDI   $6 -6
                    SUB    $6 $3
                    EXCH   $10 $8
                    ADD    $6 $3
                    ADDI   $6 6
                    XORI   $7 2
                    SUB    $8 $7
                    ADDI   $8 -2
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    XORI   $7 2
                    ADDI   $6 -6
                    SUB    $6 $3
                    XORI   $11 1
assArrElem_40:      ADD    $10 $11
                    XORI   $11 1
                    ADD    $6 $3
                    ADDI   $6 6
                    XORI   $7 2
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    ADDI   $8 2
                    ADD    $8 $7
                    XORI   $7 2
                    ADDI   $6 -6
                    SUB    $6 $3
                    EXCH   $10 $8
                    ADD    $6 $3
                    ADDI   $6 6
                    XORI   $7 2
                    SUB    $8 $7
                    ADDI   $8 -2
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    XORI   $7 2
                    ADDI   $6 -6
                    SUB    $6 $3
                    ADD    $6 $3
                    ADDI   $6 4
                    XORI   $7 2
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    ADDI   $8 2
                    ADD    $8 $7
                    XORI   $7 2
                    ADDI   $6 -4
                    SUB    $6 $3
                    EXCH   $10 $8
                    ADD    $6 $3
                    ADDI   $6 4
                    XORI   $7 2
                    SUB    $8 $7
                    ADDI   $8 -2
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    XORI   $7 2
                    ADDI   $6 -4
                    SUB    $6 $3
                    XORI   $11 4
assArrElem_41:      ADD    $10 $11
                    XORI   $11 4
                    ADD    $6 $3
                    ADDI   $6 4
                    XORI   $7 2
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    ADDI   $8 2
                    ADD    $8 $7
                    XORI   $7 2
                    ADDI   $6 -4
                    SUB    $6 $3
                    EXCH   $10 $8
                    ADD    $6 $3
                    ADDI   $6 4
                    XORI   $7 2
                    SUB    $8 $7
                    ADDI   $8 -2
                    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
                SUB   $8 $7
                ADDI  $8 -2
                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
```

```
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 4
                        ADDI    $6 -5
                        SUB     $6 $3
                        ADD     $11 $3
                        ADDI    $11 10
                        EXCH    $12 $11
                        ADDI    $11 -10
                        SUB     $11 $3
assArrElem_47:          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 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
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 4
                        ADDI    $6 -5
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 4
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 4
                        ADDI    $6 -6
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 4
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 4
                        ADDI    $6 -6
                        SUB     $6 $3
                        ADD     $11 $3
                        ADDI    $11 10
                        EXCH    $12 $11
                        ADDI    $11 -10
                        SUB     $11 $3
assArrElem_48:          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 4
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 4
                        ADDI    $6 -6
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 6
                        XORI    $7 4
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 4
                        ADDI    $6 -6
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 4
                        XORI    $7 4
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
                        XORI    $7 4
                        ADDI    $6 -4
                        SUB     $6 $3
                        EXCH    $10 $8
                        ADD     $6 $3
                        ADDI    $6 4
                        XORI    $7 4
                        SUB     $8 $7
                        ADDI    $8 -2
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        XORI    $7 4
                        ADDI    $6 -4
                        SUB     $6 $3
                        XORI    $11 4
assArrElem_49:          ADD     $10 $11
                        XORI    $11 4
                        ADD     $6 $3
                        ADDI    $6 4
                        XORI    $7 4
                        EXCH    $9 $6
                        XOR     $8 $9
                        EXCH    $9 $6
                        ADDI    $8 2
                        ADD     $8 $7
```

```
                XORI  $7 4
                ADDI  $6 -4
                SUB   $6 $3
                EXCH  $10 $8
                ADD   $6 $3
                ADDI  $6 4
                XORI  $7 4
                SUB   $8 $7
                ADDI  $8 -2
                EXCH  $9 $6
                XOR   $8 $9
                EXCH  $9 $6
                XORI  $7 4
                ADDI  $6 -4
                SUB   $6 $3
                ADD   $6 $3
                ADDI  $6 3
                XORI  $7 5
                EXCH  $9 $6
                XOR   $8 $9
                EXCH  $9 $6
                ADDI  $8 2
                ADD   $8 $7
                XORI  $7 5
                ADDI  $6 -3
                SUB   $6 $3
                EXCH  $10 $8
                ADD   $6 $3
                ADDI  $6 3
                XORI  $7 5
                SUB   $8 $7
                ADDI  $8 -2
                EXCH  $9 $6
                XOR   $8 $9
                EXCH  $9 $6
                XORI  $7 5
                ADDI  $6 -3
                SUB   $6 $3
                XORI  $11 4
assArrElem_50:  ADD   $10 $11
                XORI  $11 4
                ADD   $6 $3
                ADDI  $6 3
                XORI  $7 5
                EXCH  $9 $6
                XOR   $8 $9
                EXCH  $9 $6
                ADDI  $8 2
                ADD   $8 $7
                XORI  $7 5
                ADDI  $6 -3
                SUB   $6 $3
                EXCH  $10 $8
                ADD   $6 $3
                ADDI  $6 3
                XORI  $7 5
                SUB   $8 $7
                ADDI  $8 -2
                EXCH  $9 $6
                XOR   $8 $9
                EXCH  $9 $6
                XORI  $7 5
                ADDI  $6 -3
                SUB   $6 $3
                ADD   $6 $3
                ADDI  $6 5
                XORI  $7 5
                EXCH  $9 $6
                XOR   $8 $9
                EXCH  $9 $6
                ADDI  $8 2
                ADD   $8 $7
                XORI  $7 5
                ADDI  $6 -5
                SUB   $6 $3
                EXCH  $10 $8
                ADD   $6 $3
                ADDI  $6 5
                XORI  $7 5
                SUB   $8 $7
                ADDI  $8 -2
                EXCH  $9 $6
                XOR   $8 $9
                EXCH  $9 $6
                XORI  $7 5
                ADDI  $6 -5
                SUB   $6 $3
                ADD   $11 $3
                ADDI  $11 7
                EXCH  $12 $11
                ADDI  $11 -7
                SUB   $11 $3
assArrElem_51:  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 5
                EXCH  $9 $6
                XOR   $8 $9
                EXCH  $9 $6
                ADDI  $8 2
                ADD   $8 $7
                XORI  $7 5
                ADDI  $6 -5
                SUB   $6 $3
                EXCH  $10 $8
                ADD   $6 $3
                ADDI  $6 5
                XORI  $7 5
                SUB   $8 $7
                ADDI  $8 -2
                EXCH  $9 $6
                XOR   $8 $9
                EXCH  $9 $6
                XORI  $7 5
                ADDI  $6 -5
                SUB   $6 $3
                ADD   $6 $3
                ADDI  $6 6
                XORI  $7 5
                EXCH  $9 $6
                XOR   $8 $9
                EXCH  $9 $6
                ADDI  $8 2
                ADD   $8 $7
                XORI  $7 5
                ADDI  $6 -6
                SUB   $6 $3
```

```
               EXCH  $10 $8
               ADD   $6 $3
               ADDI  $6 6
               XORI  $7 5
               SUB   $8 $7
               ADDI  $8 -2
               EXCH  $9 $6
               XOR   $8 $9
               EXCH  $9 $6
               XORI  $7 5
               ADDI  $6 -6
               SUB   $6 $3
               ADD   $11 $3
               ADDI  $11 8
               EXCH  $12 $11
               ADDI  $11 -8
               SUB   $11 $3
assArrElem_52: ADD   $10 $12
               ADD   $11 $3
               ADDI  $11 8
               EXCH  $12 $11
               ADDI  $11 -8
               SUB   $11 $3
               ADD   $6 $3
               ADDI  $6 6
               XORI  $7 5
               EXCH  $9 $6
               XOR   $8 $9
               EXCH  $9 $6
               ADDI  $8 2
               ADD   $8 $7
               XORI  $7 5
               ADDI  $6 -6
               SUB   $6 $3
               EXCH  $10 $8
               ADD   $6 $3
               ADDI  $6 6
               XORI  $7 5
               SUB   $8 $7
               ADDI  $8 -2
               EXCH  $9 $6
               XOR   $8 $9
               EXCH  $9 $6
               XORI  $7 5
               ADDI  $6 -6
               SUB   $6 $3
               ADD   $6 $3
               ADDI  $6 4
               XORI  $7 5
               EXCH  $9 $6
               XOR   $8 $9
               EXCH  $9 $6
               ADDI  $8 2
               ADD   $8 $7
               XORI  $7 5
               ADDI  $6 -4
               SUB   $6 $3
               EXCH  $10 $8
               ADD   $6 $3
               ADDI  $6 4
               XORI  $7 5
               SUB   $8 $7
               ADDI  $8 -2
               EXCH  $9 $6
               XOR   $8 $9
               EXCH  $9 $6
               XORI  $7 5
               ADDI  $6 -4
               SUB   $6 $3
               XORI  $11 5
assArrElem_53: ADD   $10 $11
               XORI  $11 5
               ADD   $6 $3
               ADDI  $6 4
               XORI  $7 5
               EXCH  $9 $6
               XOR   $8 $9
               EXCH  $9 $6
               ADDI  $8 2
               ADD   $8 $7
               XORI  $7 5
               ADDI  $6 -4
               SUB   $6 $3
               EXCH  $10 $8
               ADD   $6 $3
               ADDI  $6 4
               XORI  $7 5
               SUB   $8 $7
               ADDI  $8 -2
               EXCH  $9 $6
               XOR   $8 $9
               EXCH  $9 $6
               XORI  $7 5
               ADDI  $6 -4
               SUB   $6 $3
               ADD   $6 $3
               ADDI  $6 3
               XORI  $7 6
               EXCH  $9 $6
               XOR   $8 $9
               EXCH  $9 $6
               ADDI  $8 2
               ADD   $8 $7
               XORI  $7 6
               ADDI  $6 -3
               SUB   $6 $3
               EXCH  $10 $8
               ADD   $6 $3
               ADDI  $6 3
               XORI  $7 6
               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_54: ADD   $10 $11
               XORI  $11 5
               ADD   $6 $3
               ADDI  $6 3
               XORI  $7 6
               EXCH  $9 $6
               XOR   $8 $9
               EXCH  $9 $6
               ADDI  $8 2
               ADD   $8 $7
               XORI  $7 6
               ADDI  $6 -3
               SUB   $6 $3
```

```
                    EXCH   $10 $8
                    ADD    $6 $3
                    ADDI   $6 3
                    XORI   $7 6
                    SUB    $8 $7
                    ADDI   $8 -2
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    XORI   $7 6
                    ADDI   $6 -3
                    SUB    $6 $3
                    ADD    $6 $3
                    ADDI   $6 5
                    XORI   $7 6
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    ADDI   $8 2
                    ADD    $8 $7
                    XORI   $7 6
                    ADDI   $6 -5
                    SUB    $6 $3
                    EXCH   $10 $8
                    ADD    $6 $3
                    ADDI   $6 5
                    XORI   $7 6
                    SUB    $8 $7
                    ADDI   $8 -2
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    XORI   $7 6
                    ADDI   $6 -5
                    SUB    $6 $3
assArrElem_55:      ADD    $10 $0
                    ADD    $6 $3
                    ADDI   $6 5
                    XORI   $7 6
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    ADDI   $8 2
                    ADD    $8 $7
                    XORI   $7 6
                    ADDI   $6 -5
                    SUB    $6 $3
                    EXCH   $10 $8
                    ADD    $6 $3
                    ADDI   $6 5
                    XORI   $7 6
                    SUB    $8 $7
                    ADDI   $8 -2
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    XORI   $7 6
                    ADDI   $6 -5
                    SUB    $6 $3
                    ADD    $6 $3
                    ADDI   $6 6
                    XORI   $7 6
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    ADDI   $8 2
                    ADD    $8 $7
                    XORI   $7 6
                    ADDI   $6 -6
                    SUB    $6 $3
                    EXCH   $10 $8
                    ADD    $6 $3
                    ADDI   $6 6
                    XORI   $7 6
                    SUB    $8 $7
                    ADDI   $8 -2
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    XORI   $7 6
                    ADDI   $6 -6
                    SUB    $6 $3
assArrElem_56:      ADD    $10 $0
                    ADD    $6 $3
                    ADDI   $6 6
                    XORI   $7 6
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    ADDI   $8 2
                    ADD    $8 $7
                    XORI   $7 6
                    ADDI   $6 -6
                    SUB    $6 $3
                    EXCH   $10 $8
                    ADD    $6 $3
                    ADDI   $6 6
                    XORI   $7 6
                    SUB    $8 $7
                    ADDI   $8 -2
                    EXCH   $9 $6
                    XOR    $8 $9
                    EXCH   $9 $6
                    XORI   $7 6
                    ADDI   $6 -6
                    SUB    $6 $3
                    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
                    XORI   $11 4
assArrElem_57:      ADD    $10 $11
                    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 7
                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
                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
```

```
                    EXCH    $13 $6
swap_83:            XOR     $12 $13
                    XOR     $13 $12
                    XOR     $12 $13
                    EXCH    $13 $6
                    ADD     $11 $3
                    ADDI    $11 2
                    EXCH    $12 $11
                    ADDI    $11 -2
                    SUB     $11 $3
                    EXCH    $11 $6
                    ADD     $12 $3
                    ADDI    $12 2
                    EXCH    $13 $12
                    ADDI    $12 -2
                    SUB     $12 $3
copy_84:            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
                    ADD     $11 $3
                    ADDI    $11 2
                    EXCH    $12 $11
                    ADDI    $11 -2
                    SUB     $11 $3
                    XOR     $13 $12
loadMetAdd_85:      EXCH    $14 $13
                    ADDI    $14 2
                    EXCH    $15 $14
                    XOR     $16 $15
                    EXCH    $15 $14
                    ADDI    $14 -2
                    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    $7 $1
                    ADDI    $1 -1
                    EXCH    $6 $1
                    ADDI    $1 -1
                    EXCH    $13 $1
                    ADDI    $1 -1
                    ADDI    $16 -2792
l_rjmp_top_87:      RBRA    l_rjmp_bot_88
l_jmp_86:           SWAPBR  $16
                    NEG     $16
l_rjmp_bot_88:      BRA     l_rjmp_top_87
                    ADDI    $16 2792
                    ADDI    $1 1
                    EXCH    $13 $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
                    ADD     $11 $3
                    ADDI    $11 2
                    EXCH    $12 $11
                    ADDI    $11 -2
                    SUB     $11 $3
                    EXCH    $14 $13
                    ADDI    $14 2
                    EXCH    $15 $14
                    XOR     $16 $15
                    EXCH    $15 $14
                    ADDI    $14 -2
loadMetAdd_85_i:    EXCH    $14 $13
                    XOR     $13 $12
                    ADD     $11 $3
                    ADDI    $11 2
                    EXCH    $12 $11
                    ADDI    $11 -2
                    SUB     $11 $3
                    EXCH    $11 $6
                    EXCH    $12 $7
copy_89:            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 $7
                    XOR     $12 $11
loadMetAdd_90:      EXCH    $13 $12
                    ADDI    $13 2
                    EXCH    $14 $13
                    XOR     $15 $14
                    EXCH    $14 $13
                    ADDI    $13 -2
                    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 -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
```

```
                        SUB     $6 $3
                        ADD     $6 $3
                        ADDI    $6 13
                        EXCH    $7 $6
                        ADDI    $6 -13
                        SUB     $6 $3
                        XORI    $8 6
                        ADD     $7 $8
                        XORI    $8 6
                        ADD     $6 $3
                        ADDI    $6 13
                        EXCH    $7 $6
                        ADDI    $6 -13
                        SUB     $6 $3
                        EXCH    $3 $1
                        ADDI    $1 -1
                        BRA     l_simulate_5
                        ADDI    $1 1
                        EXCH    $3 $1
l_init_4_bot:           BRA     l_init_4_top
l_simulate_5_top:       BRA
    l_simulate_5_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $3 $1
                        ADDI    $1 -1
l_simulate_5:           SWAPBR  $2
                        NEG     $2
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
                        XORI    $6 1
entry_150:              BEQ     $6 $0
    assert_152
                        ADD     $7 $3
                        ADDI    $7 11
                        EXCH    $8 $7
                        ADDI    $7 -11
                        SUB     $7 $3
                        ADD     $9 $3
                        ADDI    $9 12
                        EXCH    $10 $9
                        ADDI    $9 -12
                        SUB     $9 $3
cmp_top_154:            BNE     $8 $10
    cmp_bot_155
                        XORI    $11 1
cmp_bot_155:            BNE     $8 $10
    cmp_top_154
f_top_156:              BEQ     $11 $0
    f_bot_157
                        XORI    $12 1
f_bot_157:              BEQ     $11 $0
    f_top_156
                        XOR     $6 $12
f_bot_157_i:            BEQ     $11 $0
    f_top_156_i
                        XORI    $12 1
f_top_156_i:            BEQ     $11 $0
    f_bot_157_i
cmp_bot_155_i:          BNE     $8 $10
    cmp_top_154_i
                        XORI    $11 1
cmp_top_154_i:          BNE     $8 $10
    cmp_bot_155_i
                        ADD     $9 $3
                        ADDI    $9 12
                        EXCH    $10 $9
                        ADDI    $9 -12
                        SUB     $9 $3
                        ADD     $7 $3
                        ADDI    $7 11
                        EXCH    $8 $7
                        ADDI    $7 -11
                        SUB     $7 $3
                        ADD     $7 $3
                        ADDI    $7 2
                        EXCH    $8 $7
                        ADDI    $7 -2
                        SUB     $7 $3
                        XOR     $9 $8
loadMetAdd_158:         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 -3745
l_jmp_159:              SWAPBR  $12
                        NEG     $12
                        ADDI    $12 3745
                        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_158_i:       EXCH    $10 $9
                        XOR     $9 $8
                        ADD     $7 $3
                        ADDI    $7 2
                        EXCH    $8 $7
                        ADDI    $7 -2
                        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
```

```
                  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
cmp_top_184:      BNE    $8 $14
    cmp_bot_185
                  XORI   $15 1
cmp_bot_185:      BNE    $8 $14
    cmp_top_184
                  ADD    $16 $3
                  ADDI   $16 14
                  EXCH   $17 $16
                  ADDI   $16 -14
                  SUB    $16 $3
                  ADD    $18 $3
                  ADDI   $18 5
                  ADD    $19 $3
                  ADDI   $19 16
                  EXCH   $20 $19
                  ADDI   $19 -16
                  SUB    $19 $3
                  EXCH   $22 $18
                  XOR    $21 $22
                  EXCH   $22 $18
                  ADDI   $21 2
                  ADD    $21 $20
                  ADD    $19 $3
                  ADDI   $19 16
                  EXCH   $20 $19
                  ADDI   $19 -16
                  SUB    $19 $3
                  ADDI   $18 -5
                  SUB    $18 $3
                  EXCH   $23 $21
                  ADD    $18 $3
                  ADDI   $18 5
                  ADD    $19 $3
                  ADDI   $19 16
                  EXCH   $20 $19
                  ADDI   $19 -16
                  SUB    $19 $3
                  SUB    $21 $20
                  ADDI   $21 -2
                  EXCH   $22 $18
                  XOR    $21 $22
                  EXCH   $22 $18
                  ADD    $19 $3
                  ADDI   $19 16
                  EXCH   $20 $19
                  ADDI   $19 -16
                  SUB    $19 $3
                  ADDI   $18 -5
                  SUB    $18 $3
cmp_top_186:      BNE    $17 $23
    cmp_bot_187
                  XORI   $24 1
cmp_bot_187:      BNE    $17 $23
    cmp_top_186
                  ANDX   $25 $15 $24
f_top_188:        BEQ    $25 $0
    f_bot_189
                  XORI   $26 1
f_bot_189:        BEQ    $25 $0
    f_top_188
                  XOR    $6 $26
f_bot_189_i:      BEQ    $25 $0
    f_top_188_i
                  XORI   $26 1
f_top_188_i:      BEQ    $25 $0
    f_bot_189_i
                  ANDX   $25 $15 $24
cmp_bot_187_i:    BNE    $17 $23
    cmp_top_186_i
                  XORI   $24 1
cmp_top_186_i:    BNE    $17 $23
    cmp_bot_187_i
                  ADD    $18 $3
                  ADDI   $18 5
                  ADD    $19 $3
                  ADDI   $19 16
                  EXCH   $20 $19
                  ADDI   $19 -16
                  SUB    $19 $3
                  EXCH   $22 $18
                  XOR    $21 $22
                  EXCH   $22 $18
                  ADDI   $21 2
                  ADD    $21 $20
                  ADD    $19 $3
                  ADDI   $19 16
                  EXCH   $20 $19
                  ADDI   $19 -16
                  SUB    $19 $3
                  ADDI   $18 -5
                  SUB    $18 $3
                  EXCH   $23 $21
                  ADD    $18 $3
                  ADDI   $18 5
                  ADD    $19 $3
                  ADDI   $19 16
                  EXCH   $20 $19
                  ADDI   $19 -16
                  SUB    $19 $3
                  SUB    $21 $20
                  ADDI   $21 -2
                  EXCH   $22 $18
                  XOR    $21 $22
                  EXCH   $22 $18
                  ADD    $19 $3
                  ADDI   $19 16
                  EXCH   $20 $19
                  ADDI   $19 -16
                  SUB    $19 $3
                  ADDI   $18 -5
                  SUB    $18 $3
                  ADD    $16 $3
                  ADDI   $16 14
                  EXCH   $17 $16
                  ADDI   $16 -14
                  SUB    $16 $3
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
```

```
ADDI $15 -3
SUB  $15 $3
XOR  $21 $14
SUB  $21 $20
ADD  $8 $21
ADD  $21 $20
XOR  $21 $14
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
ADDI $15 -3
SUB  $15 $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  $7 $3
ADDI $7 11
EXCH $8 $7
ADDI $7 -11
SUB  $7 $3
ADD  $7 $3
ADDI $7 14
EXCH $8 $7
ADDI $7 -14
SUB  $7 $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   $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
```

```
assert_183:                BNE   $6 $0
    assert_true_181
                           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_190:               BNE   $8 $14
    cmp_bot_191
                           XORI  $15 1
cmp_bot_191:               BNE   $8 $14
    cmp_top_190
                           ADD   $16 $3
                           ADDI  $16 14
                           EXCH  $17 $16
                           ADDI  $16 -14
                           SUB   $16 $3
                           ADD   $18 $3
                           ADDI  $18 6
                           ADD   $19 $3
                           ADDI  $19 16
                           EXCH  $20 $19
                           ADDI  $19 -16
                           SUB   $19 $3
                           EXCH  $22 $18
                           XOR   $21 $22
                           EXCH  $22 $18
                           ADDI  $21 2
                           ADD   $21 $20
                           ADD   $19 $3
                           ADDI  $19 16
                           EXCH  $20 $19
                           ADDI  $19 -16
                           SUB   $19 $3
                           ADDI  $18 -6
                           SUB   $18 $3
                           EXCH  $23 $21
                           ADD   $18 $3
                           ADDI  $18 6
                           ADD   $19 $3
                           ADDI  $19 16
                           EXCH  $20 $19
                           ADDI  $19 -16
                           SUB   $19 $3
                           SUB   $21 $20
                           ADDI  $21 -2
                           EXCH  $22 $18
                           XOR   $21 $22
                           EXCH  $22 $18
                           ADD   $19 $3
                           ADDI  $19 16
                           EXCH  $20 $19
                           ADDI  $19 -16
                           SUB   $19 $3
                           ADDI  $18 -6
                           SUB   $18 $3
cmp_top_192:               BNE   $17 $23
    cmp_bot_193
                           XORI  $24 1
cmp_bot_193:               BNE   $17 $23
    cmp_top_192
                           ANDX  $25 $15 $24
f_top_194:                 BEQ   $25 $0
    f_bot_195
                           XORI  $26 1
f_bot_195:                 BEQ   $25 $0
    f_top_194
                           XOR   $6 $26
f_bot_195_i:               BEQ   $25 $0
    f_top_194_i
                           XORI  $26 1
f_top_194_i:               BEQ   $25 $0
    f_bot_195_i
                           ANDX  $25 $15 $24
cmp_bot_193_i:             BNE   $17 $23
    cmp_top_192_i
                           XORI  $24 1
cmp_top_192_i:             BNE   $17 $23
    cmp_bot_193_i
                           ADD   $18 $3
                           ADDI  $18 6
                           ADD   $19 $3
                           ADDI  $19 16
                           EXCH  $20 $19
                           ADDI  $19 -16
                           SUB   $19 $3
                           EXCH  $22 $18
                           XOR   $21 $22
                           EXCH  $22 $18
                           ADDI  $21 2
                           ADD   $21 $20
                           ADD   $19 $3
                           ADDI  $19 16
```

```
                        EXCH  $20 $19
                        ADDI  $19 -16
                        SUB   $19 $3
                        ADDI  $18 -6
                        SUB   $18 $3
                        EXCH  $23 $21
                        ADD   $18 $3
                        ADDI  $18 6
                        ADD   $19 $3
                        ADDI  $19 16
                        EXCH  $20 $19
                        ADDI  $19 -16
                        SUB   $19 $3
                        SUB   $21 $20
                        ADDI  $21 -2
                        EXCH  $22 $18
                        XOR   $21 $22
                        EXCH  $22 $18
                        ADD   $19 $3
                        ADDI  $19 16
                        EXCH  $20 $19
                        ADDI  $19 -16
                        SUB   $19 $3
                        ADDI  $18 -6
                        SUB   $18 $3
                        ADD   $16 $3
                        ADDI  $16 14
                        EXCH  $17 $16
                        ADDI  $16 -14
                        SUB   $16 $3
cmp_bot_191_i:          BNE   $8 $14
    cmp_top_190_i
                        XORI  $15 1
cmp_top_190_i:          BNE   $8 $14
    cmp_bot_191_i
                        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   $7 $3
                        ADDI  $7 11
                        EXCH  $8 $7
                        ADDI  $7 -11
                        SUB   $7 $3
                        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
                        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
cmp_top_200:            BNE   $8 $14
    cmp_bot_201
                        XORI  $15 1
cmp_bot_201:            BNE   $8 $14
    cmp_top_200
                        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_202:    BNE     $21 $23
    cmp_bot_203
                XORI    $24 1
cmp_bot_203:    BNE     $21 $23
    cmp_top_202
                ANDX    $25 $15 $24
f_top_204:      BEQ     $25 $0
    f_bot_205
                XORI    $26 1
f_bot_205:      BEQ     $25 $0
    f_top_204
                XOR     $6 $26
f_bot_205_i:    BEQ     $25 $0
    f_top_204_i
                XORI    $26 1
f_top_204_i:    BEQ     $25 $0
    f_bot_205_i
                ANDX    $25 $15 $24
cmp_bot_203_i:  BNE     $21 $23
    cmp_top_202_i
                XORI    $24 1
cmp_top_202_i:  BNE     $21 $23
    cmp_bot_203_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
                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
cmp_bot_201_i:  BNE     $8 $14
    cmp_top_200_i
                XORI    $15 1
cmp_top_200_i:  BNE     $8 $14
    cmp_bot_201_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_196:   BEQ     $6 $0 test_false_198
            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
            ADDI    $15 -3
            SUB     $15 $3
            XOR     $21 $14
            SUB     $21 $20
            ADD     $8 $21
            ADD     $21 $20
            XOR     $21 $14
            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
            ADDI  $15 -3
            SUB   $15 $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   $7 $3
            ADDI  $7 11
            EXCH  $8 $7
            ADDI  $7 -11
            SUB   $7 $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
            ADD   $14 $3
            ADDI  $14 9
            EXCH  $15 $14
            ADDI  $14 -9
            SUB   $14 $3
cmp_top_210:   BNE   $13 $15 cmp_bot_211
            XORI  $16 1
cmp_bot_211:   BNE   $13 $15 cmp_top_210
f_top_212:     BEQ   $16 $0 f_bot_213
            XORI  $17 1
f_bot_213:     BEQ   $16 $0 f_top_212
            XOR   $7 $17
f_bot_213_i:   BEQ   $16 $0 f_top_212_i
            XORI  $17 1
f_top_212_i:   BEQ   $16 $0 f_bot_213_i
cmp_bot_211_i: BNE   $13 $15 cmp_top_210_i
            XORI  $16 1
cmp_top_210_i: BNE   $13 $15 cmp_bot_211_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
                        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:
    test_false_208      BEQ    $7 $0
                        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:
    assert_true_207     BNE    $7 $0
                        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 9
                        EXCH   $15 $14
                        ADDI   $14 -9
                        SUB    $14 $3
cmp_top_214:
    cmp_bot_215         BNE    $13 $15
                        XORI   $16 1
cmp_bot_215:
    cmp_top_214         BNE    $13 $15
f_top_216:
    f_bot_217           BEQ    $16 $0
                        XORI   $17 1
f_bot_217:
    f_top_216           BEQ    $16 $0
                        XOR    $7 $17
f_bot_217_i:
    f_top_216_i         BEQ    $16 $0
                        XORI   $17 1
f_top_216_i:
    f_bot_217_i         BEQ    $16 $0
cmp_bot_215_i:
    cmp_top_214_i       BNE    $13 $15
                        XORI   $16 1
cmp_top_214_i:
    cmp_bot_215_i       BNE    $13 $15
                        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
                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_222:    BNE     $13 $15
    cmp_bot_223
                XORI    $16 1
cmp_bot_223:    BNE     $13 $15
    cmp_top_222
f_top_224:      BEQ     $16 $0
    f_bot_225
                XORI    $17 1
f_bot_225:      BEQ     $16 $0
    f_top_224
                XOR     $7 $17
f_bot_225_i:    BEQ     $16 $0
    f_top_224_i
                XORI    $17 1
f_top_224_i:    BEQ     $16 $0
    f_bot_225_i
cmp_bot_223_i:  BNE     $13 $15
    cmp_top_222_i
                XORI    $16 1
cmp_top_222_i:  BNE     $13 $15
    cmp_bot_223_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
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
```

```
                     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
cmp_bot_231_i:       BNE    $8 $14
    cmp_top_230_i
                     XORI   $15 1
cmp_top_230_i:       BNE    $8 $14
    cmp_bot_231_i
                     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    $7 $3
                     ADDI   $7 11
                     EXCH   $8 $7
                     ADDI   $7 -11
                     SUB    $7 $3
l_inst_6_bot:        BRA    l_inst_6_top
l_moveRight_7_top:   BRA
    l_moveRight_7_bot
                     ADDI   $1 1
                     EXCH   $2 $1
                     EXCH   $3 $1
                     ADDI   $1 -1
l_moveRight_7:       SWAPBR $2
                     NEG    $2
                     ADDI   $1 1
                     EXCH   $3 $1
                     EXCH   $2 $1
                     ADDI   $1 -1
localBlock_302:      XOR    $6 $1
                     XOR    $7 $0
                     EXCH   $7 $1
                     ADDI   $1 -1
localBlock_301:      XOR    $7 $1
                     XOR    $8 $0
                     EXCH   $8 $1
                     ADDI   $1 -1
                     ADD    $8 $3
                     ADDI   $8 2
                     EXCH   $9 $8
                     ADDI   $8 -2
                     SUB    $8 $3
                     XOR    $10 $9
loadMetAdd_236:      EXCH   $11 $10
                     ADDI   $11 3
                     EXCH   $12 $11
                     XOR    $13 $12
                     EXCH   $12 $11
                     ADDI   $11 -3
                     EXCH   $11 $10
                     ADD    $8 $3
                     ADDI   $8 2
                     EXCH   $9 $8
                     ADDI   $8 -2
                     SUB    $8 $3
                     ADD    $14 $3
                     ADDI   $14 14
                     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 -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
```

```
                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
cmp_bot_247_i:  BNE    $9 $0 cmp_top_246_i
                XORI   $10 1
cmp_top_246_i:  BNE    $9 $0 cmp_bot_247_i
                EXCH   $9 $6
test_242:       BEQ    $8 $0 test_false_244
                XORI   $8 1
                ADD    $9 $3
                ADDI   $9 14
                EXCH   $10 $9
                ADDI   $9 -14
                SUB    $9 $3
                ADD    $11 $3
                ADDI   $11 10
                EXCH   $12 $11
                ADDI   $11 -10
                SUB    $11 $3
                XOR    $10 $12
                ADD    $11 $3
                ADDI   $11 10
                EXCH   $12 $11
                ADDI   $11 -10
                SUB    $11 $3
                ADD    $9 $3
                ADDI   $9 14
                EXCH   $10 $9
                ADDI   $9 -14
                SUB    $9 $3
                EXCH   $3 $1
                ADDI   $1 -1
                EXCH   $7 $1
                ADDI   $1 -1
                EXCH   $6 $1
                ADDI   $1 -1
obj_con_252:    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_252_i:  ADDI   $10 -8
                ADDI   $1 1
                EXCH   $6 $1
                ADDI   $1 1
                EXCH   $7 $1
                ADDI   $1 1
                EXCH   $3 $1
                XORI   $10 10
                EXCH   $10 $9
                ADDI   $9 1
                XORI   $10 1
                EXCH   $10 $9
obj_con_252_bot: ADDI  $9 -1
                EXCH   $9 $6
                EXCH   $9 $7
                EXCH   $10 $6
copy_253:       XOR    $9 $10
                ADDI   $10 1
                EXCH   $11 $10
                ADDI   $11 1
                EXCH   $11 $10
                ADDI   $10 -1
                EXCH   $10 $6
                EXCH   $9 $7
                EXCH   $9 $6
                XOR    $10 $9
loadMetAdd_254: EXCH   $11 $10
                ADDI   $11 2
                EXCH   $12 $11
                XOR    $13 $12
                EXCH   $12 $11
                ADDI   $11 -2
                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 -5989
l_rjmp_top_256: RBRA   l_rjmp_bot_257
l_jmp_255:      SWAPBR $13
                NEG    $13
l_rjmp_bot_257: BRA    l_rjmp_top_256
                ADDI   $13 5989
                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 2
                EXCH   $12 $11
                XOR    $13 $12
                EXCH   $12 $11
                ADDI   $11 -2
loadMetAdd_254_i: EXCH $11 $10
                XOR    $10 $9
                EXCH   $9 $6
                EXCH   $9 $6
                XOR    $10 $9
loadMetAdd_258: EXCH   $11 $10
                ADDI   $11 0
                EXCH   $12 $11
                XOR    $13 $12
                EXCH   $12 $11
                ADDI   $11 0
                EXCH   $11 $10
                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
```

```
                         ADDI   $15 10
                         EXCH   $16 $15
                         ADDI   $15 -10
                         SUB    $15 $3
cmp_top_272:             BNE    $14 $16
    cmp_bot_273
                         XORI   $17 1
cmp_bot_273:             BEQ    $14 $16
    cmp_top_272
                         ANDX   $18 $12 $17
f_top_274:               BEQ    $18 $0
    f_bot_275
                         XORI   $19 1
f_bot_275:               BEQ    $18 $0
    f_top_274
                         XOR    $9 $19
f_bot_275_i:             BEQ    $18 $0
    f_top_274_i
                         XORI   $19 1
f_top_274_i:             BEQ    $18 $0
    f_bot_275_i
                         ANDX   $18 $12 $17
cmp_bot_273_i:           BNE    $14 $16
    cmp_top_272_i
                         XORI   $17 1
cmp_top_272_i:           BNE    $14 $16
    cmp_bot_273_i
                         ADD    $15 $3
                         ADDI   $15 10
                         EXCH   $16 $15
                         ADDI   $15 -10
                         SUB    $15 $3
                         ADD    $13 $3
                         ADDI   $13 14
                         EXCH   $14 $13
                         ADDI   $13 -14
                         SUB    $13 $3
cmp_bot_271_i:           BNE    $11 $0
    cmp_top_270_i
                         XORI   $12 1
cmp_top_270_i:           BNE    $11 $0
    cmp_bot_271_i
                         ADD    $10 $3
                         ADDI   $10 2
                         EXCH   $11 $10
                         ADDI   $10 -2
                         SUB    $10 $3
test_266:                BEQ    $9 $0
    test_false_268
                         XORI   $9 1
                         EXCH   $10 $7
                         XOR    $11 $10
loadMetAdd_276:          EXCH   $12 $11
                         ADDI   $12 2
                         EXCH   $13 $12
                         XOR    $14 $13
                         EXCH   $13 $12
                         ADDI   $12 -2
                         EXCH   $12 $11
                         EXCH   $10 $7
                         ADD    $15 $3
                         ADDI   $15 2
                         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 -6213
l_jmp_277:               SWAPBR $14
                         NEG    $14
                         ADDI   $14 6213
                         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 -2
                         SUB    $15 $3
                         EXCH   $10 $7
                         EXCH   $12 $11
                         ADDI   $12 2
                         EXCH   $13 $12
                         XOR    $14 $13
                         EXCH   $13 $12
                         ADDI   $12 -2
loadMetAdd_276_i:        EXCH   $12 $11
                         XOR    $11 $10
                         EXCH   $10 $7
                         ADD    $10 $3
                         ADDI   $10 2
                         EXCH   $11 $10
                         ADDI   $10 -2
                         SUB    $10 $3
                         EXCH   $12 $7
uncopy_278:              XOR    $11 $12
                         ADDI   $12 1
                         EXCH   $13 $12
                         ADDI   $13 -1
                         EXCH   $13 $12
                         ADDI   $12 -1
                         EXCH   $12 $7
                         ADD    $10 $3
                         ADDI   $10 2
                         EXCH   $11 $10
                         ADDI   $10 -2
                         SUB    $10 $3
                         EXCH   $10 $7
obj_des_279_top:         EXCH   $11 $10
                         XORI   $11 10
                         ADDI   $10 1
                         EXCH   $11 $10
                         XORI   $11 1
                         ADDI   $10 -1
                         EXCH   $3 $1
                         ADDI   $1 -1
                         EXCH   $7 $1
                         ADDI   $1 -1
                         EXCH   $6 $1
                         ADDI   $1 -1
obj_des_279:             ADDI   $11 8
                         EXCH   $11 $1
                         ADDI   $1 -1
```

```
                            EXCH    $10 $1
                            ADDI    $1 -1
                            RBRA    l_malloc
                            ADDI    $1 1
                            EXCH    $10 $1
                            ADDI    $1 1
                            EXCH    $11 $1
obj_des_279_i:              ADDI    $11 -8
                            ADDI    $1 1
                            EXCH    $6 $1
                            ADDI    $1 1
                            EXCH    $7 $1
                            ADDI    $1 1
                            EXCH    $3 $1
                            EXCH    $10 $7
                            ADD     $10 $3
                            ADDI    $10 14
                            EXCH    $11 $10
                            ADDI    $10 -14
                            SUB     $10 $3
                            ADD     $12 $3
                            ADDI    $12 10
                            EXCH    $13 $12
                            ADDI    $12 -10
                            SUB     $12 $3
                            XOR     $11 $13
                            ADD     $12 $3
                            ADDI    $12 10
                            EXCH    $13 $12
                            ADDI    $12 -10
                            SUB     $12 $3
                            ADD     $10 $3
                            ADDI    $10 14
                            EXCH    $11 $10
                            ADDI    $10 -14
                            SUB     $10 $3
                            XORI    $9 1
assert_true_267:            BRA     assert_269
test_false_268:             BRA     test_266
assert_269:                 BNE     $9 $0
      assert_true_267
                            EXCH    $10 $7
cmp_top_280:                BNE     $10 $0
      cmp_bot_281
                            XORI    $11 1
cmp_bot_281:                BNE     $10 $0
      cmp_top_280
f_top_282:                  BEQ     $11 $0
      f_bot_283
                            XORI    $12 1
f_bot_283:                  BEQ     $11 $0
      f_top_282
                            XOR     $9 $12
f_bot_283_i:                BEQ     $11 $0
      f_top_282_i
                            XORI    $12 1
f_top_282_i:                BEQ     $11 $0
      f_bot_283_i
cmp_bot_281_i:              BNE     $10 $0
      cmp_top_280_i
                            XORI    $11 1
cmp_top_280_i:              BNE     $10 $0
      cmp_bot_281_i
                            EXCH    $10 $7
                            EXCH    $9 $6
                            XOR     $10 $9
loadMetAdd_284:             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 -6347
l_rjmp_top_286:             RBRA    l_rjmp_bot_287
l_jmp_285:                  SWAPBR  $13
                            NEG     $13
l_rjmp_bot_287:             BRA     l_rjmp_top_286
                            ADDI    $13 6347
                            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_284_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 $6
swap_288:                   XOR     $10 $11
                            XOR     $11 $10
                            XOR     $10 $11
                            EXCH    $11 $6
                            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_289:             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   $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
                                EXCH   $9 $6
                                ADD    $8 $3
                                ADDI   $8 2
                                EXCH   $9 $8
                                ADDI   $8 -2
                                SUB    $8 $3
                                XOR    $10 $9
loadMetAdd_297:                 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 -6548
l_rjmp_top_299:                 RBRA   l_rjmp_bot_300
l_jmp_298:                      SWAPBR $13
                                NEG    $13
l_rjmp_bot_300:                 BRA    l_rjmp_top_299
                                ADDI   $13 6548
                                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_297_i:               EXCH   $11 $10
                                XOR    $10 $9
                                ADD    $8 $3
                                ADDI   $8 2
                                EXCH   $9 $8
                                ADDI   $8 -2
                                SUB    $8 $3
                                ADDI   $1 1
                                EXCH   $8 $1
                                XOR    $8 $0
localBlock_301_i:               XOR    $7 $1
                                ADDI   $1 1
                                EXCH   $7 $1
                                XOR    $7 $0
localBlock_302_i:               XOR    $6 $1
l_moveRight_7_bot:              BRA
    l_moveRight_7_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_303:                    ADDI   $8 32
                                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_303_i:                  ADDI   $8 -32
                                ADDI   $1 1
                                EXCH   $3 $1
                                ADD    $6 $3
                                ADDI   $6 2
                                XORI   $8 3
                                EXCH   $8 $7
                                ADDI   $7 1
                                XORI   $8 1
                                EXCH   $8 $7
obj_con_303_bot:                ADDI   $7 -1
                                EXCH   $7 $6
                                ADDI   $6 -2
                                SUB    $6 $3
                                ADD    $6 $3
                                ADDI   $6 2
                                EXCH   $7 $6
                                ADDI   $6 -2
                                SUB    $6 $3
                                XOR    $8 $7
loadMetAdd_304:                 EXCH   $9 $8
                                ADDI   $9 3
                                EXCH   $10 $9
                                XOR    $11 $10
                                EXCH   $10 $9
                                ADDI   $9 -3
                                EXCH   $9 $8
                                ADD    $6 $3
                                ADDI   $6 2
                                EXCH   $7 $6
                                ADDI   $6 -2
                                SUB    $6 $3
                                EXCH   $3 $1
                                ADDI   $1 -1
                                EXCH   $8 $1
                                ADDI   $1 -1
                                ADDI   $11 -6647
l_jmp_305:                      SWAPBR $11
                                NEG    $11
```

```
                     ADDI   $11 6647
                     ADDI   $1 1
                     EXCH   $8 $1
                     ADDI   $1 1
                     EXCH   $3 $1
                     ADD    $6 $3
                     ADDI   $6 2
                     EXCH   $7 $6
                     ADDI   $6 -2
                     SUB    $6 $3
                     EXCH   $9 $8
                     ADDI   $9 3
                     EXCH   $10 $9
                     XOR    $11 $10
                     EXCH   $10 $9
                     ADDI   $9 -3
loadMetAdd_304_i:    EXCH   $9 $8
                     XOR    $8 $7
                     ADD    $6 $3
                     ADDI   $6 2
                     EXCH   $7 $6
                     ADDI   $6 -2
                     SUB    $6 $3
l_main_0_bot:        BRA    l_main_0_top
start:               BRA    top
                     START
                     ADDI   $4 6715
                     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 2
                     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   $1 1
                     EXCH   $6 $3
                     XORI   $6 2
                     XOR    $3 $1
                     ADDI   $1 4
                     ADDI   $1 -2048
                     XOR    $1 $5
                     ADDI   $5 -10
                     XOR    $5 $4
                     ADDI   $4 -6715
finish:              FINISH
```