Bali++ Specifications

Part 2

CS212 Fall 2005

1 Introduction

1.1 Bali Specifications

As discussed in lecture, we can specify a language in terms of its syntax (spelling and structural rules) and semantics (meaning). Although formal description methods exist for both kinds of specifications, we will only formalize the Bali syntax, as developed in Section 2. Bali's semantics are informally defined in Section 3.

1.2 Notation

We use the following notation throughout this document:

- Specific language elements, like keywords, operators, and punctuation, are shown as **`bold`**.
- Other terminals are shown as ***`bolditalic`***, such as names and integers, because they can have different values.
- Non-terminals in production rules are shown as plain red.
- Square brackets are used to denote optional items. [**`int`**] indicates that the keyword **`int`** can be present but is not required.
- A single | denotes that either the item on the left or the right can be present. So **`a`** | **`b`** indicates that either **`a`** or **`b`** must be present, but not both.
- Parentheses are used to group options. Thus, (**`a`** | **`b`**) **`c`** indicates that **`c`** must be prefixed by either **`a`** or **`b`**. Square brackets can also be used for grouping with the difference that the group would be optional.
- An asterik (*) is used to indicate zero or more occurrences of this item and a plus (+) is used to indicate one or more occurrences of this item.
- An arrow (→) indicates a production rule, which means *can be expressed as*.

2 Bali Syntax

2.1 Functions

program	→ [globalVars] function*	program has globals and functions
globalVars	→ **global** varBlock	global variables
function	→ functionHeader varBlock statementBlock	function has header, variables and statements
functionHeader	→ type name **( )**	header
type	→ **int** | **boolean** | **char**	primitive/pointer types
varBlock	→ **{** varDecl* **}**	block of variable declarations
statementBlock	→ **{** statement* **}**	block of statements
varDecl	→ type name (**,** name)* **;**	can declare more than one variable

2.2 Statements

statement	→ statementBlock	block statement
statement	→ **if (** expression **)** statement [**else** statement]	conditional statement
statement	→ **do** statement **until (** expression **);**	do...until loop
statement	→ **for (** [expression] **;** [expression] **;** [expression] **)** statement	for loop
statement	→ expression **;**	expression statement
statement	→ **print** expression **;**	output
statement	→ **return** expression **;**	return statement
statement	→ **;**	empty statement

2.3 Expressions

expression	→ expPart [binop expPart]	basic expression
expression	→ expPart **=** expression	assignment
expPart	→ unaryop expPart	unary operation
expPart	→ ***int*** | ***boolean*** | ***'char'***	constant
expPart	→ **readInt()**	integer input
expPart	→ **readChar()**	character input
expPart	→ name	variable
expPart	→ **(** expression **)**	nested expression
binop	→ arithmeticOp | comparisonOp | booleanOp	operators
arithmeticOp	→ **+** |**−**| ***** | **/** | **%**	arithmetic operators
comparisonOp	→ **==** | **!=** | **>** | **<** | **>=** | **<=**	comparison operators
booleanOp	→ **&&** | **||** | **^**	boolean operators (**&&** and **||** are short-circuiting)
unaryOp	→ **−** | **!**	unary operators
name	→ (**a−z**|**A−Z**) (**a−z**|**A**−**Z**|_|**0−9**)*	names must begin with a letter

3 Bali Semantics

3.1 Reserved Keywords

The following keywords may not be used as variable or function names: **`global`**, **`int`**, **`boolean`**, **`char`**, **`void`**, **`true`**, **`false`**, **`if`**, **`else`**, **`do`**, **`until`**, **`for`**, **`print`**, **`null`**, **`return`**, **`free`**, **`malloc`**, **`readInt`**, and **`readChar`**.

3.2 Variable Scope

The scope of a variable depends on where it is defined. Variables defined in the global block are visible throughout the program. Function arguments and function variables are visible in the function in which they are defined. No two variables in the same scope can have the same name (i.e. no two global variables can have the same name and for a given function no argument or function variable can have the same name as another argument or function variable in that same function). Global variables can be redefined inside a function (variable shadowing). Variables can have the same name as functions.

3.3 Expressions

Operators

Variable types can only be mixed in expressions under certain circumstances. Logical operators accept and produce booleans. The equality and inequality operators accept all types and produce a boolean result. However, the types of the two operands must be the same. All other relational comparison operators accept only integers and produce a boolean result. Division is integral for integers. Arithmetic operators accept two integers, and produce and integer.

The assignment operator must have a valid variable on the left that is the same type as the output of the expression on the right. It assigns the value of the expression on the right to the variable on the left and produces a result of the same type and value as the value assigned.

Execution Order

Binary operations should be evaluated starting with the expression on the left and then the expression on the right. So for **`a + b`**, **`a`** should be evaluated first, then **`b`**, and then their results should be added. Assignment expressions should be evaluated from right to left.

Constants

Any Java integer is a constant of type **`int`**. The keywords **`true`** and **`false`** are constants of type **`boolean`**. Any Java character delimited by single quotes (') is a constant of type **`char`**.

3.4 Statements

Each statement type that requires expressions requries a particular type of expression. **`if`** and **`do..until`** statements require an expression returning a boolean value. The **`for`** statement's second expression must return a boolean value. A **`print`** statement requires an expression returning an integer, character, or character pointer. It uses the appropriate instructions to generate integer, character, or string output, respectively. A **`return`** statement requires an expression returning a type identical to that of the function return type. Finally an expression statement can have an expression with any return type, since the return value is eliminated.

Bali supports two kinds of loop statements. The `do..until` loop executes the statement, and loops if its condition evaluates to false. The `for` loop executes the first expression (if present) before entering the loop cycle. Then it checks the second expression, and if true, executes the statement. If there is no second expression, true is assumed. Then the third expression is executed (if present), and the code loops.

3.5 Functions

- All functions contain an implicit return statement at the end of the function that returns `0` for integers, `'\0'` for characters, and `false` for booleans. Thus a function will return something even if no specific return statement is encountered.
- There must be one function called `main` with a return type of `int` and no parameters. This function will be called on startup.
- There are two built-in functions. The `readInt` and `readChar` functions request an integer or a chracter, respectively, from the user.