ALACS
Language Reference Manual

Manager: Gabriel Lopez (gal2129)
Language Guru: Gabriel Kramer-Garcia (glk2110)
System Architect: Candace Johnson (crj2121)
Tester: Terence Jacobs (tj2316)

Table of Contents

I. Introduction

Alacs is an imperative object-oriented language. The goal of our project is to develop an object-oriented language that incorporates lambda expressions and multiple inheritance. Our language will compile to LLVM.

II. Data Types

num

num is a type that stores a supplied value in 4 bytes. The type will be used to store all real numbers whether they are integers or not. Initialization is shown below.

```
num x = 10; /*x will be assigned the value 10.*/
num y = 10.5;/*y will be assigned the value 10.5*/
num z = x * y; /*z will be assigned the value 105. (Note the decimal point after the 5)*/
```

bool

bool is a binary indicator that can be either true or false.

```
bool x = true;
bool y = false;
if (x and y) /*will evaluate to false*/
{
	bool done = true;
}
elif (x or y) /*will be true*/
{
	bool done = false;
}
```

char

char is a single alphabetic ASCII character between single quotes. The range is ‘a’ - ‘z’, ‘A’ - ‘Z’. Initialization is shown below

```
char x = ‘x’;
```

void

void is used to imply that a method will not return a value. Below is a sample method declaration:

```
void method (parameters)
{
        /*method body*/
}
```

Arrays

Arrays are container objects that hold a fixed number of values of a single type, and a fixed length

```
int intarray[7];        /*integer array of length 7*/
```

Strings

An instance of a string object contains an array of the primitive datatype char. Alacs supports three methods to manipulate strings: substring(), charAt(), length().

```
String alacs =  "ALACS";
String lacs = alacs.substring(1, 4);   /*lacs = "LACS"*/
char a = lacs.charAt(1);  /* a = 'A'*/
int len = alacs.length();   /*len = 5*/
```

III. Lexical Conventions

A. Identifiers

Identifiers are sequences of ASCII characters used for naming variables, methods, and classes.

B. Literals

Literals are numeric, characters or string values.

1. Numeric Literals

Numeric literals are defined by: sign part, a number part, and a decimal point. These literals are used to identify the **num** data type. If there is a '-' sign, this indicates the value is negative. Otherwise, all numbers will be interpreted as positive.Valid numeric literals are shown below:

1

-1

4.1

+5.50

0.00

2. Character Literals

Character literals are single ASCII characters contained in single quotes. Valid character literals are shown below:

'P'

'l'

'4'

3. String Literals

A string literal are a sequence of characters enclosed in double quotes. A string is considered as an array of characters. Valid string literals are shown below:

"Hello World"

"COMS 4115"

C. Comments

Comments are enclosed by /* */. Anything after /* will be ignored until a */ is seen. You cannot nest comments. For example, in the line /*hello/*bye*/hi*/, the comment ends after bye. An example of a valid comment is shown below:

```
/*This is a comment in Alacs*/
```

D. Whitespace

Whitespace is defined by any sequence of single space characters, tab characters, or newline characters. Whitespace will be ignored by the compiler.

E. Operators

Operators will include arithmetic, relational, boolean and logical operators**.**

1. Arithmetic Operators

Arithmetic operators take numeric values as their operands and return a single numeric value. Arithmetic operators will have precedence based on the order of operations. Parentheses, exponents, multiplication/division and addition/subtraction will have precedence in that order. If two statements have similar operators (ex. Addition and subtraction) they will operate in left associative order.

+
  Adds two numbers
-
  Subtracts two numbers
*
  Multiplies two numbers
/
  Divides two numbers
%
  Divides two numbers and returns the remainder
^
  Exponent

2. Relational Operators

Relational operators test for relations between the two values. Returns boolean value.

==

Compares two values to see if they are the same. Evaluates to true if they are; false otherwise.

!=

Compares two values to see if they are equal. Evaluates to true if they are not; false otherwise.

>

Compares left value to right value to see if left value is greater. Evaluates to true if it is; false otherwise.

<

Compares left value to right value to see if left is less than. Evaluates to true if it is; false otherwise.

=>

Compares left value to right value to see if left is greater than or equal to. Evaluates to true if it is; false otherwise.

<=

Compares left value to right value to see if left is less than or equal to. Evaluates to true if it is; false otherwise.

3. Assignment Operators

Alacs will use the standard assignment operator to assign the value of the right of the operand into the variable to the left of the operand. The left operand cannot be a literal value. The left and right operands must be the same type.

Operator	Type	Associativity	Example
=	Assignment	Right to Left	num x = 3

```
2 = 3  /* will not be accepted*/
String name = Alacs    /* will be accepted*/
```

4. Logical Operator

Logical operators similar to Java are operators that will return a Boolean result, or in respect to our language a bool result. The result is based on the bool result of one or two other expressions. The “and” operator returns true if both of the expressions on the left and right hand side of the “and” evaluates to true. Both expressions are evaluated before the “and” operator is applied. The “or” operator returns true if at at least one of the operands evaluates to true. Similar to the “and” operator both will expressions will be evaluated before the “or” operator is applied.

Operator	Type	Associativity	Example
and	Logical AND	Left to Right	(true and false) returns false

Operator	Type	Associativity	Example
or	Logical OR	Left to Right	(true or false) returns true

IV. Syntax

Syntax in ALACS will be similar to Java. Each block of code will begin with an open bracket and end with a close bracket. Each individual line of code will end in a semi-colon. You will declare variables by first stating the type then the variable name followed by an equals sign and what the value of the variable will be. For example, {num x = 6;}.

V. Functions

A function in Alacs will be a collection of statements grouped together to perform a certain operation. A function must be declared before it can be used. The required elements of a method declaration:

1. A return type
    a. the return type portrays the type of object the function will be returning. A function can return void meaning it returns nothing
2. A modifier, public or private
    a. a public function is globally accessible while a private the method is only accessible within its declared class
3. A name
    a. the function name must be a valid sequence of ASCII characters
4. A pair of parentheses () after the name, which can hold the parameters if needed
    a. The list of parameters in the parentheses will be a comma separated list of input parameters. It is optional to have parameters but the parentheses must always appear after the function name, even if they are empty
5. Curly braces after the function declaration which will hold the statements inside. {}
    a. functions must be followed by one opening bracket, then the body of the function should start. Once the body of the function is complete it should be followed by a closing bracket.
6. Uniqueness
    a. every function should have a unique combination of a name and parameters.

Examples:

```
public num add(num a, num b) {
        return a +b;
}
public void create(){
        String alacs = “ALACS”;
}
```

The print() function will be built in and can be used as seen below.

```
num x = 5;
String s = “hi”;
print(x); /*prints 5*/
print(s);/*prints hi*/
```

VI. Variables

A variable in Alacs will provide one with named storage that the program can manipulate. Each variable in alacs will have a specific type, which will determine the size of the variable’s memory. All variables must be declared before they are used.

The declaration of variables will follow this format:

data type variable = value;

1. The possible “data types” in alacs are nums, bools, char, or strings
2. Variable names will comprise of a sequence of ASCII characters
3. The value given must match the data type

VII. Conditional statements and Looping

If-Else:

The if state executes a particular section of code if the bool expression given is true. An if statement can be followed by an optional else statement. The else statement executes when the bool expression is false.

```
num x = 1
if(x == 3) {
        x = x-1         /* will run if bool expression is true
} else {
        x = x +1        /* will run if bool expression if false
}
```

While:

The while statement will continuously run a section of code until a specific condition/expression becomes false. Its syntax is as follows:

```
while(expression) {
        statement
 }
```

The expression above must return a boolean value. If the expression never becomes false the while statement will keep executing the statement and or statements thus becoming stuck in an infinite while loop.

An example of looping through numbers 1 through 100 using a while statement:

```
num track = 1;
while (track < 100) {
        track = track +1
}
```

For:

The for statement repeatedly loops over a code block until a particular condition is satisfied. Its syntax is as follows:

```
for (initialization; termination; increment)
        { statement(s) }
```

Definitions and Rules for the for loop is as follows:

1. The initialization expression initializes the loop. It will be executed once when the loop begins

2. The termination expression must evaluate to a bool. When the termination expression evaluates to false, the loop will terminate.
3. The increment expression is invoked after each iteration through the loop.

An example of looping through numbers 1-100 and incrementing count through each loop iteration:

```
 num count = 0;
 for (num track = 1; track < 101; track=track+1) {
        count = count +1;
}
```