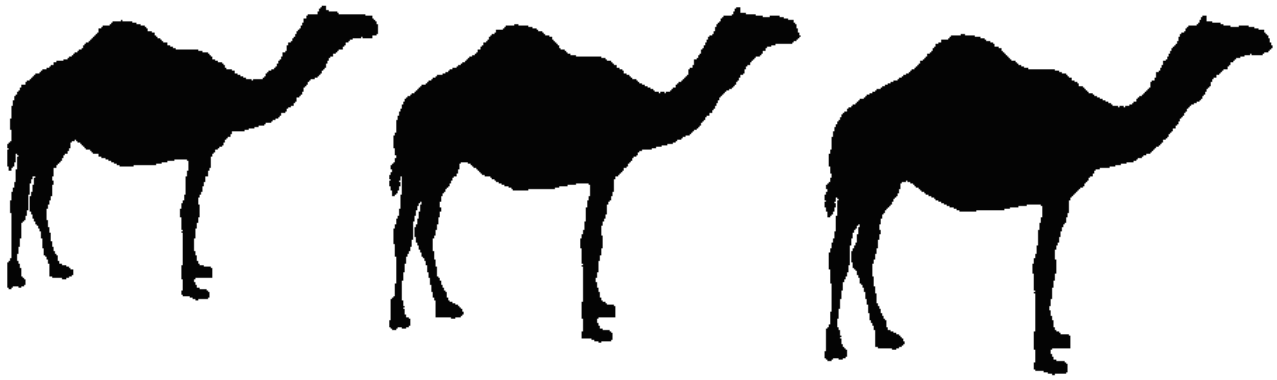
[K]AML



Array Manipulation Language
Reference Manual

***K**aori Fukuoka*	*kf2284@columbia.edu*
***A**nkush Goel*	*ag2812@columbia.edu*
***M**aninder Singh*	*ms3770@columbia.edu*
*Mayur **L**odha*	*mdl2130@columbia.edu*

COMS 4115: Programming Languages and Translators, Fall 2008: Prof. Stephen Edwards.

TABLE OF CONTENTS

1. Introduction

[K]AML is an expressive and concise Array Manipulation language which features rich set of operations on Arrays. Unlike other structured programming languages, **[K]AML** treats the array as a primitive data type and array manipulations is done using high level operators. Thus, it would not require a beginner to think of arrays as a collection of data but as a single data-type in itself. Using **[K]AML**, it becomes possible to express a computable function using just an expression in a single line of code. This reduces the potential number of loops and allows for concise and compact programs.

In Computer Science, most of the programming languages have concept of Arrays as one of the fundamental data structures which consists of ordered, integer indexed collection of objects. Array manipulations form an important as well as error prone part of many algorithms. For functionalities which have support for arrays, it becomes important and necessary to have a language which is handy and allows one to write more effective code. So far, no language has direct support for array manipulations as they have for integer or other primitive data type. We need a language using which array manipulating operations like adding all elements, returning a sub array satisfying certain conditions, array concatenation, copying arrays, set operations on arrays can be performed effectively.

2. Lexical Conventions

2.1 Identifiers

Identifiers consist of a sequence of letters and numbers. Letters contain lowercase and uppercase alphabets from the ASCII set. The first character of an identifier must start with a letter. Symbols are not allowed for identifiers.

2.2 White Spaces

White spaces are ignored during the compilation. The following characters are defined as white spaces: space, newline, tabs, and comments.

2.3 Comments

Comments are defined by the two slashes without blanks: // and are single-line only. Comments are ignored during compilation.

2.4 End of Statement

A single semicolon character (;) indicates that it is the end of each statement.

2.5 Constants

Only number is the type of constants that is defined in [K]AML language.

2.6 Keywords

The following identifiers are the reserved keywords and cannot be used otherwise:

show **get** **for** **function** **return**

The following chart is a list of all character sequences that are also keywords. Each pattern is listed with its corresponding token name.

Pattern	Token name	Pattern	Token name
;	SEMICOLON	<-	INSERT
[	L_BRACKET	->	DELETE
]	R_BRACKET	?	IF
{	L_BRACE	!	ELSE
}	R_BRACE	“	QUOTE
(	L_PAREN	>	GREATER_THAN
)	R_PAREN	<	LESS_THAN
=	ASSIGN	>=	GREATER_THAN_EQUAL
+	PLUS	<=	LESS_THAN_EQUAL
-	MINUS	==	EQUAL
*	TIMES	!=	NOT_EQUAL
++	INCREMENT	&&	AND
--	DECREMENT	||	OR
+=	PLUSASSIGN	%-	DIFFERENCE
-=	MINUSASSIGN	%+	UNION
*=	TIMESASSIGN	%=	INTERSECT
#	HASH	>>	MAX
..	RANGE	<<	MIN
,	COMMA	<>	AVG

3. Types

Identifier can be defined in [K]AML as:

3.1 Number

The basic type in [K]AML is a number which consists of one or more digits in sequence.

3.2 Arrays

An array contains a sequence of elements of the same type. In [K]AML, all arrays can contain only numbers.

3.2.1 Declaration

Declaration of an array is specified with square bracket followed by its name. For multi-dimensional array, the constant expression that specify the bounds of arrays can be missing only for the first member of sequence. Eg: a[], b[][5]

3.2.2 Initialization

Array can be initialized as:

[]a = {1,2,3,4,5};

[][4]b = {{1,2,3,4},{,1,2,4,5}};

3.2.3 Element

Each element of an array can be denoted as [i]a where 'a' is the identifier name of the array and 'i' is the index number which indicates the element position at a distance from the beginning of the array. For example, the first element of an array can be denoted as [0]a.

3.2.4 Length of an array

Length of an array is the number of elements in an array. Length of an array can be defined with a palm sign followed by the identifier name: #a specifies the number of elements in an array.

3.2.5 Multiple elements

Multiple elements of an array can be retrieved by using comma (,) and range (..) notations.

For example :-

[1, 4, 6]a; represents array with a sequence of elements [1]a, [4]a, and [6]a.

[2..4, 7]a; represents array with a sequence of elements [2]a,[3]a,[4]a,[7]a.

[2..7][4]b; represents elements of rows 2 to 7 in column 4.

4. L-values

L-values are the variables which values can be assigned with corresponding addresses of memory location. Any identifiers of basic type and array names can be modifiable L-value, which assigned object can be changed and examined.

5. Scope and Lifetime

The variable declared in the code is extended throughout the file. If re-declaring of the same variable name occurs, the most current value will be extended.

6. Expressions

Expressions are executed in a sequential order. They fall in following groups:

Expression → NumberExpression

| ArrayFunctionExpression

| AssignmentExpression

| ArithmeticExpression

| ConditionalExpression

6.1 Number Expressions

It specifies the numeric constant. It has the form

NumberExpression → Number

6.2 Array Function Expressions

These expressions are basically used to perform initialization and manipulation actions on arrays. They have the following form:

ArrayFunctionExpression → ArrayExpression

| ArrayInitialiseExpression

| ConcatenateExpression

| InsertExpression

| DeleteExpression

| SetExpression

| SumArrayExpression

| ReverseExpression

| ArraySizeExpression

They have the form:

ArrayExpression → (LEFTSQUAREBRACKET (ListExpression)?

RIGHTSQUAREBRACKET)+ Identifier

ArrayInitialiseExpression → LEFTCURLBRACKET ElementList

RIGHTCURLBRACKET

ConcatenateExpression → ArrayExpression (PLUS ArrayExpression)+

InsertExpression → ArrayExpression LEFTARROW InitialiseExpression

DeleteExpression → ArrayExpression RIGHTARROW Number

SetExpression → (SETUNION | SETDIFFERENCE | SETINTERSECTION)?

ParameterizedExpression

SumArrayExpression → PLUS ArrayExpression

ReverseExpression → MINUS ArrayExpression

ArraySizeExpression → HASH ArrayExpression

6.3 Assignment Expressions

It is used to assign the value of second Expression to the first Expression. It has the form:

AssignmentExpression → Expression Operators Expression

6.4 Arithmetic Expressions

It takes first expression and second expression as the operands and evaluates them using the operator defined. They have the form:

ArithmeticExpression → Expression (ArithmeticOperators Expression)+

The ArithmeticOperators includes the basic arithmetic operators like +, -, *.

6.5 ConditionalExpression

It has the form:

ConditionalExpression → Expression (ConditionalOperator Expression)*

The ConditionalOperator includes the basic conditional operators && and ||.

7. STATEMENTS

Unless explicitly specified, Statements are executed in sequence. Each statement is terminated with a semicolon(;). Statements fall into following groups:

7.1 Conditional Statements

Conditional Statements is specified with CONDITION (?) followed by optional (!) Expression.

? (Expression) {Statement} **!**{Statement}

If the Expression evaluates to a non zero value than first statement is executed else statements within ! is executed. Each ! matches to its nearest CONDITION (?).

7.2 For Statements

It functions same as in other programming languages like C. It has the form:

for (Expression ; Expression) { Statement }

The first expression specifies the initialization and the condition for continuation of the loop while the second expression specifies an action which is executed for each iteration of the loop.

7.3 Return Statements

Return statement is used to return some value from the function. It is the onlyjump statement in the language. It has the form:

return Expression;

The value of expression is evaluated and returned from the function call.

7.4 Function Definitions

Language provides support for user defined functions which has the form:

function Expression {Statements}

Expression specifies the name of the function with the parameterized list while statements specifies the function body.

7.5 Function Calls

Functions can be called using the following form:

Identifier (Expression);

Identifier specifies the name of the function to be called while Expression contains the parameter list to be passed to the function.

7.6 Input/Output Statements

These statements are of the form:

get (Expression)

Get statement is used to read data from stdin.

show (String)

Show statement is used to write to the stdout.

8. Example

Here is the example of programs based on [K]AML syntax.

Consider a 2 dimensional array ‘a’ which stores the marks secured by various students in different subjects. The rows represent the students and the columns represent the subjects. The following program shows various operations that can be performed on this array.

```
for (j=0..#a[ ])
{
        show(“Maximum marks of any student in this subject is:” >>a[ ][j]);
        show(“Minimum marks of any student in this subject is:” <<a[ ][j]);
        show(“Average marks of the class in this subject is” < >a[ ][j]);
        [ ]c1=?(a[ ][j],<40);
        [ ]c2=?(a[ ][j],>=60 && <80 );
        [ ]c3=?(a[ ][j],>80 && <90);
        [ ]c4=?(a[ ][j],=100 );
        show(“Number of students getting F in this course are” #c1);
        show(“Number of students getting C in this course are” #c2);
        show(“Number of students getting B+ in this course are” #c3);
        show(“Number of students getting A+ in this course are” #c4);
}
```

Appendix - A

Operator	Name
;	end of statement
[ ]	array
{ }	brackets
( )	parenthesis
=	assignment
+	summation
-	elimination
*	multiplication
+=	increment
#	length of array
..	range
,	separator
“	string
<-	insert
->	delete
?	conditional if
!	conditional else
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
==	equal
!=	not equal
&&	and
||	or
%-	set difference
%+	set union
%=	set intersection
>>	max
<<	min
<>	average

Appendix – B

Grammar

CompilationUnit → (Statement)*

Statement → ConditionalStatement

| ForStatement

| ReturnStatement

| FunctionDefinition

| FunctionCall

| InputStatement

| OutputStatement

| Expression

ConditionlStatement → CONDITION '(' Expression ')' '{' Statement '}'

('**!** '{' Statement '}')?

ForStatement → FOR (Expression ';' Expression) '{' Statement '}'

ReturnStatement → RETURN Expression

FunctionDefinition → FUNCTION Expression '{'Statements'}'

FunctionCall → Identifier Expression

InputStatement → GET '(' Expression ')'

OutputStatement → SHOW '(' String ')'

String → ' " ' Any combination of ASCII characters ' " '

Expression → NumberExpression

| ArrayFunctionExpression

| AssignmentExpression

| ArithmeticExpression

| ConditionalExpression

NumberExpression → Identifier

| Number

Identifier -> [‘a’..’z’ ‘A’..’Z’] [‘a’..’z’ ‘A’..’Z’ ‘0’..’9’]*

Number → (Digit)+

ArrayFunctionExpression → ArrayExpression

| ArrayInitialiseExpression

| ConcatenateExpression

| InsertExpression

| DeleteExpression

| SetExpression

| SumArrayExpression

| ReverseExpression

| ArraySizeExpression

ArrayExpression → LEFTSQUAREBRACKET (ListExpression)?

RIGHTSQUAREBRACKET Identifier

ListExpression → IncludeExpression | EliminateExpression

IncludeExpression → (ElementList IntervalList)*

ElementList → (Element (COMMA)?)*

Element → Number

IntervalList → (Interval (COMMA)?)*

Interval → Number RANGE Number

EliminateExpression → (EliminateElementList EliminateIntervalList)*

EliminateElementList → (MINUS Element (COMMA)?)*

EliminateIntervalList → (MINUS Interval (COMMA)?)*

ArrayInitialiseExpression → LEFTCURLBRACKET ElementList

RIGHTCURLBRACKET

ConcatenateExpression → ArrayExpression (PLUS ArrayExpression)+

InsertExpression → ArrayExpression LEFTARROW InitialiseExpression

DeleteExpression → ArrayExpression RIGHTARROW Number

SetExpression → (SETUNION | SETDIFFERENCE | SETINTERSECTION)?

ParameterizedExpression

ParameterizedExpression → LEFTROUNDBRACKET ParameterList

RIGHTROUNDBRACKET

ParameterList → (Expression (COMMA Expression)*)

SumArrayExpression → PLUS ArrayExpression

ReverseExpression → MINUS ArrayExpression

ArraySizeExpression → HASH ArrayExpression

AssignmentExpression → Expression Operators Expression

Operators → EQUAL | ArithmeticOperators

ArithmeticOperators → PLUS | MINUS | ASSIGNSUM | ASSIGNDIFFERENCE

ArithmeticExpression → Expression (ArithmeticOperators Expression)+

ConditionalExpression → Expression (ConditionalOperator Expression)+

ConditionalOperator → ‘<’ | ‘>’ | ‘<=’ | ‘>=’ | ‘==’ | ‘!=’ | ‘&&’ | ‘||’