CHAD Language Reference Manual

INTRODUCTION

The CHAD programming language is a limited purpose programming language designed to allow teachers and students to quickly code algorithms involving arrays, queues and stacks and see graphically how their data is being manipulated. CHAD is an interpreted language in which the interpreter outputs a graphical representation of the data structures used in the program as well as performing any calculations. The goal of CHAD is to make visualization of algorithms easier by providing a way to easily and compactly create graphical representations of them.

STRUCTURE OF THE PROGRAM

A CHAD language program consists of a series of variable declarations followed by a series of statements to be executed followed by a series of optional user-defined functions. User-defined functions consist of the function keyword followed by the return type of the function (array, int, queue, stack, string or void for no return value) followed by the name of the function. Next comes a list of optional comma separated arguments of the form type name, enclosed in parentheses (which are not optional) and a list of variable declarations followed by a list of statements, and are ended with the endfunction keyword. The last thing before an endfunction keyword must be a return statement, which can only appear there. Note all declarations must be made before any statements in the block that contains them.

VARIABLES AND DECLARATION

The CHAD language has 5 native data-types. They are integers, strings, arrays, queues and stacks. All variables must be declared before they are used and all declarations are to be located at the beginning of a program or function definition. All variables are local in scope. Variable defined within a function can be used only within that function and variable defined in the main body of the program (outside of any function declaration) cannot be used by any functions in the program. Any parameters needed by a function must be explicitly passed to that function and any variables modified by a function must be returned in order to be accessed from elsewhere in the program. The only exception to this is some built in functions which can be called on a variable using a variable.function() syntax which gets the variable as a reference and is able to modify the variable it is given as well as return another value. This will be discussed further when we discuss function calls. When variables are declared they can be declared with or without initializing them. A variable cannot actually be used before it is initialized, however. The syntax for initialization of a variable is exactly the same as the assignment syntax. Variable and function names are identifiers (see later definition of an identifier).

Integers

The integer data type in CHAD supports integers values from –999 to 999 the declaration of an integer variable consists of the keyword int followed by the name of the variable. A value may be assigned using standard assignment rules (as described below), and the declaration must terminate in a semicolon.

Strings

The string data type in CHAD supports alphanumeric strings up to 30 characters in length. For a list of characters to be interpreted as a string literal by CHAD it must be enclosed in double quotes and can include any characters that are not double quotes or newline characters. The declaration of a string variable consists of the keyword string followed by the name of the variable. Again, it may be assigned immediately, and the declaration must terminate in a semicolon.

Arrays

The array data type in CHAD supports arrays of either string or integer types with a fixed length chosen at declaration. The declaration of an array consists of a statement such as array[int, 10] where the first argument is the type of the values to be stored in the array and the second argument is the length of the array. The index of the array begins at 0 and goes to n-1 where n is the length of the array. The length of an array is fixed as soon as it is created. Trying to access an element of the array that does not exist causes an error by the compiler.

Queues

The queue data type supports queues of either string or integer types. Queues are dynamically assigned memory by the interpreter and therefore have no fixed length. However the programmer should use caution to avoid overflowing available memory.
The declaration of a queue consists of the queue keyword followed by the type of the queue in square brackets followed by the name of the queue and then a semicolon. e.g. queue[int] myQueue;

Stacks

The stack data type supports support stacks of either string or integer types. Stacks are dynamically assigned memory as queues are and therefore have no fixed size. The declaration of a stack consists of the stack keyword followed by the type of the stack in square brackets and the name of the stack. e.g. stack[int] myStack;

BUILT IN OPERATORS

Note that a single variable or a literal value are each considered an expression of the same type as the variable or the literal.

ASSIGNMENT (=)

The assignment operator,. = is used to give a particular value to a variable after that variable has been declared. The syntax is variable = value where value can be any expression that evaluates to the same type as the variable it is being assigned to, or any single literal value of the appropriate type. There are no conversion operators in CHAD, assignments must be of the same type of variable or value.

Integers and Strings

These use only basic assignments as shown above. The value of an integer must be a decimal number between -999 and 999. The value of a string may be any sequence of characters except for double quotes and newlines, enclosed in double quotes.

Arrays.

There are two types of assignments to arrays. The first treats each cell in the array individually. The syntax to extract a single cell from an array is the name of the array followed by the index of the desired value in square brackets e.g. myArray[5] accesses the 6th element of myArray. This syntax is treated as an integer or string depending on the type of the array. If an attempt is made to access an index which is beyond the length of the array an error will be raised. The second method of assignment to an array handles multiple values at the same time. This syntax is variable = {vals} where vals is a comma separated list of n values to be put into the first n cells of the array. All members of vals must be of the appropriate type for the array they are being assigned to. Any values currently in these cells will be discarded in the process of the assignment. If you attempt to assign more values than there are cells in the array an error will be raised. An array can also be assigned to another array of the same type. However if an array is assigned to an array which is shorter in length it will raise an error.

Queues and Stacks

There is no assignment syntax for queues and stacks. Values are added and subtracted through special access methods which will be defined later.

ADDITION (+)

The addition operator can be applied to integers and strings with the syntax value + value., where values can be either literals or expressions. The results can then be assigned to a variable as defined above. The two values must be of the same type and if they are assigned to a variable that variable must also be of the same type. If integers are added, they are arithmetically added. If two strings are added they are concatenated.

SUBTRACTION (-)

Is the usual arithmetic operation and can only be applied to integer expressions. Can also be applied to a single integer value in prefix notation e.g. -8 in which case it changes the sign of the integer to which it is applied.

DIVISION (/)

Is the usual arithmetic operation and can only be applied to integer expressions.

MULTIPLICATION (*)

Is the usual arithmetic operation and can only be applied to integer expressions.

INCREMENT (++)

Can be applied in postfix notation to an integer variable to increase the value by 1.

DECREMENT (–)

Can be applied in postfix notation to an integer variable to decrease the value by 1.

PARENTHESES (())

Expressions that are placed within parentheses should be evaluated before any outside operations are applied to them.

The precedence of the above operators is negation and increment and decrement, multiplication and division, addition and subtraction, and then assignment. If another order of operations is desired, parentheses should be used to specify it. Within any level of precedence operators are applied from left to right.

COMPARISON OPERATORS

Is Equal to (==)

e.g variable == value compares the values on the two sides of the operator. Returns 0 if the variable are not equal and 1 if the variables are equal. Can only be used on integer and string expressions. If the types of the two values are different the function will always return 0. For integers the variables are considered equal if they have the same numerical value. In the case of strings, the variables are considered equal if they consist of the same sequence of characters.

Is Greater Than (>)

Can be used on either integer or string expressions but the behavior is undefined if applied to unmatched types. The syntax is ex1 > ex2 and compares ex1 and ex2. Returns 1 if ex1 is larger and 0 otherwise. Integers are compared using their numeric values. Strings are compared lexically according to the ASCII values.

Is Less Than (<)

Behaves exactly like greater than except return the opposite value in all cases except where the values are equal.

Greater Than or Equal To (>=)

Behaves like greater than but returns 1 in the case that the two values are equal.

Less Than or Equal To (<=)

Behaves like less than but returns 1 in the case that the two values are equal.

LOGICAL CONJUNCTION OPERATORS
AND

The keyword AND is used to signify the logical conjunction of two expressions. Returns 1 if the value of both expressions are 1 and 0 otherwise.

OR

The keyword OR is used to signify the logical disjunction of two expressions. Returns 0 if the value of both expressions are 0 and 1 otherwise.

NOT

The keyword NOT is used to signify the logical negation of an expression. Returns1 if the value of the expression is 0 and 0 if the value of the expression is 1.

In general all expressions in a complex logical expression are evaluated regardless of whether the outcome could be known at an earlier point. The order of operations for logical expressions is the comparison operators evaluated from left to right followed by the conjunction operators evaluated from left to right. Parenthesis should be used to enforce any other desired precedence relationships.

CONTROL STATEMENTS

for loops

The for loop syntax allows a block of code to be run multiple times. The syntax for the for loop is

```
for(var; test; change;)
statements
endfor
```

where end and endfor are keywords, var represents an assignment to the loop control variable, test represents the test expression which determines whether or not the body of the loop will be executed. If the test evaluates to true the body is executed otherwise it is skipped. Change represents an assignment which changes the value of the control variable each time the loop is executed. The endfor keyword indicates the end of the for loop body. When the test evaluates to zero execution of the program picks up at the statement following the endfor keyword.

IF-THEN-ELSE

The IF-THEN-ELSE syntax allows statements to be executed or not depending on upon the value of a test expression. The syntax is:

```
if(test)
statements
```

```
endif
elseif(test)
statements
endelseif
else
statements
endelse
```

if, endif, else, endelse are keywords of the language. Test represents a conditional statement. If test evaluates to true the statements between if and endif are executed, otherwise the statements between else and endelse are executed. After these statements are executed the program continues at the statement following the endelse keyword. The block

```
else
statements
endelse
```

is optional. If it is left off and the test evaluates to false the program begins execution at the statement following the endif. In addition, the block

```
elseif(test)
statements
endelseif
```

is both optional and repeatable: it can be left out entirely, or there can be multiple elseif blocks, each of which has a different test expression.

Control statements can be arbitrarily nested within each other. If this is the case the endfor, endif and endelse keywords match the closest for, if or else respectively.

BUILT IN FUNCTIONS

There are two kinds of built in functions for CHAD. The first are standard function which take arguments and return based on them. The second are specific to a data type and are called on a particular variable by using the syntax of variable.function(args) where function is the name of the function and args is a comma separated list of arguments which are passed to the function. The built in functions are defined to work on specific data types.

min

min(var1, var2) returns the argument with the least value. Min can be called with either integer or string expressions. It causes an error if called with one of each. The comparisons are done as if the < operator were called.

max

max(var1, var2) returns the argument with the greater value. Max can be called with either integer or string expressions but causes an error if called with one of each. The comparisons are done as if the > operator were called.

show

Takes as an argument any variable and makes that variable part of the display. By default, variables are not included in the display.

hide

Takes as an argument any variable already part of the display and removes it from the display. By default, variables are not included in the display.

print

Takes as an argument a string variable or literal and adds it to the comments portion of the display. Allows an instructor to make notes about an algorithm while it is executing.

Arrays

myArray.swap(index1, index2); swaps the values in the locations indicated by index1 and index2 in the array indicated by the identifier before the ‘,’, Has no return value.

myArray.sortAZ(index1, index2); puts the values in the locations indicated by index1 and index2 in the array indicated by the identifier before the ‘.’ in increasing order in the locations indicated. Returns 1 if a switch was made and 0 otherwise.

myArray.sortZA(index2, index1); puts the values in the locations indicated by index1 and index2 in the array indicated by the identifier before the ‘.’ in decreasing order in the locations indicated. Returns 1 if a switch was made and 0 otherwise.

Stacks

myStack.pop(); removes the top value from the given stack and returns it. The return type is the same as the type of the that the stack was declared to store.

myStack.push(arg); adds the value represented by arg to the given stack. Returns 1 if successful, 0 otherwise. Arg must be of the same type as the stack was declared to store.

Queues

myQueue.enqueue(arg); adds the value given by arg to the given queue. Returns 1 if successful, 0 otherwise. Arg must be of the same type as the queue was declared to store.

myQueue.dequeue(); takes the value from the front of the queue, removes it and returns it. The return type is the same as the type of the queue was declared to store.

DEFINING YOUR OWN FUNCTIONS

User defined functions can be added to the end of a program. The structure of a user defined function is:

```
function returnType myFunction(args)
statements
return val;
endfunction
```

where function and endfunction are keywords, returnType is the type that the function returns myFunction represents the name of the function and args represents a comma separated list of arguments each preceded by their types. Return is also a keyword which must be the last statement in the functiond definition and val represents the value to be returned by the function. The return statement must be the last statement in the function definition and is required. If the return type of the function is void no value follows the return statement.

e.g.

```
function int min(int x, int y)
int temp;
if(x < y)
temp = x;
endif
else
temp = y;
endelse
return temp;
endfunction
```

Assignments and function calls must be ended with semi-colons. For, if and else statements must be ended with their appropriate endfor, endif or endelse. Functions can be defined to not have a return value. In this case, returnType in the declaration is "void", and the return statement is merely "return;".