User's Manual to the Package krclater18, a Compiler and Executor for the transy Language

By Kelsey Clater

transy Language

- Types of identifiers
    - Simple variable
        - All variables in transy are of type double
        - All uninitialized variables have the value .0123456789
            - See z flag
        - Naming convention:
            - Must begin with an alpha character and consists of alpha characters, digits, and underscore characters
                - Example of valid variable names:
                    - a
                    - b789
                    - c_99
            - Can be no longer than 128 characters
    - Aggregate variable
        - Array of type double
        - Naming conventions follow the variable naming conventions
        - See dim command
        - See r flag
    - Literal variable
        - variable that stores a literal string
        - Naming convention:
            - Must begin with '$' character, and consist of alpha characters, digits, and underscore characters
            - Can be no longer than 128 characters
        - See lread and lwrite commands

- Commands:
    - dim
        - dim is used to create an array
        - ****All dim commands must come before all other commands****
        - example :
            - dim a[10]
        - The first identifier is the name of the array and must be a variable
        - The second identifier is the size of the array and must be a positive, non-zero integer
        - Up to 7 arrays can be declared with one dim command, each separated by a comma

            - Example of multiple array dim statement:
                - Dim alpha[10], beta[5], sigma[4], pi[9]
- read
    - read is used to give a simple variable a value from the user
    - Example:
        - Read hello
    - Up to seven variables can be declared at once using a read command and should be separated by commas
- Write
    - Write is used to write the value of a simple variable to the screen
    - Example:
        - Write up7
    - Up to seven simple variables can be written to the screen with a single write command and should be separated by commas
- Stop
    - Stop is used to terminate a program
    - It can placed arbitrarily throughout a program
- Listo
    - Listo prints to the entire contents of the .obj file to the screen
- Subp
    - Subp is used to compute mathematical operations
    - Format:
        - Subp sin(x,y)
        - This will compute x = sin(y)
        - The math function code follows the command. The first identifier must be a variable and the second can be a constant or a variable
    - Subp is can be used for to perform the following mathematical functions:
        - Sine, cosine, exp, absolute value, log base 2, log base e, log base 10, and square root
        - Each computation has a code:
            - Sine = sin; cosine = cos; exp = exp; absolute value = abs; log base 2 = alg; log base e = aln; log base 10 = log; square root = sqr
- Goto
    - Goto is used to jump to the line of code where the specified line label is found
    - Example:
        - Goto linelabel3
        - The identifier must be a valid line label. In other words, the line label must exist
- Nop
    - Nop does nothing
- Ifa
    - Ifa is an arithmetic conditional
    - Example

- ifa(mouse) dog, cat, bunny
- The first identifier is a variable or constant and the next three are line labels
- If the identifiers value is less than 0, the program will go to the line where dog is. If the identifier is equal to 0, then the program will go to the line where cat is. If the identifier is greater than 0, the program will go to the line where bunny is

- Lread
  - lread is used to fill a literal value with input from the user
  - Format
    - lread $graphite
    - The identifier must be literal variable
    - Only one identifier per command
- Lwrite
  - Lwrite is used to print a literal string to the screen
  - Format
    - lwrite $helloWorld
    - lwrite “hello world”
    - lwrite can either take a literal variable or a literal string
    - Only one identifier per command
- Cdump
  - Cdump prints the contents of core memory to the screen
  - Format
    - Cdump 0,100
    - The first identifier is the index at which the user would like to start printing values of core, in this case, 0. The second is the stopping index, in this example, 100
- Loop
  - Loop is used for pre-test iteration
  - Format
    - Loop i = 0,10,1
    - The first identifier must be a variable, the second, third, and fourth can be variables or integer constants or double constants
    - The first variable serves as the runner of the loop
    - The second is the start value
    - The third is the stop value
    - The fourth is the runner increment value
      - If increment is zero, the loop with terminate when the runner equals the stop value (third identifier)
    - The fourth identifier is optional. If none is specified, the value is assumed to be 1
- loop-end

        - loop-end is used to signal the end of a block of statements for the loop command
        - Example:

```
loop i = 0,10
        write i
loop-end
```

- aread
    - aread is used to fill an array with values from the user
    - Format
        - aread green,2,6
            - The first identifier must be a declared array
            - The second identifier can be a variable or a positive integer and is the start index at which the user will begin inserting values
            - The third identifier can be a variable or a positive integer and is the stop index at which the user will insert the final value
- awrite
    - awrite is used to print the values of an array to the screen
    - Format
        - awrite blue, 9,15
            - The first identifier must be a declared array
            - The second identifier can be a variable or a positive integer and is the start index at which the command will begin printing values
            - The third identifier can be a variable or a positive integer and is the stop index at which the command will print the final value
- If
    - If is a condition
    - Format
        - if(alpha > omega) then linelable99
            - The first and second identifiers can be variables or integer constants or float constants separated by a condition
            - Conditions include:
                - <, less than
                - >, greater than
                - =, equal to
                - >=, greater than or equal to
                - <= less than or equal to
                - != not equal to
            - The third identifier must be a line label
            - If the condition is true, the program jumps to the line on which the line label corresponds to
- Cls

        - Cls is used to clear the command window screen

How to use the compiler:

- Compiling a program
    - Typing "./compiler" will automatically attempt to compile the file **test1.transy** found in the **same** directory as the compiler
    - Typing "./compiler <fileName>.transy will compile the specified file found in the same directory as the compiler
    - If the user would like to compile a file found in a different directory, simply **add the path**
        - *Example*: "./compiler ~/Documents/test1.transy"
    - Additionally, appending the **file type is optional**
        - *Note*: "./compiler test2" and "./compiler test2.transy" will compile the same file
- Compiler Flags
    - Upon successful compilation 3 files of the same name, but different type will be produced: a *".obj"*, ".core" and ".literal" file
    - Upon a failed compilation 3 files of the same name, but different type will be produced: a *".noblanks"*, ".core", and ".literal" file
        - Note that there is a difference between the files produced upon successful compilation and not successful compilation
    - **o Flag**: If the user would like to keep the *".obj"* file upon an unsuccessful compilation, use the **x flag**
        - *Example*: "./compiler -o test3.trasy"
    - **n Flag**: If the user would like to keep the ".noblanks" files upon successful compilation, use the **x flag**
        - *Example*: "./compiler -n test4.transy"
    - **x Flag**: If the user would like to execute their program upon successful compilation, use the **x flag**
        - *Note*: this flag will only work if the program compiled successfully
    - For more information on the .noblanks and .obj files, please see the **Programmers Guide**

How to use the executor:

- Executing a program
    - The guidelines for compiling and executing are very similar
    - Typing "./executor" will automatically attempt to compiler the file **test1.obj** found in the **same** directory as the executor
    - Typing "./executor <fileName>.obj will compile the specified file found in the same directory as the executor
    - If the user would like to execute a file found in a different directory, simply **add the path**
    - Additionally, appending the **file type is optional**
- Executor Flags

- **r Flag**: The r flag turns on **range checking**. If the user attempts to read or write to an array outside of the defined boundaries, then the executor will produce an error:
- **z Flag**: The z flag will set all **undefined variables to zero**, otherwise, undefined variables have the value .0123456789

Compilation Error Messages:

- General Notes:
  - Aside from flag and file errors, compilation errors should be printed with the corresponding line on which they were found before the error
  - If an error is not listed under a specific command, it can be found under general errors

Flag Errors

- ERROR: UNRECOGNIZED FLAG

File Errors

- ERROR: UNABLE TO REMOVE THE ".obj" OUTPUT FILE
- ERROR: UNABLE TO REMOVE THE ".noblanks" OUTPUT FILE
- ERROR: INVALID INPUT FILE TYPE
- ERROR: UNABLE TO OPEN THE FILE: <filename>!
- ERROR: UNABLE TO OPEN " .noblanks" FILE!
- ERROR: UNABLE TO OPEN ".obj" FILE!
- ERROR: UNABLE TO OPEN ".literal" FILE!
- ERROR: UNABLE TO OPEN ".core" FILE!

General Errors

- ERROR: UNIDENTIFIED COMMAND
- ERROR: MISSING IDENTIFIER
- ERROR: INVALID IDENTIFIER
- ERROR: TOO MANY IDENTIFIERS
- ERROR: INVALID SYNTAX
- ERROR: UNIDENTIFIED IDENTIFIER

dim Errors

- ERROR: DIM ERROR
- ERROR: INVALID ARRAY
- ERROR: CANNOT DECLARE AN ARRAY LARGER THAN THE MAXIMUM CORE SIZE

read/write Errors

- ERROR: A MAXIMUM OF 7 VARIABLES CAN BE READ OR WRITTEN AT ONE TIME

stop Errors

- ERROR: UNRECOGNIZED STOP COMMAND

listo Errors

- ERROR: UNRECOGNIZED LISTO COMMAND

nop Errors

- ERROR: UNRECOGNIZED NOP COMMAND

goto Errors

- ERROR: UNIDENTIFIED LINE LABEL

aread/awrite Errors

- ERROR: THE VARIABLE, <varName>, MUST BE DEFINED BEFORE AREAD/AWRITE
- ERROR: AREAD/AWRITE MUST CONTAIN 3 IDENTIFIERS

loop-end Errors

- ERROR: UNRECOGNIZED LOOP-END COMMAND

lwrite Errors

- ERROR: CANNOT LWRITE AN UNDEFINED LITERAL VARIABLE

cls Errors

- ERROR: UNRECOGNIZED CLS COMMAND

Assignment Expression Errors

- ERROR: INVALID ASSIGNMENT STATEMENT
- ERROR: THE FIRST ID MUST BE A VARIABLE
- ERROR: INVALID SYMBOL FOLLOWING AN ID ON THE LEFT HAND SIDE OF ASSIGNMENT
- ERROR: INVALID SYMBOL FOLLOWING AN ID
- ERROR: INVALID SYMBOL FOLLOWING AN EQUAL
- ERROR: INVALID SYMBOL FOLLOWING A OPEN PARENTHESES
- ERROR: INVALID SYMBOL FOLLOWING A CLOSE PARENTHESES
- ERROR: INVALID SYMBOL FOLLOWING AN OPEN BRACKET
- ERROR: INVALID SYMBOL FOLLOWING A CLOSED BRACKET
- ERROR: INVALID SYMBOL FOLLOWING AN OPERATION
- ERROR: UNRECOGNIZED INPUT
- ERROR: MISSING '(' or ')'
- ERROR: MISSING '[' or ']'
- ERROR: CANNOT END AN ASSIGNMENT STATEMENT WITH A '=', '(', or '['
- ERROR: CANNOT END AN ASSIGNMENT STATEMENT WITH A '+'
- ERROR: INVALID ID
- ERROR: INVALID SYMBOL IN ASSIGNMENT STATEMENT

Executor Error Messages

- General Notes:
  -

- ERROR: CANNOT EXECUTE FILE THAT FAILED TO COMPILE
- ERROR: UNRECOGNIZED FLAG
- ERROR: OBJECT FILE IS EMPTY
- ERROR: INVALID RANGE
- ERROR: ATTEMPTING TO EXCEED THE RANGE OF CORE MEMORY
- ERROR: CORE FILE IS EMPTY
- WARNING: ".literal" FILE IS EMPTY

stop

- ERROR: MISSING TERMINATING COMMAND

cdump

- ERROR: CDUMP FAILED

aread

- ERROR: ATTEMPT TO READ OUT OF RANGE

awrite

- ERROR: ATTEMPT TO WRITE OUT OF RANGE

loop Errors

- ERROR: A MAXIMUM OF 7 LOOPS CAN BE NESTED
- ERROR: MISSING LOOP-END

loop-end Errors

- ERROR: MISSING LOOP
- ERROR: A MAXIMUM OF 7 LOOPS CAN BE NESTED

Assignment Expression Errors

- ERROR: INVALID ACCESS TO MEMORY
- ERROR: ATTEMPT TO READ OUT OF RANGE