Programmer's Guide to the Package krclater18, a Compiler and Executor for the transy Language

By Kelsey Clater

General Structure of Compiler Summary

1. **Pass 1**
   a. Pass one creates a .noblanks file
   b. The first pass of the program is handled by the noblanks function found in the noblanks.c file. It removes all spaces, tabs, blanks lines, and uppercases all characters in the input program with the exception of literal strings (i.e: anything enclosed in quotes). Additionally, all line labels are removed and stored in the Line Label Table. Finally, an array keeps track of which source lines correspond to which object lines, or lines that will be translated to object code. This helps keep track of which line errors correspond to. Additionally the nop command is place on empty lines with line labels. All output is stored in the .noblanks file.
   c. Example:
      i. "lineLabel: read a" would be translated to: "READA"
      ii. The line label is removed, spaces are removed and all characters are uppercased
2. **Pass 2**
   a. Pass two creates a .obj file
   b. The second pass of the compiler reads in lines from the .noblanks file one at a time. If a valid command was found on that line, it is passed to a function with the corresponding command name. All functions named after commands begin with a t in the compiler (ex: read is handled by the tread function). Each t-function is located in its own file. These functions are responsible for:
      i. Parsing the input lines for compiler errors
      ii. Storing variables and constants in the Symbol Table
      iii. Storing literal variables and literal strings in the Literal Table
      iv. Generating the corresponding object code for that line which is stored in the .obj file.

General Structure of the Executor

1. All values from the .core, .literal, and .obj file are loaded into abstract data types (ADT)
2. Based on the command values found at the beginning of each line, functions are called with that correspond to each command. Each of these functions begins with an e (i.e. read is handled by eread) and are located in their own file. These functions are responsible for executing the line of object code where the command was found.

Constants

All constants are located in the constants.h file

Symbol Table ADT

Located in the SymbolTable.c file. The symbol table stores variables and constants with their corresponding index in memory. Variables are stored in the order they appear.

Line Label Table ADT

Located in the SymbolTable.c file. The line label table stores line labels with the corresponding object line they were found on

Literal Table ADT

Located in the SymbolTable.c file. The literal table is generated during compilation stores all literal variables and their values in addition to literal strings

Object ADT

Located in the object.c file. The object ADT is a two dimensional array of integers and stores one line of object code in each line.

Core ADT

Located in the core.c file. The core ADT is generated from the .core file created during compilation. It is a two dimensional char array. It is able to manipulate values of variables.

Range ADT

Located in the range.c file. The range ADT is for range checking in the aread, awrite, and assignment commands. When an array is dimmed, its start and end memory locations are stored. If the user exceeds the boundaries of an array, the range ADT will throw an error.

Transducer ADT

Located in the transducer.c file. The transducer ADT which implements algorithms and two stacks to simulate a matrix in Compiler Construction: A Practical Approach by Dr. James Miller. The transducer is an algorithm that helps parse assignment statements in the compiler

.noBlanks File

The .noblanks file is created from pass 1 of the compiler

.core File

The core file is generated at the end compilation and is used during execution. It contains the values of all variables and constants. Variables are initialized to .0123456789 and constants are assigned their constant value. All of these values are loaded into a two dimensional abstract data type called core in the executor. The last line of the core files holds a zero if compilation failed and a 1 if compilation was successful.

.literal File

The literal file is generated during compilation and is used during execution. When a literal string or variable is found, it is copied into the literal file. Literal variables are represented by blanks lines, while the literal string is represented by a copy of itself

.obj File

The object file is produced during compilation and is used for execution. Each line of transy code is translated into object code.