CSc 354 – Assignment #1 – Hamer – Due: 09-18-24

Write a complete module used to maintain the **symbol table** for the SIC/XE assembler:

* use a **binary search tree** implementation to store each symbol along with its associated attributes
  + exportable binary search tree operations: insert, search, view (create, destroy – non-class based)

Write a complete **main/driver program** that uses the symbol table module to process two text files:

* **SYMS.DAT** used to populate the symbol table.
  + file format (each line): SYMBOL VALUE RFLAG
* zero or more leading spaces in front of the SYMBOL attribute – ends with a colon (:)
* one or more leading spaces in front of the VALUE and RFLAG attributes.
* search file used to search the symbol table.
  + search file name obtained from the command line.
  + file format (each line): SYMBOL
    - zero or more leading spaces in front of the SYMBOL attribute – no colon

**Basic Algorithm**

1. read symbols and their attributes one line at a time from the file named **SYMS.DAT**.
   * invalid symbols and/or invalid symbol attributes are not inserted in the symbol table.
     + display the symbol along with a detailed error message.
   * valid symbols with valid attributes are inserted in the symbol table.
2. read symbols one at a time from the search file.
   * if no file name was specified on the command line then prompt the user for the file name.
   * invalid symbol: display the symbol along with a detailed error message.
   * valid symbol: search for the symbol in the symbol table (significant portion only).
     + found: display the symbol and its associated attributes.
     + not found: display the symbol along with a detailed error message.
3. perform an **inorder traversal** of the symbol table.
   * display all symbols and associated attributes in a tabular format using output formatting techniques.

**SYMBOL** (also referred to as a **label** in assembly language programing)

* starts with a letter (A..Z, a..z).
* followed by letters (A..Z, a..z), digits (0..9), and the underscore (\_).
* maximum of 10 characters in length in the source program – does not include the colon (:)
  + only the first 4 characters are significant – only the first 4 characters are stored in the symbol table.
* not case sensitive (CSC\_354, CSc\_354, csc\_354 – all the same symbol – stored as CSC\_).

**VALUE**

* signed integer value (+, –, 0..9).

**RFLAG** (Boolean)

* false
* true
* not case sensitive.

**IFLAG** (Boolean)

* indicates whether or not a symbol has been defined within the current control section (true for now).

**MFLAG** (Boolean)

* indicates whether or not a symbol has been defined more than one time in the same control section.
* each valid symbol is inserted into the symbol table exactly one time (invalid symbols are never inserted).

**Sample Program Run**

**Step #1** – **SYMS.DAT** // File names are case sensitive in Linux as well as some languages

ABCD: 50 True // Valid – insert **ABCD** and all attributes into symbol table (\*)

B12\_34: -3 false // Valid – insert **B12\_** and all attributes into symbol table (\*)

a1B2\_c3\_D4: +45 true // Valid – insert **A1B2** and all attributes into symbol table (\*)

ABCD!: 33 true // ERROR – symbols contain letters, digits and underscore: **ABCD!**

1234567890: 0 false // ERROR – symbols start with a letter: **1234567890**

ABCD\_EF: +100 TRUE // ERROR – symbol previously defined: **ABCD** (+)

a1234: 3.5 FALSE // ERROR – symbol **a1234** invalid value: **3.5**

XYZ: 100 5 // ERROR – symbol **XYZ** invalid rflag: **5**

(\*) no message displayed for valid symbols with valid attributes – set IFLAG to true – set MFLAG to false

(+) set MFLAG attribute to true for symbol **ABCD**

**Step #2** – **search file**

ABCD // Found – display symbol **ABCD** and all attributes

A1b2C3\_xYz // Found – display symbol **A1B2** and all attributes

CDEF // ERROR – **CDEF** not found in symbol table

abc~def // ERROR – symbols contain letters, digits and underscore: **abc~def**

a1b2c3d4e5f6 // ERROR – symbols contain 10 characters maximum: **a1b2c3d4e5f6**

**Step #3** – **view the symbol table – required output order and format**

Symbol Value RFlag IFlag MFlag // Do not allow the data to scroll off of the screen

// Hold the output every 20 lines – Tera Term screen size

A1B2 45 1 1 0 // Continue when user indicates to do so

ABCD 50 1 1 1

B12\_ -3 0 1 0 **// Perform an inorder traversal of symbol table**

**Notes and Suggestions**

* Do NOT stop on error!!! Process all data in both files completely!!! Display detailed error messages!!!
* Check for errors in all symbols and all symbol attributes read from both files
  + Step #1 SYMBOL VALUE RFLAG
  + Step #2 SYMBOL
* Convert all values to one common format:
  + All symbols were converted to uppercase: csc, CSC, CSc => CSC
  + All flag values were converted to Boolean values: false => 0 true => 1

**Other Requirements**

* The module/program must use proper data abstraction techniques.
  + See the Assignment Requirements document on the course web site.
* All module/program files must be fully documented.
  + See the Documentation Requirements document on the course web site.
* All C/C++/Java programs must build and run using the Computer Science Linux server: cscssh.sdstate.edu
* All C# programs must build and run as a Visual Studio 2022 Community Edition solution/project.
* Zip all files together and upload to D2L before class on the due date.
  + Visual Studio zip entire solution folder (containing project folder)
* All duplicate or near duplicate assignments will earn a grade of 0.