CS4100/5100 COMPILER PROJECT

Part 1, Foundations: Assignment 1, Reserve Table

The purpose of this assignment is to give the student a practical introduction to the structure and usage of the foundational data elements of the compiler project. The code developed here will be used throughout the rest of the semester. **Creating the 3 upcoming Abstract Data Types (ADTs), exactly as described, is crucial to the success of the project!** While there is room for individuality in the actual *implementation*, it is necessary that each structure utilize exactly the ***interfaces*** and ***storage capabilities*** described, in order for them to be used effectively and efficiently in the development of the rest of the project. **The language required to operate with the automated testing system for all submissions is Java. Read all requirements below carefully!**

Program Objectives- 1: Implement a class for the Reserve Table according to the interface requirements provided below.

Main Program:
The student should do initial testing by creating their own main program for this part of the project. A main test program and expected output .txt file will be provided by the instructor to test the ADT methods and verify that they work according to specifications. Code will be submitted via Canvas.

The First ADT to Implement: The Reserve Table
The following ADT must be established as a Java class, according to the exacting requirements below:

Class ReserveTable
The class must be named ReserveTable, and be implemented in its own .java file. This is a lookup table/association list ADT that will be used later for the language Reserve Words and the assembler Opcode lookups. Later, two separate instances of this class will be instantiated, one to hold the opcodes, and one to hold the reserved word list for the language.

The underlying data structure is an unsorted, indexed list which can be accessed like an array. Each indexed entry is a pair consisting of a Name string and an integer Code value. The table as we use it is static, and intialized once at the start of the program, and then used only for lookups later on. There are no future updates or deletions from the list. An example of the data it might contain would be:

```
Index   Name    Code
 1      IF      27
 2      DO      11
 3      ELSE    34
        ...
```

The needed ReserveTable methods are:

ReserveTable(int maxSize)
Constructor, initializes the internal storage to contain up to *maxSize* rows of data. The maximum size of the structure will not change, and the ADT should operate correctly with fewer than *maxSize* entries; it should not be required to add all the entries to fill the structure.

int Add(string name, int code)
Adds a new row to the storage with *name* and *code* as the values. Returns the index of the row where the

data was placed; just adds to end of list, and does not check for duplicates, and does not sort entries.

int LookupName(String name)
Returns the integer *code* associated with *name* if *name* is in the table, else returns -1 to indicate a failed search. This must be a **case insensitive search.**

String LookupCode(int code)
Returns the associated *name* contained in the list if *code* is there, else an empty string to indicate a failed search for *code*.

PrintReserveTable(String filename)
Prints to the named plain ASCII text file the currently used contents of the Reserve table in neat tabular format, containing the index of the row, the name, and the code, one row per output text line. Any empty lines must be omitted from the list. *(If the list was initialized to a maxSize of 100, but only 5 rows were added, only 5 rows will be printed out.)*

IMPLEMENTATION HINTS:
1) The ReserveTable must keep track of how many elements are currently in use, regardless of the number of allocated slots created in the constructor.

2) It is *imperative* to recall that Java does not do the correct string comparison in a way that one might intuitively expect based on other languages.

BAD: if (myString1 == myString2) {...} *This compares addresses, essentially*
GOOD: if (mystring1.CompareToIgnoreCase(myString2) == 0) {...} *Alpha compare*

Turn-In Requirements
The turn-in consists of **three files:** the .java class file, the console output from the instructor-provided **main.java** program (saved as a .txt file), and a Reserve.txt file created by the call to ***PrintReserveTable(...Reserve.txt)***.