# Program 1 CS 237 Kutztown

**Topic:** Recursion, Data Structures, Justification of Choice of Data Structure

**Points:** 25

**Due:** TBA in class. Submit late at your own risk policy is in effect.

**Overview:** This project requires you to implement a solution to a simple problem using several data structures, incorporating both iteration and recursion. The basic problem is to read a text file and count the number of occurrences of its constituent tokens. You will perform this task using the following data structures:

- Parallel arrays
  - One array of string and one array of int, with corresponding elements related by the indices
- Array of object
  - A language-based array of a class of type *WordData*
- Built-in library sequential container object
  - A list of *WordData* objects of unlimited length

```
1: PARALLEL ITERATIVE
2: PARALLEL RECURSIVE
3: OBJECT ARRAY ITERATIVE
4: OBJECT ARRAY POINTER (No Indexing)
5: OBJECT ARRAY RECURSIVE
6: SEQ CONTAINER INDEXING
7: SEQ CONTAINER ITERATOR (No Indexing)
8: EXIT
```

where *WordData* is an object-type with the following attributes:

- A string to hold the word
- An int to store its multiplicity in the input
- Member functions to
  - Construct an immutable instance
  - Increment the counter
- An associated stream operator to print the word and count formatted appropriately.

**WHAT YOUR PROGRAM WILL DO:**

Your program will start by prompting for the name of a file and, if it exists, opening it according to these specs. (If the file is not found an error message is printed and the program terminates.) Next, it presents the user with a menu (see box above). The user chooses one of the options and the application invokes the appropriate function. Alternatively, the program can be run in batch mode, where it runs all options non-interactively, obtaining the file name from the command line.

Each menu item is carried out in a function (referred to as a *subfunction)*. It designates an associated print function. Each subfunction takes the input file stream as its parameter. It will declare the appropriate container, parse the file's contents into that container, and call the designated function to output the data, well labeled, in tabular form. There will be 8 menu items and 7 corresponding subfunctions (the 8$^{th}$ menu item is the exit option). Some or more likely, all, of these 7 functions are best written to call other functions to carry out parts of their task.

**Requirements, Notes and Suggestions:**

- You may implement the program in any language available on acad. Instructor preference: C++
  - If your chosen language doesn't have explicit pointers, contact the instructor to find an alternative
- There are to be seven print functions, corresponding to the menu items, **but you will only write three functions for reading the data**, one for each data structure.

- The arrays (only) will be limited to a capacity of 10. The first 10 ***unique*** tokens take up the array slots. The multiplicity of the unique words in the input includes repeat occurrences before and after the 10th unique word has been encountered.
- The object array pointer option can print iteratively or recursively. **It may not use indexing**. Indicate your choice in the function's header block.
- Data is taken as it is input using stream extraction, e.g. *data, Data,* and *data!* **are** distinct.
- Each time the data is output (In tabular form), the table created should be labeled with the data structure and whether it was generated iteratively (and note if using pointers) or recursively. It should be clear which menu choice was selected. The table must have a neat, crisp appearance.
- DO NOT write the code for carrying out a menu item's task within a case of a switch. Use a subfunction as described above. This is a point of emphasis.
- The program is to be runnable from the command line. This is denoted as *batch mode*. If there is a command line argument, your program will assume it is a file name. It will then process the data in the same manner as if the input had come from the user. If the file is found it will apply each of the seven methods without any user intervention, labeling each output as specified above. After all possibilities are exhausted, the program terminates.
  - The file *CmdArgs.cpp* is provided to demonstrate processing command line arguments.
- Recursive functions may be called by a function that is in turn called when the corresponding menu item is selected, i.e. the subfunction can call a function that calls the recursive function.
- There is a C++ *WordData* class available on acad in the project's directory, and also on the web.
- You are to provide a working, correctly designed makefile that builds an executable named, precisely, **p1**.
- You must rewind the data file after each menu selection completes. Depending on the language you may be able to move the file pointer to the beginning using seek(), or you may need to close the file and then reopen. In either case, reset the bad bit (set at eof from the previous read) using clear().
- **Work incrementally**. This assignment provides good practice of this critical skill, where you will implement and test one data structure at a time by writing its parse function and the print functions, also one at a time, testing each before moving on to the next.
- You must carefully and completely document your code and place id blocks in all files, including those provided from external sources (e.g. the course instructor). This includes labeling parameters by type (import, export, or import/export) and member functions as mutators, facilitator, or inspector. Lazy, chintzy, or otherwise substandard documentation WILL be penalized.
- You must provide a file with name **readme** (pdf, txt, etc) that contains a well-organized, coherent, and at least somewhat erudite **narrative** (NOT just your choice) stating:
  - the STL container you used, which ones you considered and why you chose the one you did, in the Project 1 dropbox.
  - An explanation of how you handled inserting new terms and discovery of duplicate tokens. Include a statement about whether you feel your method was efficient or whether some other method may have been more efficient.
- There will be no support for setting up turnin after the deadline.

**Deliverables:**

**D2L:** Your readme. Make sure it is readable,

**Turnin:**

- WordData class, application, and the makefile. Turnin allows wild cards, but NOT directories
  - Do NOT submit o files, executables, backup files, etc. Follow specs.
- At least two test files that you used in testing this project.
  - One must have less than 10 unique tokens and one should have more, including unique and duplicates of word(s) that appear after the 10th unique word is placed.

Penalties for non-compliance.

Not setting up turnin is not an excuse for late submission.
E-mail or D2L submissions will NOT be accepted.