CIS361 System Programming

A Cross-Reference Generator

(Due Nov. 3, 2016)

In this project, you design and implement a program that scans a C source file and outputs all identifiers, along with the line numbers on which they occur. Your program will maintain an ordered collection of the data and, when the source file has been read, output identifiers and their corresponding line numbers. An ordered collection of such identifiers is referred to as a cross-reference generator. As a matter of fact, this is a general problem that occurs in many other contexts. For instance, it can generalize the creation of an index for a book. It can also list, for each function in a program, the names of all other functions that it directly called.

Shown below is a sample C source file, date.c, which can be found in the "Proj 2 - F16" folder on the instructor's Web site.

```
#include "date.h"           /* date type and prototypes */

int areDatesEqual(const Date *xptr, const Date *yptr)
{
        return xptr->year  == yptr->year  &&
        xptr->month == yptr->month &&
        xptr->day   == yptr->day;
}
```

In a programming language, an identifier is a sequence of one or more letters, digits, and underscore, but it cannot begin with a digit. Here, for simplicity, you may consider a keyword like include as an identifier; however, words in a program comment are not nor that in a string constant (i.e., anything that appears between double quotes). Note that an identifier may appear more than once on a line; obviously, there is no need to have duplicate line numbers for that identifier.

Use a linked list to store an ordered collection of identifiers. Each identifier and the line numbers on which it occurs is stored in a structure, called node. When an identifier is read, check and see if it is already in the list of nodes. If so, add the current line number to the node. Otherwise, create a new node and add it to the linked list. After the entire input is read, iterate over the linked list and output identifiers and their corresponding line numbers.

A node stores the identifier and the line numbers on which it occurs. Implement the set of line numbers as a queue of integer. This makes sense because when your program outputs those numbers, they will come out in the same order as they went in – increasing order.

Design each data structure in your program as an abstract data type (ADT), meaning that the client code can use the data structure without having to know the detail of its implementation. According to the “separation of concerns” principle, create several files to organize processing elements that serve for different purposes in your program, such as:

- queue.h contains type definitions and function prototypes for queue
- queue.c contains function definitions for queue
- list.h contains type definitions and function prototypes for list
- list.c contains function definitions for list
- proj2.c contains main() and other functions you have introduced

Your proj2.c has to include queue.h and list.h. Each .h file provides what your program needs (like the interface of a class in an object-oriented program) and the related .c file hides what it does not need to know (like the implementation of a class). Since you will introduce a number of functions as needed in this program, you should give a meaningful name to each of them and provide comments to help the reader understand its functionality.

In addition, use command-line arguments to pass the input and output filenames into your program. Also use git to manage your source code and use make to build and test your program.

When your program works correctly, get a hardcopy of the source code, the makefile, a git log, and the test data as well as program output for submission.