

CSI 333 – Programming at the Hardware/Software Interface – Fall 1999

Laboratory Exercises 3

First checkoff **MUST** be completed in a lab session for you to receive passing credit. Get permission to go to another lab session if you must miss your own.

Second checkoff (all source files plus Makefile) is to be turned in to project lab3.

1 Introduction

This Lab will get you started for Project 2. Project 2 is to write Keyword In Context (“KWIC”) index creation software. Briefly, this software will print an index for a file of lines. Each line contains words. Since there will be one index output report for every **word occurrence in a line**, you will need a data structure in which each record represents a line together with one occurrence of a word within that line. Lists of such objects are also required.

The software will also require a module to handle reading each input line, checking or overcoming buffer length limits, and constructing the character array to hold it.

This Lab will emphasize:

- Rapid first prototype implementation of a simple software design in terms of **modules** and **interface/implementation separation** for each module.
- Learning the commands to perform all build steps “by hand.”
- Applying the basic operational principles of **make** to write and use a simple Makefile so the build steps can be automated **safely** and **efficiently**

2 Lab Software Design

For the inlab checkoff, create the C++ files

`lines.h lines.cpp strread.h strread.cpp main.cpp tester.cpp`

to contain data type, function prototype and possibly external static variable declarations (in the `.h` (interface) files and **function stubs** and possibly static variable definitions in the `.cpp` (implementation) files.

2.1 strread module

It provides function with prototype

```
extern char * strread(void);
```

which tries to reads a line from standard input. If standard input is at the end of file, return 0 (the null pointer). Otherwise, read the line, avoid all buffer overflow, allocate a free-store `char` array and get the line contents into it, and return the pointer to the array.

WHAT TO DO: Put the function prototype in `strread.h` all by itself.

For `strread.cpp`: Write the include statement

```
#include "strread.h"
```

near the beginning of `strread.cpp` (after standard library includes), and write the definition (“body”) of `strread()` after all the includes.

If you don’t remember how to read a line of input in C++, just write a really simple **stub** function, which could even be

```
char * strread( void )
{
    return 0; //stub: always return end of file indication
}
```

or

```
char * strread( void )
{
    char dummy[] = "strread stub dummy string\n";
    char * pch = new char[ sizeof( dummy )];
    strcpy( pch, dummy);
    return pch;
}
```

For simplicity of checkoff 2, you can make the program print input line too long. and exit to avoid buffer overflow.

2.2 lines module

It provides the datatype

```
struct line {
    char *pchars;
    int length;
    int lineNumber;
    struct line *pNext;
};
```

plus functions to (1) construct (by using the C++ `new` operator) one line from a `char *` and (2) link a line into a linked list of other lines.

For help with applying the principles illustrated for the `strread` module, ask an assistant during the Lab.

2.3 main and tester modules

These should each define a `main` function, they should each use both `strread` and `lines`, and they should both test these modules in different ways.

3 Software Build Steps and Files

Practice compiling each .cpp file by invoking the compiler manually, for example:

```
g++ -g -c strread.cpp
```

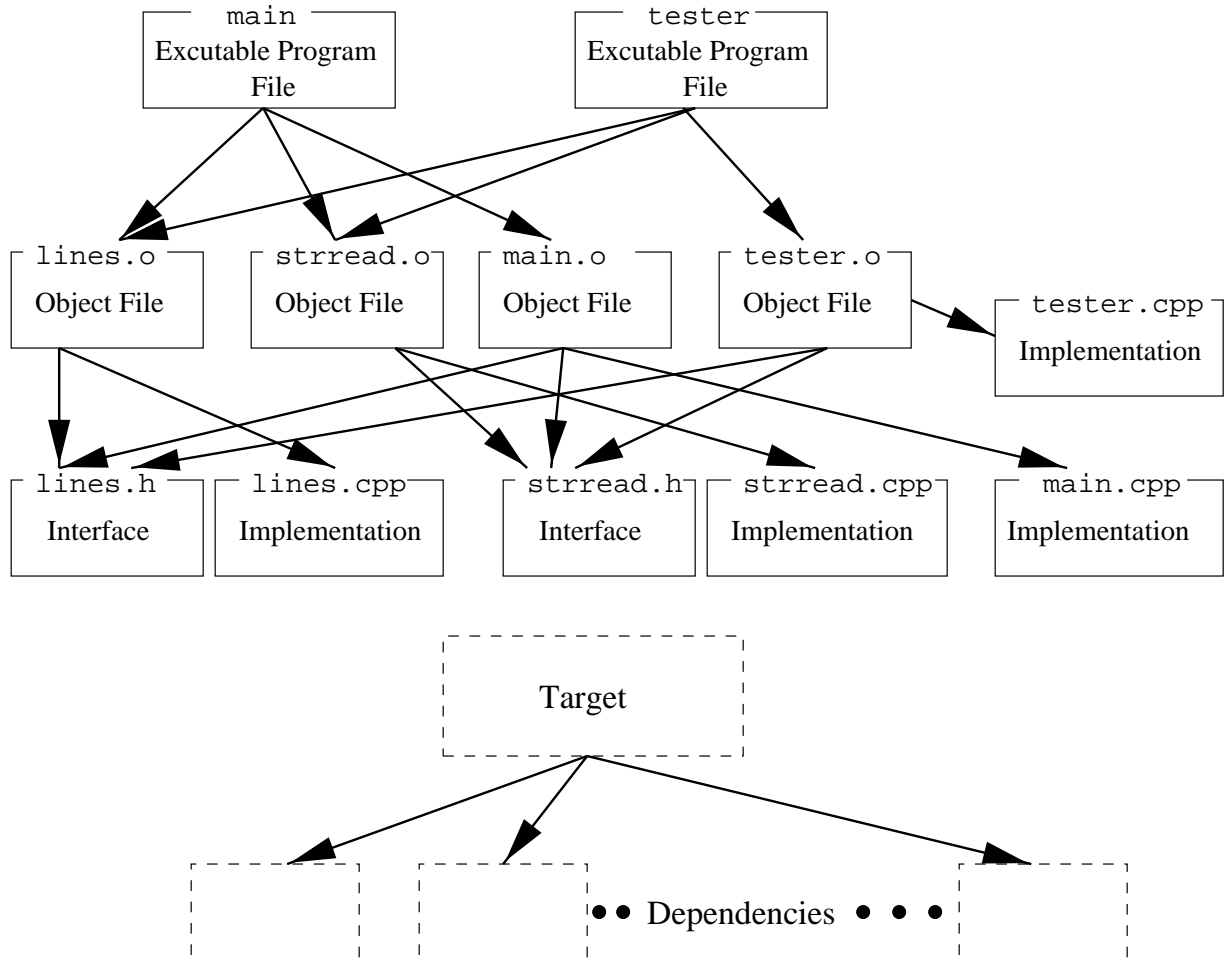
Fix syntax errors. Observe the object files with the `ls -l` command. (You did start the project in a new directory, didn't you? If not, now is the time to move your work to a new directory.)

Practice linking the resulting object files by using commands like

```
g++ -o main main.o strread.o lines.o  
g++ -o tester tester.o strread.o lines.o
```

If you are new to the Lab, or simply forgot some details, ask the TA or a neighbor to help get this done within the session. For this lab, it is OK to get help from others in writing the C++ code.

Think about how the software you are building is structured according to the figure:



4 Using Make

Open emacs, (1) type `C-H i`, (2) move the cursor to **Make**, (3) press enter, (4) select the **Introduction** page similarly, skim the Introduction, (5) select and read **Rule Introduction** and **Simple Makefile**

After (or before) the Lab, study **How Make Works**, and Stroustrup sections (9.1), (9.2.1), and (9.3). Ignore the material on namespaces, for this course.

Write your own Makefile after the model given in the **Simple Makefile** page. Warning: Even if you have used a template makefile in previous courses, follow the instructions for this course now.

Now verify that your makefile works by using the `touch` command on your various `.cpp` and `.h` to simulate you making modifications of them. Also, experiment with `rm` (file deletion command: **Don't use on .cpp or .h files!!** on object or executable files.

Verify that `make` (when configured with your Makefile) does **all and only** the build steps that are made necessary by your modifying or deleting various files.

For checkoff ONE, the TA will “touch” or remove various files and observe whether or not the `make` command rebuilds the given target both **safely** and **efficiently**.

“Safely” means that if any file on which any target depends either directly or indirectly is modified, then necessary targets are rebuilt.

“Efficiently” means that if it is not necessary to rebuild a target, then it is not rebuilt. Rebuilding all objects (such as `strread.o` after touching a file on which only a few depend (such as `lists.h`) is not acceptable.

The TA will also check that your Makefile is “Simple”: All rules, dependencies and commands written out explicitly.

5 Final Checkoff

Get the implementations to actually work, make sure to test them and your Makefile, and submit just the sources (`.h` and `.cpp`) and the **Makefile** to your section of the `lab3` project using `turnin-csi333`

Strict Due Time: Before the beginning of the NEXT lab session of the section in which you are registered.