GENERAL INFORMATION
about ALL PROGRAMS in CS163

ALWAYS make a backup of your programs before using tar
Double check that the arguments specified with for tar are correct!
***Penalties** will apply to all submissions incorrectly archived/uploaded*

Scope: For each project, we only have approximately 2 weeks to complete the write-ups and code. Therefore, it is critical that you focus on a limited scope, with an emphasis on the class(es) and data structures, rather than building a comprehensive application. Each program this term will have you create an Abstract Data Type (ADT), **which is what you will be primarily graded on**. We will learn about Abstract Data Types in lecture #2. *We will primarily grade your projects based on the use of classes, member functions, arguments, data structures, pointers, efficiency, and how well the software demonstrates that it has been fully tested. The code needs to compile and run, and include a test program for the member functions provided.* **It is not appropriate to hard code in the test cases - all tests should be interactive with the user.** Your user interface must be clear for us to thoroughly test all features. This means you need to build a simple menu interface for each program this term.

Using Recursion: **The use of recursion** should be included **at least once** in each assignment to better prepare us for future programs. The more you use recursion, the better equipped you will be for the midterm and final proficiency demos.

Program for Efficiency: Avoid multiple traversals whenever possible.

Class construct: Most programming assignments this term require only one class but you have the option to also implement classes to represent the underlying data as well. The use of structs is allowed. We expect three source files: one .h file for the class interface, one .cpp file where the member functions are implemented, and another .cpp file for your test program. *If you intend to create more files than this, please contact your Instructor.*

Design is the First Step: The first step will be to design your software. This should include designing classes that are well structured and that efficiently use the data structure. *Take a look at the style sheet for instruction on the topics that your write-ups need to cover. Also, the writeups should not be archived (don't 'use tar or zip)*

Creating Test Plans: Part of the first step will be to develop a test plan. This includes planning out each of the cases that need to be implemented and subsequently tested.

Program #1
CS 163 Data Structures

Submit your program to the D2L Dropbox

BACKGROUND INFORMATION for Program #1:

This first program is an exercise in building, traversing, and destroying linear linked lists.

One of the great things about computer science is that there are many local companies along the west coast hiring new graduates and offering internship programs. This is one of the most exciting times that I have seen as a computer science professional. So, as we begin our journey in computer science, it would be good to start keeping information on the local companies. We want to know what each company does, what kind of graduates they are looking for, and what experience they need. We also want to collect information about what people think about these companies (i.e., how they rate the) so that we can start planning our own futures.

Programming Assignment: For the first programming assignment, we are going to create a program that helps the user create, organize, and search for companies that are looking to hire entry level computer science positions (or internships). Because we will want to use this program all through our time at PSU, the data should be stored in an external data file (we wouldn't want to lose the information after using the program!!). The data file should originally start as empty and as the user enters in information into the program, your program should ultimately store the data in the external data file. Then this data would be available for the next time you run the program! My advice would be to have the constructor load existing data from the external data file and then save the data using the destructor. (Remember when working with an external data file, you will need to store a delimiter between each field which is not part of the data set. Common delimiters are ':' and '|'. The last field should have a newline after it.) *Please upload your data file when turning in the programming assignment! It should be part of your archive (zip or tar).*

The Data

The information that we want to track should include the following data. You are welcome to add more to this if you find other information that is interesting:

1. Name of the company (e.g., Apcon)
2. A description of what they do (e.g., Build network visibility and security Solutions)
3. Qualifications (e.g., C, C++, Java)
4. Degree Requirements (e.g., Sophomore level BS in CS)
5. Location (e.g., Wilsonville)
6. Job Specification (e.g., Internship)
7. Pay rate (e.g., $29,000 to $43,000)
8. Rating (e.g, 2.3)
9. Review (e.g., A great place to work)

The Abstract Data Type

Topics 1 and 2 along with Lecture #2 will introduce the concept of abstract data types. Every program this term will be focusing on isolating the data structure from the application program. We do this with the class construct. To work with the job data, we will be building a Job_List Abstract Data Type (ADT). The **public member** functions should include the following:

1. Constructor - Construct an object and initialize the data members and preload existing data from an external data file.
2. Destructor – Write the data out to an external data file and release all dynamic memory and reset data members to their zero equivalent value
3. Add a job entry
4. Add a new location (e.g., Seattle)
5. Edit the review information about a particular job
6. Display all jobs at a particular location (these should be in sorted order by the company name)
7. Display all jobs for all locations
8. Remove a job

Data Structures

In memory, we will want to store the job information as a linear linked list of locations where each location has a linear linked list of the jobs at that location. You will need two different struct "node" types to support this.

General Syntax Rules

1) Do not use statically allocated arrays in your classes or structures. All memory must be dynamically allocated and kept to a minimum!
2) Global variables are not allowed in CS163; global constants are encouraged
3) **Do not use the String class! (use arrays of characters instead);** you may use the cstring library with strlen, strcpy, strcmp
4) Avoid using break or return from within the body of a loop
5) Avoid using while(1) type of syntax
6) Strive towards structured programming techniques
7) Remember that at least one function needs to be written using recursion!
8) For functions that have non-void return types, make sure to return a value through each path through the function.
9) NEVER pass class types by value (always by reference)
10) NEVER return class types by value.
11) Use the iostream library for all I/O; do not use stdio.h for I/O
12) MOST IMPORTANT:
    BACKUP your files before creating a TAR archive!!!!

General rules for building ADTs

1) The ADTs should be designed to "hide" the data structure as part of the private section of the class. This means your head pointer to a node should be a private data member in your ADT.
2) All data members in a class must be private
3) Public member functions should NEVER have a node pointer as an argument (Remember client programs should not be involved with the physical data structure when working with ADTs)
4) Never prompt and read from the user when inside a class member function; this should be done in the "client" program when developing ADTs
5) Never output error messages from a class member function; this should also be done in the "client" program when developing ADTs
6) Each public member function should have a way to communicate success or failure back to the "client" program. This can be done with a returned value, and argument passed by reference, or exception handling.
7) Use modular design, separating the .h files from the .cpp files. Remember, .h files should contain the class header and any necessary prototypes. The .cpp files should contain function definitions. You must have at least 1 .h file and 2 .cpp files. **Never implement functions in your .h file! And, never "#include" .cpp files!**

8) D2L can't work with multiple source files, so we have to archive our source (.h and .cpp) files as specified in the linux & vim manual. Make sure to follow the instructions in the main very precisely. But before EVER using tar, first backup your files (you can ftp them to your desktop for example).

9) NEVER archive your writeups. Your writeups need to be read by the grader on D2L and should be uploaded as separate documents.