STYLE SHEET
CS 260: Data Structures

Design Write-up and Test PLan

- **Prior to the first four programs**, submit a design write-up and a test plan.

- **Include a "test plan", which outlines each test case and the expected outcomes.**

- Your design write-up should be at least one page in length **or at least 600 words**.

- The design write-up should be in a file named: design#.txt. It should describe the major design considerations for your program, and all data structures and algorithms that you intend to use. It must be a plain text document. *You must have your name in this file.*

- The design write-up must be written in English using complete sentences. It should describe:
  1) **What functions your ADT will need to have**
  2) **How each function will report success/failure back to the calling routine**
  3) **How each function will shield the client program from the details of the data structure**
  4) **What data structures you will be using and how each of the special cases. Consider drawing pointer diagrams with "before" and "after" diagrams for each special case.**
  5) **Which function/functions are recursive?**

- Each major design issue should be described. In the design considerations, discuss what the main design considerations are, why they are the main design considerations, how you solved them, and why you solved the way you did. **Think in terms of analyzing your solution!**

- Please note that your design write-up should not look like C++ code; words such as "cin", "cout", "++", "--" should not be part of your design document. Instead, describe the major tasks that your program accomplishes. Any design document submitted which looks identical to C++ code will not be accepted.

Efficiency Review Write-up

- **With each program (all 5 programs)**, submit a file (e.g., review1.txt) that reviews the design after creating your program, and all data structures. DO NOT include this file in your tar archive.

- Your efficiency review write-up should be at least one page in length **or at least 600 words**.

- The efficiency write-up must be a plain text document. *You must have your name in this file.*

- Answer the following questions in your design write-up:
  1) **How well did the data structure selected perform for the assigned application?**
  2) **Would a different data structure work better? If so, which one and why...**
  3) **What was efficient about your design and use of the data structure?**
  4) **What was not efficient?**
  5) **What would you do differently if you had more time to solve the problem?**

<u>Requirements</u>

The following are source code format requirements.

1. A consistent pattern of indentation. See the attached C++ style requirement for examples.

2. White space (blank lines) to separate program sections. At least three lines of white space must separate functions.

3. In your program heading, explicitly list the input data to your program, following the header "INPUT:". This should include the source and format for the data being input to the program.

4. Also in your program heading, explicitly list the output that will result from running your program, following the header "OUTPUT:". This should include the nature and format for the program results.

5. For functions, make sure to include a header comment for each function; this should explain the purpose of the function as well as describe the INPUT and OUTPUT parameters.

6. NEVER use global variables in these programs!

7. Avoid the use of exit, continue, or break to alter the flow of control of loops

8. Avoid using while(1) type of loop control

9. NEVER use the string class in CS260 – instead use arrays of characters

10. Abstract Data Types should NEVER prompt the user or read information in. All information should be passed from the client program or application

11. Abstract Data Types should NEVER display error messages but instead supply success/fail information back to the calling routine.

<u>Documentation of Source Code</u>

Requirements for internal documentation in the form of comments are listed below.

1. A heading explaining what the program does and listing the name of the program author, date, class number and program number. A heading must be supplied indicating the purpose of the entire program; in addition, each separate function should have a heading describing it purpose and arguments.

2. Each file should also have a heading, explaining the purpose of that module <u>and</u> the listing the filename (this is important!).

3. A comment following each variable declaration telling what it will be used for.

4. Comments to explain any program action whose purpose is not obvious to anyone who reads the code.

5. Use mnemonic names for identifiers that relate to their purpose.

Files to Submit with Each Assignment

1. **Programs must be emailed to me as tar format on the linux server.**
2. **Submit your Design write-ups, design#.txt and Efficiency write-ups efficiency#.txt** separately into D2L (not within your tar archive)
3. Do **NOT** submit the tar file or individual .h, .cpp file to D2L.

With each program, tar together the following files:

1. Makefile: I will use your makefile to build and clean the project

2. Header files (.h files)
   -- comment the beginning of this file with:
   a) your name, class number, project number, name of the file
   b) a description of what this file contains (purpose of the header file)
   c) structures, classes, prototypes definitions

3. Implementation files of your classes (.cpp files)
   -- comment the beginning of this file with:
   a) your name, class number, project number, name of the file
   b) a description of what this file contains (purpose of the header file)

   -- place the code in the following order (for a class' implementation):
   a) constructors -- default, ones with arguments
   b) destructor (if you have one)
   c) remaining member functions

   -- if you destructor calls another function -- then place that function immediately after the destructor

4. Implementation of the client/application program (i.e., the main), (other .cpp files)

5. Data file if there is any

Instructions to Email Each Assignement

Organize your directories:

Create directory for the class: `mkdir cs260`

Go inside the directory: `cd cs260`

Create directories for the project: `mkdir project1`

Go inside the project directory: `cd project1`

Edit, compile and run your code:

Create source file for the program using vi, the editor: `vi project1.cpp`

An alternative editor is nano: `nano project1.cpp`

Compile the source code (you should create your own makefile): `g++ -o proj1 *.cpp`

Makefile is required for every project. When I test your program, I will use your makefile to build the program.

Run the executable file: `./proj1`

```
  valgrind proj1
```

Email the .tar file containing your project directory

To create a tar file of the project directory:

- remove the executable files in the project directory: `rm proj1 (or make clean)`
- go to the parent directory that contains the project directory: `cd ..`
- `tar -cvf project1.tar project1`

To email the tar file to the instructor and yourself:

```
mailx -a project1.tar lliang your-login-name
```

entering the subject “CS260 Project 1 Submission”

entering the message here (an example: known bugs: ...)

Known bugs: ...

```
.
```

a single period at the beginning of new line will send the email

To check if you have sent the file correctly:

```
mailx
```

assuming the message index is 3, type

```
w 3
```

to save the message. It will ask you for attached file name if the message is not empty. If it's an empty message, 3 is the tar file name. The file will be saved in the current working directory.

The following command will extract from the tar file.

```
tar xvf tarFileName
```

REMINDERS: *Every program must have a comment at the beginning with your first and last name, the class (CS260), and the assignment number. This is essential!*