Program Set #2
Total Points: 30

Three problems must be implemented for full credit. The required (starred) problem marked in the set must be implemented by everyone. See the 2436 Grading and Submission Guide Sheets for additional grading/submission information. Partial credit will be given. (10 points each)

Section One- Choose two problems.

1. Implement Programming Exercise #7, Chapter 17-Word Index (Gaddis) pp. 1120/21. Modify the problem to let the user enter the file name from the keyboard and the index file should be output the screen and not to a separate file. Note how the words are sorted from the textbook. Finally, the program must be able to read text files of various sizes (large text files with punctuation). Output should be user friendly.

Name the program: `WordIndexXX.cpp`, where XX are your initials.

2. Write a program that checks if delimiters of a mathematical expression are nested correctly. The delimiters are parentheses ( ), brackets [ ], and braces { } as delimiters. For a mathematical expression to have correctly nested delimiters, the following two statements must be true:

1. There are an equal number of right and left delimiters
2. Every right delimiter is preceded by a matching left delimiter.

For example, the expression `7 – ((x * ((x + y) / (j – 3)) + y) / (4 – 2.5))` is nested correctly because the two above conditions are satisfied. The expression `((a + b)` is not nested correctly because there are two left parentheses but only one right parenthesis, thus causing statement 1) above to be not met. The expression `{a * (b + c}`) is not nested correctly either. Although there are equal numbers of right and left delimiters, the right brace } is preceded by a left parenthesis (, thus causing statement 2) not to be met.

Input from the keyboard a mathematical expression. The expression may be composed of integer literals, variables, addition, subtraction, multiplication, and division, but the only delimiters allowed are parentheses, brackets, and braces. Output to the screen the expression followed by either `"Is nested correctly"` or `"Is not nested correctly"` on the following line. Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

Sample Run:

```
Enter an equation: {[(a + b) * 10 + c] * 10 + d}

Is nested correctly

Run Again (Y/N): y
```

```
Enter an equation: ((((((((((()))))))))))

Is not nested correctly

Run Again (Y/N): N
```

Name the program: `IsNestedXX.cpp`, where XX are your initials.

3. Write an interactive program for evaluating postfix expressions. The program will allow the user to enter a postfix expression from the keyboard. Assume the expression is entered correctly. Output to the screen the stack evaluation (showing the stack) after each step and the final expression value. The program must use a stack data structure. Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

Sample Run:

```
Enter a postfix expression: 4 5 2 * +

Stack Contents:

After push operation: 4
After push operation: 4 5
After push operation: 4 5 2
After * operation: 4 10
After + operation: 14

Final Value: 14

Run Again (Y/N): y

Enter a postfix expression: 5 7 + 6 2 - *

Stack Contents:

After push operation: 5
After push operation: 5 7
After + operation: 12
After push operation: 12 6
After push operation: 12 6 2
After - operation: 12 4
After * operation: 48

Final Value: 48

Run Again (Y/N): N
```

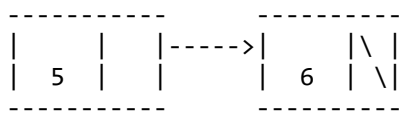
Name the program: `ShowPostEvalXX.cpp`, where XX are your initials.

Required Problem- Comprehensive.

4 (**). Write a menu-driven program to illustrate the use of a linked list. The entries will only consist of integer number keys (values). The program should implement the following options in order:

- `Insert`- insert a key into a list- not allowing duplicate integers.
- `Delete`- delete a key from the list.
- `Search`- finds or does not find a particular key in the list.
- `Print`- prints the list graphically in horizontal form. If the list is empty output- "`Nothing to print`".
- `Size`- count of all the keys in the list.
- `Sort`- sorts the keys in the list in ascending order.
- `Reverse`- reverses the order of the keys in the list.
- `Rotate`- moves the key at the front of the list to the end of the list. If the list has 0 or 1 elements it should have no effect on the list.
- `Shift`- rearranges the keys of a list by moving to the end of the list all values that are in odd number positions (indexes) and otherwise preserving list order.
- `Clear` - delete all the keys from the list. Output "`Empty List`".
- `Quit`- Quit the program.

The print list option should consist of a data field which is an integer and pointer to the next node. It should look similar to below:



Error check all menu inputs. A linked list data structure must be used. Output should be user friendly.

Name the program: `LListMenuXX.java` or `LListMenuXX.cpp`, where `XX` are your initials.

Extra Credit: Implement the following problem. See the 2436 Grading and Submission Guide Sheets for additional grading/submission information. Partial credit will be given. (10 points)

Given a string of parentheses, write a program that uses the vertical bar (`'|'`) and underscore (`'_'`) characters to connect the matching parentheses. Input from the keyboard a string with at most 30 characters and contains only the open and close parentheses and spaces. Assume there will be at least one parenthesis in each input string.

Output to the screen the string with exactly one space between parentheses. If there are any mismatched parentheses in the string, print the message `"Parentheses do not match!"` Otherwise, use the vertical bar and underscore characters to draw lines between matching parentheses. Each line must be at least one vertical bar in depth, but no line should be deeper than it needs to be. Follow the format illustrated in sample output below. The program must use a stack data structure. Finally, the program should ask if the user wants to run the program again (Check case). Refer to the sample output below.

Sample Run:

```
Enter a string up to 30 characters long: ( ()  ( ( )() )   )   ( )

( ( ) ( ( ) ( ) ) ) ( )
| |_| | |_| |_| | | |_|
|     |_________| |
|___________________|

Run again (Y/N): Y

Enter a string up to 30 characters long: ( ()

( ( )

Parentheses do not match!

Run again (Y/N): n
```

Name the program: `MatchParensXX.java` or `MatchParensXX.cpp`, where `XX` are your initials.