**Introduction to Computer Science – 150005**

**Homework Assignment #9**

**Dynamic Memory Allocation, Strings**

**Comments:**

1. Be careful on code readability and appearance (indentation).
2. Make sure you do exactly what is requested by each question.
3. In the case of incorrect input or error, the message ERROR should be given and you should allow the user to enter new input.
4. For each of the questions has functions you should define functions to improve program structure and readability.
5. At the end of each question, you need to provide a main program that shows the correctness of the functions that you wrote.
6. Use meaningful variable names
7. Comment each program, including a comment before the main program explaining its purpose and how it works, and a comment at the end of the program containing test runs with input and output.
8. Reminder: submit your own work, do not submit in pairs!

**Important notice: At times the automatic checker does not accept the delete statement, and therefore won’t assign a grade of 80. In this case, comment out the delete statement before resubmitting. (Note, comment out but don’t delete the statement, since a missing delete statement will not get full credit on the manual grading.)**

**Another important notice: In the following programs you will need to include the cstring library as follows:**

#include<cstring>  
#include<string>  
#pragma warning (disable:4996)

1. String encryption is a technique which substitutes the characters in a string according to some agreed upon convention that allows only someone who knows the convention to understand what is written.

The **azby** convention for encryption (like אתב"ש in Hebrew) works as follows. Whenever the letter a (the first letter in the alphabet) appears in the text, it is replaced with the letter z (the last letter). Similarly, the letter b is replaced with y, and so on (the letter z is replaced with an a). For example, the string “bamba” in azby is converted to the string “yznyz”.

* Write a function called **crypto** which receives a string no greater than 80 characters as input and returns the encrypted text.
* Write a main program that reads a string up to 80 characters long from the user after printing Enter a string: The program calls **crypto** and prints the encrypted string after the following message:  
    
  After crypto:

Example runs:

|  |  |
| --- | --- |
| Enter a string:  bamba  After crypto:  yznyz | Enter a string:  baA?ba  After crypto:  yzA?yz |

1. A string sentence is a string with sequences of characters (called words) of varying lengths. Words are separated by 1 or more spaces. The reverse of a string sentence is a string sentence in which all the words in the original string sentence are reversed. For example, if the word hello is in the original string sentence, then the word olleh will be in the reverse string sentence.

As an example, If the original string sentence was:

This is a silly88 sentence

Then the reverse string sentence is:

sihT si a 88yllis ecnetnes

Note: digits and spaces are preserved in the reverse statement.

Write a function called **reverse** which receives as input a string sentence not greater than 80 characters and returns its reverse string sentence.

Note: You should use the function getline to read in the input string from the user as follows:

**char sentence[80];**

**cout<<"enter a string: "<<endl;**

**cin.getline(sentence,80);**

Note: The inputs and outputs should be handled in the main program.

|  |
| --- |
| Enter a string:  This is a silly88 sentence After reverse: sihT si a 88yllis ecnetnes |

1. The Language Academy wants to create a computerized dictionary. The dictionary will store words in lexicographical order. Words will not appear more than once in the dictionary.

The Academy is interested in the following kinds of information:

* Does a given word exist in the dictionary
* Print all the words in the dictionary
* Print all words that start with a given letter

In addition, the Academy would like the information to be dynamic and be able to accommodate changes such as adding words and deleting words in the dictionary.

Write a program that contains the following information

* The dictionary will be stored in **lexicon** an array of char pointers. Each element in the array points to a word in the dictionary. The number of elements in the array is exactly the number of words that exist in the dictionary. The array will always be kept in sorted lexicographic order of the words its elements point to. Words should not appear more than once in the dictionary.
* Strings will be no greater than 80 characters.

The program contains the following functions:

1. newStr – Add a word to the dictionary.  
   Input to the function is the array **lexicon**, its size, and the word to be added. As a result of calling this function, the size of the array will increase by 1 and the new array will be in lexicographical order. Note, if the new word already appears in the dictionary, the array is unchanged.
2. delStr – Deletes a string from the dictionary  
   Input to the function is the array **lexicon**, its size, and the word to be deleted. As a result of calling this function, the size of the array will decrease by 1 and the new array will remain in lexicographical order. Note, if the new word already appears in the dictionary, the array is unchanged (with no error message and no request for new word to delete).
3. printAll – Prints the dictionary  
   Input to the function is the array **lexicon** and its size. The function prints all the words in the dictionary on one line separated by spaces. (Note, the system is essentially a 2-dimensional array.). In the event of an empty dictionary, nothing is printed.
4. searchStr – Search for a given string in the dictionary.  
   Input to the function is the array **lexicon**, its size, and the word to be searched. The function checks for the existence of the word in the dictionary and returns a pointer to the word or NULL otherwise.
5. printChar – Prints all words in the dictionary which start with a given character.  
   Input to the function is the array **lexicon**, its size, and the start character. The function prints all the words in the dictionary that start with the given character.

**Remember: You must free all unused dynamically allocated memory.**

The main program declares the **lexicon** array and then continuously loops handling user’s requests (a number between 0 and 5). For each request, the main program calls the appropriate function as described below. The program terminates on an exit request (input 5).

* On input 0, the main program prints Enter the word:, reads in a word to add, and calls the function newStr() and then printAll().
* On input 1, the main program prints Enter the word to delete:, reads in a word to delete, and calls the function delStr() and then printAll().
* On input 2, the main program prints Enter the word to search for:, reads in a word to search, calls the function searchStr(), and prints Found or   
  Not Found depending if the word exist in the dictionary or not.
* On input 3, the main program prints Enter the char:, reads in a character, and calls the function printChar().
* On input 4, the main program calls the function printAll().
* On input 5, the main program exits without further prints.

Comment: You can assume that input strings contain only lower-case characters.

|  |
| --- |
| Enter 0-5:  0  Enter the word:  good  good  Enter 0-5:  0  Enter the word:  hello  good hello  Enter 0-5:  0  Enter the word:  shalom  good hello shalom  Enter 0-5:  0  Enter the word:  today  good hello shalom today  Enter 0-5:  2  Enter the word to search for:  what  Not found  Enter 0-5:  2  Enter the word to search for:  hello  Found  Enter 0-5:  1  Enter the word to delete:  hello  good shalom today  Enter 0-5:  1  Enter the word to delete:  toda  good shalom today  Enter 0-5:  3  Enter the char:  s  shalom  Enter 0-5:  0  Enter the word:  toda  good shalom toda today  Enter 0-5:  3  Enter the char:  t  toda today  Enter 0-5:  5 |