CS217 Object Oriented Programming
Assignment No. 1
January 27, 2020

Deadline: February 7, 2020 at 5:00 PM

Attention

- Make sure that you read and understand each and every instruction. If you have any questions or comments you are encouraged to discuss with your colleagues and instructors on Piazza (`https://piazza.com/nu.edu.pk/spring2020/cs217/home`).
- Create each problem solution in a separate .cpp file, i.e. you must name the file containing solution of Q1 as 'q1.cpp', Q2 as 'q2.cpp' and Q3 as 'q3.cpp'.
  - Combine all your work in one .zip file.
  - Name the .zip file as ROLL-NUM_SECTION.zip (e.g. 19i-0001_B.zip).
  - Submit the .zip file on Google Classroom within the deadline.
- For all the problems, use " char * ", instead of string. The usage of string is strictly prohibited.
- **Start early otherwise you will struggle with the assignment.**
- **You must follow the submission instructions to the letter, as failing to do so will get you a zero in the assignment.**
- **All the submitted evaluation instruments (quizzes, assignments, lab work, exams, and the project) will be checked for plagiarism. If found plagiarized, both the involved parties will be awarded zero marks in the relevant evaluation instrument, all of the instruments, or even an F grade in the course.**

Q1: String Manipulation

Write the implementation of the following functions:

```
1 int Strlen(char *s1)
2 /*Returns the length of the string in number of characters.*/
3 {
4 }
```

```
1 char *Strcpy( char *s1, const char *s2 )
2 /*Copies string s2 into array s1. The value of s1 is returned.*/
3 {
4 }
```

```
1 char *Strncpy( char *s1, const char *s2, int n )
2 /*Copies at most n characters of string s2 into array s1.
3 The value of s1 is returned.*/
4 {
5 }
```

```
char *StrCat( char *s1, const char *s2 )
/*Appends string s2 to array s1.
The first character of s2 overwrites the terminating null character of s1.
The value of s1 is returned.*/
{
}
```

```
char *StrnCat( char *s1, const char *s2, int n )
/*Appends at most n characters of string s2 to array s1.
The first character of s2 overwrites the terminating null character of s1.
The value of s1 is returned.*/
{
}
```

```
int StrCmp( const char *s1, const char *s2 )
/*Compares the string s1 with the string s2.
The function returns 0, less than 0 or greater than 0 if s1 is equal to,
less than or greater than s2, respectively.*/
{
}
```

```
int StrnCmp( const char *s1, const char *s2, int n )
/*Compares up to n characters of the string s1 with the string s2.
The function returns 0, less than 0 or greater than 0 if s1 is equal to,
less than or greater than s2, respectively.*/
{
}
```

```
char **StrTok( char *s1, const char s2)
/*A call to StrTok breaks string s1 into ''tokens''
(logical pieces such as words in a line of text) separated by character
contained in char s2*/
{
}
```

```
int StrFind(char *s1, char *s2)
/*Searches the string s1 for the first occurrence of the string s2
and returns its starting index, if s2 not found returns -1.*/
{
}
```

```
char * SubStr (char *, int pos, int len)
/*This function returns a newly constructed string with its value initialized
to a copy of a substring of this variable.
The substring is the portion of the string that starts at character position
''pos'' and spans ''len'' characters
(or until the end of the string, whichever comes first).*/
{
}
```

Q2: Text Analysis

The availability of computers with string-manipulation capabilities has resulted in some rather interesting approaches to analyzing the writings of great authors. This exercise examines three methods for analyzing texts with a computer. **You have to use char * for the following exercises.**

1. Write a function that receives a string consisting of several lines of text and returns an array indicating the number of occurrences of each letter of the alphabet in the text. For example, the phrase "To be, or not to be: that is the question": contains one "a," two "b's," no "c's," and so on.

```
void countLetters(char *string, int *&array, int & size)
/*Parameters:
Input:
char * : a multiline string
Output:
int *: an array containing counts of each letter,
to be allocated in function
int : array size */
{
}
```

2. Write a function that receives a string consisting of several lines of text and returns an array indicating the number of one-letter words, two-letter words, three-letter words, and so on, appearing in the text. For example, the phrase "Whether this nobler in the mind to suffer" contains 2, 3, 4, etc. length words.

```
void countWordsBasedOnLength(char *string, int *&array/*to be allocated */,
int & size/*updated array size*/)
/*Parameters:
Input:
char * : a multi-line string
Output:
int *: an array containing counts of each different length words,
to be allocated in function
int : array size */
{
}
```

3. Write a function that receives a string consisting of several lines of text and returns arrays indicating unique words and the number of occurrences of each unique word in the text along with their size.

```
void countingUniqueWords(char *string, char **&uwords/*list of unique words;*/,
int *&array/*to be allocated */, int & size/*updated array size*/)
/*Parameters:
Input:
char * : a multiline string
Output:
char **: an array of unique words
int *: their counts
int : number of unique words*/
{
}
```

Q3: Matrix Operations

A matrix is a collection of values in the form of rows and columns. In this question, you are required to implement the following matrix functions using C++. **You can only use int** data type and DMA to create a matrix**

1. Matrix Multiplication

```
int** MatrixMul(int** MatrixA, int rowsA, int colsA,
        int** MatrixB, int rowsB, int colsB){
/* MatrixMul implements MatrixA x MatrixB
Both the matrices are stored using DMA.
Number of rows are columns of both the matrices
are available in rowsA, colsA, rowsB, and colsB variables.
*/
}
```

2. Matrix Addition

```
int **MatrixAdd(int** MatrixA, int rowsA, int colsA,
        int** MatrixB, int rowsB, int colsB){
/* MatrixAdd implements MatrixA + MatrixB
Both the matrices are stored using DMA.
Number of rows are columns of both the matrices
are available in rowsA, colsA, rowsB, and colsB variables.
*/
}
```

3. Matrix Subtraction

```
int **MatrixSub(int** MatrixA, int rowsA, int colsA,
        int** MatrixB, int rowsB, int colsB){
/* MatrixSub implements MatrixA - MatrixB
Both the matrices are stored using DMA.
Number of rows are columns of both the matrices
are available in rowsA, colsA, rowsB, and colsB variables.
*/
}
```

4. Matrix Transpose

```
int** MatrixTranspose(int** Matrix, int rows, int cols){
/* MatrixTranspose implements transpose of a Matrix
The matrix is stored using DMA.
Number of rows are columns of the matrix
are available in rows and cols.
*/
}
```

5. Matrix Rotate

```
int** MatrixRotate(int **Matrix, int rows, int cols){
/* MatrixRotate rotates a matrix 90 degree clockwise.
The matrix is stored using DMA.
Number of rows are columns of the matrix
```

```
11 22 33
44 55 66
77 88 99
```

into the matrix

```
77 44 11
88 55 22
99 66 33
```

Figure 1: Matrix rotation 90 degree clockwise

```
5 are available in rows and cols.
6 */
7 }
```

6. Matrix Determinant
   See: `https://people.richland.edu/james/lecture/m116/matrices/determinant.html`
   See: `https://www.math10.com/en/algebra/matrices/determinant.html`

```
1 int MatrixDet(int** Matrix, int rows, int cols){
2 /* MatrixDet implements determinant of a square Matrix
3 The matrix is stored using DMA.
4 Number of rows are columns of the matrix
5 are available in rows and cols.
6 */
7 }
```

7. Matrix Inverse
   See: `https://people.richland.edu/james/lecture/m116/matrices/inverses.html`

```
1 float** MatrixInverse(int **Matrix, int rows, int cols){
2 /* MatrixInverse implements inverse of a Matrix
3 The matrix is stored using DMA.
4 Number of rows are columns of the matrix
5 are available in rows and cols.
6 */
7 }
```