National University Of Computer and Emerging Science
Lahore Campus

Object Oriented Programming

Assignment # 1

Due Date: Sunday, April 11th by 11:55pm.

Dear students we will be using auto-grading tools, so failure to submit according to the below format would result in zero marks in the relevant evaluation instrument.

1. For each question in your assignment, make a separate cpp file e.g. for question 1, you have to make ROLL-NUM_SECTION_Q.cpp (20i-0001_A_Q1.cpp) and so on. Each file that you submit must contain your name, student-id, and assignment on top of the file in comments.
2. Combine all your work in one folder. The folder must contain only .cpp files (no binaries, no exe files etc.).
3. Run and test your program on a lab machine before submission.
4. Rename the folder as ROLL-NUM_SECTION (e.g. 20i-0001_A) and compress the folder as a zip file. (e.g. 20i-0001_B.zip). do not submit .rar file.
5. Submit the .zip file on Google Classroom within the deadline.
6. Submission other than Google classroom (e.g. email etc.) will not be accepted.
7. The student is solely responsible to check the final zip files for issues like corrupt file, virus in the file, mistakenly exe sent. If we cannot download the file from Google classroom due to any reason it will lead to zero marks in the assignment.
8. Displayed output should be well mannered and well presented. Use appropriate comments and indentation in your source code. Five Bonus marks will be awarded to well commented/indented code (for all questions).
9. Total Marks: 100
10. If there is a syntax error in code, zero marks will be awarded in that part of assignment.
11. Your code must be generic

Problems

1 String Manipulation

Write the implementation of the following functions:

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

2. **Strcpy**
   char∗ Strcpy (char∗ s1, const char∗ s2)
   /∗ Copies strings2 into array s1. The value of s1 is returned. ∗/
   {
   }

3. **Strncpy**
   char * Strncpy (char * s1 , const char * s2 , int n)
   /* Copy at most n characters of string s2 into array s1. The value of s1 is returned.
   {
   }

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

5. **StrnCat**
   char *StrnCat (char* s1 , const char* s2 , int n)
   /* Append 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.

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

7. **StrnCmp**
   int StrnCmp(const char* s1 , const char* s2 , i nt n)
   /* Compares up to n characters of the string s1 with the string s2. Returns 0, less than 0, or greater than 0 if s1 is equal to, less than, or greater than s2 respectively.. The value of s1 is returned.
   {
   }

8. **StrTok**
   char ** StrTok (char* s1 , const char s2)
   /* break string s1 into ”tokens” (logical pieces such as words in a line of text) separated by character contained in char s2.
   {
   }

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

10. **SubStr**
    char* SubStr (char * , int pos , int len)
    /* 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).
    {
    }

---

2 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 the 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 the 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 */
{
}

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 ,i n t ** 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 (float ** 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 degrees clockwise.
   The matrix is stored using DMA.
   Number of rows are columns of the matrix
   are available in rows and cols.
   */
   }

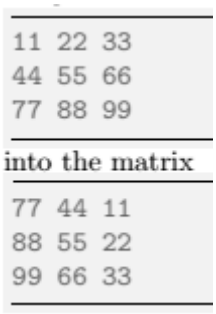

Figure 1: Matrix Rotation

6. **Matrix Determinant**

   See: https://people.richland.edu/james/lecture/m116/matrices/determinant.html
   See: https://www.math10.com/en/algebra/matrices/determinant.html
   int MatrixDet (int ** Matrix , int rows , int cols)
   {
   /* MatrixDet implements determinant of a square Matrix
   The matrix is stored using DMA.
   Number of rows are columns of the matrix
   are available in rows and cols .
   /
   }

7. **Matrix Inverse**
   See: https://people.richland.edu/james/lecture/m116/matrices/inverses.html
   float ** MatrixInverse (int **Matrix , int rows , int cols)
   {
   /* MatrixInverse implements inverse of a Matrix
   The matrix is stored using DMA.
   Number of rows are columns of the matrix
   are available in rows and cols .
   */
   }

Honor Policy

This task is a learning opportunity that will be evaluated based on your ability.**Plagiarized reports or code will get a zero**. If in doubt, ask the course instructor.