# Q1: String Manipulation

Write the implementation of the following functions:

1. in t Str l e n ( char ∗ s 1 )
2. /∗ Returns the le ngth o f the s t r i n g in number o f c h a r a c te r s . ∗/ 3 *{*

4 *}*

1. char ∗ Strcpy ( char ∗s1 , const char ∗ s 2 )
2. /∗ Copies s t r i n g s 2 in to array s 1 . The value o f s 1 i s returned . ∗/ 3 *{*

4 *}*

1. char ∗ Strncpy ( char ∗s1 , const char ∗s2 , in t n )
2. /∗ Copies at most n c h a r a c te r s o f s t r i n g s 2 in to array s 1 .
3. The value o f s 1 i s returned . ∗/ 4 *{*

5 *}*

1. char ∗ StrCat ( char ∗s1 , const char ∗ s 2 )
2. /∗Appends s t r i n g s 2 to array s 1 .
3. The f i r s t c h a ra c te r o f s 2 o ve rw r i te s the terminating n u l l c h a ra c te r o f s 1 .
4. The value o f s 1 i s returned . ∗/ 5 *{*

6 *}*

1. char ∗ StrnCat ( char ∗s1 , const char ∗s2 , in t n )
2. /∗Appends at most n c h a r a c te r s o f s t r i n g s 2 to array s 1 .
3. The f i r s t c h a ra c te r o f s 2 o ve rw r i te s the terminating n u l l c h a ra c te r o f s 1 .
4. The value o f s 1 i s returned . ∗/ 5 *{*

6 *}*

1. in t StrCmp( const char ∗s1 , const char ∗ s 2 )
2. /∗Compares the s t r i n g s 1 with the s t r i n g s 2 .
3. The f u n c t io n re tu rn s 0 , l e s s than 0 or g r e a te r than 0 i f s 1 i s equal to ,
4. l e s s than or g r e a te r than s2 , r e s p e c t i v e l y . ∗/ 5 *{*

6 *}*

1. in t StrnCmp( const char ∗s1 , const char ∗s2 , in t n )
2. /∗Compares up to n c h a r a c te r s o f the s t r i n g s 1 with the s t r i n g s 2 .
3. The f u n c t io n re tu rn s 0 , l e s s than 0 or g r e a te r than 0 i f s 1 i s equal to ,
4. l e s s than or g r e a te r than s2 , r e s p e c t i v e l y . ∗/ 5 *{*

6 *}*

1. char ∗∗ StrTok ( char ∗s1 , const char s 2 )
2. /∗A c a l l to StrTok breaks s t r i n g s 1 in to ' ' tokens ' '
3. ( l o g i c a l p i e c e s such as words in a l i n e o f te xt ) separated by c h a ra c te r
4. contained in char s 2 ∗/ 5 *{*

6 *}*

1. in t Str Find ( char ∗s1 , char ∗ s 2 )
2. /∗ Searches the s t r i n g s 1 f o r the f i r s t o ccurre nce o f the s t r i n g s 2
3. and re tu rn s i t s s t a r t i n g index , i f s 2 not found re tu rn s *−*1. ∗/ 4 *{*

5 *}*

1. char ∗ SubStr ( char ∗ , in t pos , in t le n )
2. /∗ This f u n c t io n re tu rn s a newly co ns tructe d s t r i n g with i t s value i n i t i a l i z e d
3. to a copy o f a s u b s t r in g o f t h i s v a r i a b l e .
4. The s u b s t r in g i s the po rtio n o f the s t r i n g that s t a r t s at c h a ra c te r p o s i t i o n
5. ' ' pos ' ' and spans ' ' le n ' ' c h a r a c te r s
6. ( or u n t i l the end o f the s trin g , whichever comes f i r s t ) . ∗/ 7 *{*

8 *}*

# 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.

1. void co unt Le tte rs ( char ∗ s trin g , in t ∗&array , in t & s i z e )
2. /∗ Parameters :
3. Input :
4. char ∗ : a m u l t i l i n e s t r i n g
5. Output :
6. in t ∗ : an array c o n ta in in g counts o f each l e t t e r ,
7. to be a l l o c a t e d in f u n c t io n
8. in t : array s i z e ∗/ 9 *{*

10 *}*

* 1. 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.

1. void countWordsBasedOnLength ( char ∗ s trin g , in t ∗&array /∗ to be a l l o c a t e d ∗/ ,
2. in t & s i z e /∗ updated array s i z e ∗/ )
3. /∗ Parameters :
4. Input :
5. char ∗ : a multi l i n e s t r i n g

*−*

1. Output :
2. in t ∗ : an array c o n ta in in g counts o f each d i f f e r e n t le ngth words ,
3. to be a l l o c a t e d in f u n c t io n
4. in t : array s i z e ∗/ 10 *{*

11 *}*

* 1. 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.

1. void countingUniqueWords ( char ∗ s trin g , char ∗∗&uwords /∗ l i s t o f unique words ; ∗/ ,
2. in t ∗&array /∗ to be a l l o c a t e d ∗/ , in t & s i z e /∗ updated array s i z e ∗/ )
3. /∗ Parameters :
4. Input :
5. char ∗ : a m u l t i l i n e s t r i n g
6. Output :
7. char ∗ ∗ : an array o f unique words
8. in t ∗ : t h e i r counts
9. in t : number o f unique words ∗/ 10 *{*

11 *}*

# 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
2. in t ∗∗ MatrixMul ( in t ∗∗ MatrixA , in t rowsA , in t cols A ,
3. in t ∗∗ MatrixB , in t rowsB , in t cols B )

*{*

1. /∗ MatrixMul implements MatrixA x MatrixB
2. Both the matrice s are s to re d using DMA.
3. Number o f rows are columns o f both the matrice s
4. are a v a i l a b l e in rowsA , cols A , rowsB , and cols B v a r i a b l e s . 7 ∗/

8 *}*

1. Matrix Addition
2. in t ∗∗MatrixAdd ( in t ∗∗ MatrixA , in t rowsA , in t cols A ,
3. in t ∗∗ MatrixB , in t rowsB , in t cols B )

*{*

1. /∗ MatrixAdd implements MatrixA + MatrixB
2. Both the matrice s are s to re d using DMA.
3. Number o f rows are columns o f both the matrice s
4. are a v a i l a b l e in rowsA , cols A , rowsB , and cols B v a r i a b l e s . 7 ∗/

8 *}*

1. Matrix Subtraction
2. in t ∗∗ MatrixSub ( in t ∗∗ MatrixA , in t rowsA , in t cols A ,
3. in t ∗∗ MatrixB , in t rowsB , in t cols B )

*{*

1. /∗ MatrixSub implements MatrixA MatrixB

*−*

1. Both the matrice s are s to re d using DMA.
2. Number o f rows are columns o f both the matrice s
3. are a v a i l a b l e in rowsA , cols A , rowsB , and cols B v a r i a b l e s . 7 ∗/

8 *}*

1. Matrix Transpose

*{*

1. in t ∗∗ MatrixTranspose ( in t ∗∗ Matrix , in t rows , in t c o l s )
2. /∗ MatrixTranspose implements trans po s e o f a Matrix
3. The matrix i s s to re d using DMA.
4. Number o f rows are columns o f the matrix
5. are a v a i l a b l e in rows and c o l s . 6 ∗/

7 *}*

1. Matrix Rotate

*{*

1. in t ∗∗ MatrixRotate ( in t ∗∗ Matrix , in t rows , in t c o l s )
2. /∗ MatrixRotate r o t a t e s a matrix 90 degree c lo c kw i s e .
3. The matrix i s s to re d using DMA.
4. Number o f rows are columns o f the matrix

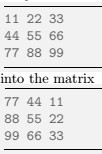


Figure 1: Matrix rotation 90 degree clockwise

1. are a v a i l a b l e in rows and c o l s . 6 ∗/

7 *}*

1. Matrix Determinant

See: <https://people.richland.edu/james/lecture/m116/matrices/determinant.html>

See: <https://www.math10.com/en/algebra/matrices/determinant.html>

*{*

1. in t MatrixDet ( in t ∗∗ Matrix , in t rows , in t c o l s )
2. /∗ MatrixDet implements determinant o f a square Matrix
3. The matrix i s s to re d using DMA.
4. Number o f rows are columns o f the matrix
5. are a v a i l a b l e in rows and c o l s . 6 ∗/

7 *}*

1. Matrix Inverse

See: <https://people.richland.edu/james/lecture/m116/matrices/inverses.html>

*{*

1. f l o a t ∗∗ Matrix Inverse ( in t ∗∗ Matrix , in t rows , in t c o l s )
2. /∗ Matrix Inverse implements in v e r s e o f a Matrix
3. The matrix i s s to re d using DMA.
4. Number o f rows are columns o f the matrix
5. are a v a i l a b l e in rows and c o l s . 6 ∗/

7 *}*