CCPROG2 Mock Exam 1 (Closed Notes, Individual, Written Exercise)

Part I. Arrays. Trace the following program and show the complete screen output. (10 pts)

```
#include <stdio.h>

#define MAX 10

void unknown ( char A[], char B[] )
{
   int m, j, f;
   char k;

   for (m=0; m < MAX; m++)
   {
      j = f = 0;
      k = A[m];
      while (j<MAX && !f)
      {
         if ( B[j] == k )
             f = 1;
         j++;
      }
      if (f)
        printf(“%c*”, k);
   }
}

void strange (char A[], int B[])
{
   int n, j;

   for (j=0; j<MAX/2; j++)
   {
       n = B[j];
       printf(“%c”, A[n]);
   }
}

int main()
{  char A[MAX]={‘m’, ’e’, ’r’, ’y’, ’*’, ’x’, ’m’, ’a’, ’s’, ’!’ };
   char B[MAX]={‘n’, ’e’, ’w’, ’*’, ’y’, ’e’, ’a’, ’r’, ’:’, ’)’ };
   int  C[5] = { 3, 1, 6, 7, 8 };

   unknown(A,B);
   printf(“\n”);
   strange(A,C);
   return 0;
}
```

Part II. **ANALYSIS .** (10 pts) What will be the screen output of the following statements, given the values of a two-dimensional array (called aTable):

aTable

	0	**1**
0	9	3
1	3	9
2	4	2

aMatrix

	0	**1**	**2**
0	0	1	2
1	1	3	2

```
#include <stdio.h>

#define ROW 3
#define COL 2


void performManipulation (int aTable[ROW][COL], int aMatrix[COL][ROW])
{ int nR, nS;

  for (nR = 0; nR < ROW; nR++)
    for (nS = 0; nS < COL; nS++)
      aTable[nR][nS] += aMatrix[nS][nR];
}

int main( )
{ int aTable[ROW][COL];
  int aMatrix[COL][ROW];
  int nR, nC;

  /* Assume that values have been assigned to the arrays. */
  performManipulation(aTable, aMatrix);

  for (nR = 0; nR < ROW; nR++)
  {
     for (nC = 0; nC < COL; nC++)
        printf (“%d_”, aTable[nR][nC]);
      printf (“\n”);
  }
  return 0;
}
```

Part III. (15 pts) Answer the following requirements using **only 1 statement**. For function calls, you can only use printf(), scanf(), strcpy(), strcmp(), strcat(), strlen(), and the user-defined function copy() we discussed in class (but rather than integer arrays, we have character arrays). Recall the copy() function with the prototype below, where d is the address of the destination, s is the address of the source, and nElem is the number of elements to be copied starting from the source address (indicated in s) to the destination address (indicated in d).

```
      void copy (char * d, char * s, int nElem);
```

Note that you can use however many function calls in your answer as long as it is considered as 1 statement in C. The following is an example of 1 statement in C: printf(“ABC length = %d and concatenating contents of str with it results to %s with a length of %d\n”, strlen(“ABC”), str, strlen(strcat(str,”ABC”)));

Use only the following variables and assume they have been initialized with some values:

```
      char strA[10], strB[10];
```

1. Display the result of comparing strings stored in strA and strB.

2. Copy the contents of string strA to strB. Use strcpy().
3. Copy the contents of string strA to strB. Use copy().
4. Copy all contents of array strA to strB. Use copy().
5. Copy 5 characters of array strA to strB, starting with strA’s 3rd element. Use function copy().
6. Copy the contents of string strA to strB, starting with strA’s 3rd character, only if strA contains at least 3 characters (count excludes the null byte ‘\0’). Use strcpy().
7. Concatenate strA to strB only if strB contains a larger value than strA.
8. Concatenate strA to strB only if strB is longer than strA.
9. Concatenate into strA the contents of string strA and the last 2 characters in string strB.

Part IV. (20 pts) Write a function **determineMajority()** that determines whether a majority element exists in a given array of integers. A majority element in an **array A of size N** is an element that appears **more than N/2** times. Thus, there is at most one such element. For example, consider the following array:

index	0	1	2	3	4	5	6	7
element	3	4	4	4	2	4	2	4

Example #1

It has a majority element equal to 4. However, for the following example, it does not have a majority element.

index	0	1	2	3	4	5	6	7
element	3	3	4	4	2	4	2	4

Example #2

In your function, take note that it should **return an index of an element from the array that holds a majority element**, **not the value** of the majority element itself. In this case, your function should return the **first occurrence** of the majority element in the given array (given Example #1, the function determineMajority() will return 1). If a majority element **does not exist**, then the function will return **-1**.

Assume the following declarations and functions are available for your use. [No need to code these.] Assume that the array a is already fully-filled with integer values.

```
#define MAX 100

void sort(int a[])
{     /* Sorts the array elements in ascending order. */ }

void copyArray(int aSource[], int aDestination[])
{  /* Copies all the elements from aSource array to aDestination array. */ }

int getIndex(int a[], int nKey)
{     /* If nKey value is found in the array a, then the function returns the
nKey value’s corresponding index. Otherwise, the function returns -1. */ }
```

Your task is to implement the following function. You may create additional functions provided that you give the corresponding operations/computations/code.

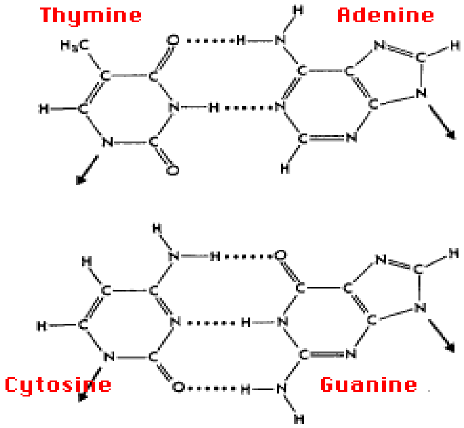
```
int determineMajority(int a[MAX])
```

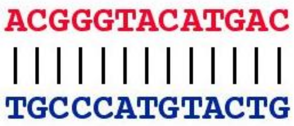
```
{     /* Supply your code and declarations for this function. */ }
```

Part V. STRING FUNCTIONS, ARRAY of STRINGS [20 points].

In the study of genetics, one of the most important topics involves finding a DNA's base pair. A DNA nucleotide is made of a molecule of sugar, a molecule of phosphoric acid, and a molecule called a base. The base of the DNA is represented by a string of characters: 'A' for adenine, 'G' for guanine, 'T' for thymine, and 'C' for cytosine.

A **base pair** refers to two bases which form a part of the DNA. In finding the base pair of a DNA, Adenine is always paired with Thymine while Cytosine pairs with Guanine (the process is demonstrated in Figure 1). Observe how in the Figure 2, a DNA sample written as "ACGGGTACATGAC" is given a base pair written as "TGCCCATGTACTG" with the rules stated.





For this problem, you are going to create two functions that are part of a C program catered to a study on genetics. These two functions are called **`normalize()`** and **`basepair()`**.

normalize() takes in two parameters: strIn and strOut, both of which are strings. **normalize()** seeks to remove unnecessary characters not ‘A’, ‘G’, ‘T’, or ‘C’ in strIn (which is assumed to contain only uppercase characters) and places the resulting string in strOut. For example, the function call(s):

normalize(‶AMLXKDGTCGTJELA″,strOut) will result in the string “AGTCGTA” being stored in strOut.

normalize(‶GTCZDSA″,strOut) will result in the string “GTCA” being stored in strOut.

normalize(‶PPPPAPPGTPP″,strOut) will result in the string “AGT” being stored in strOut.

basepair() takes in two parameters: strIn and strOut, both of which are strings. **basepair()** aims to place the resulting base pair of strIn (which is assumed to have been normalized) in strOut. For example, the function call(s):

basepair(‶AGTCGTA″,strOut) will result in the string “TCAGCAT” being stored in strOut.

basepair(‶GTCA″,strOut) will result in the string “CAGT” being stored in strOut.

basepair(‶TAGCA″,strOut) will result in the string “ATCGT” being stored in strOut.

Refer to the following skeleton for your string declarations for strIn and strOut along with the function headers for **basepair()** and **normalize()** respectively

```
typedef char Str50[51];

void basepair (Str50 strIn,Str50 strOut)
{             /* Your code here */
}

void normalize(Str50 strIn,Str50 strOut)
{             /* Your code here */
}
```