DO NOT OPEN EXAM UNTIL INSTRUCTED TO BEGIN

Name: _____ **UIN**: _____ **Section**: _____

- **This exam has X questions.**
- **We scan the front pages of exams and cut off staples; therefore,**
  - **Do not write on the back of exam pages.** *Request blank paper if you need it. Write your name and the question number on it and attach to the back of the exam.*
  - **Leave a quarter inch margin around the edges of the page.** *Otherwise, what you write may get chopped off during scanning.*
- **This is a closed book, closed note exam.** *Do not use any notes, books, or computing technology including smart watches. Do not confer with any other person.*
- **Partial credit will be given.** *Do things to make your thinking and process visible, like showing the values of variables as they change.*
- **Grading will be based on correctness, clarity, and neatness.**
- **Suggestion: Read the entire exam before you begin work on any problem.** *Budget your time wisely, according to point distribution.*
- **Make sure you have an ID.** *Your exam will not be graded until identity is confirmed.*
- **This is a 50-minute exam. When the proctor states that the exam period has concluded, stop writing immediately; you will receive a score of *zero* if you continue writing.**
  - **Make sure you've written your name on each page prior to the end of the exam; you will not be allowed to do this once time has expired.**

Please sign to acknowledge the statements above and affirm the Texas A&M University academic integrity statement below:
"On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work. In particular, I certify that I have not received or given any assistance that is contrary to the letter or the spirit of the guidelines for this exam."

Signature: _____

Note:

- The number of questions on this practice exam is not an indicative of the number of questions on the exam. The expected difficulty of the questions on this practice exam is greater than or equal to the expected difficulty of the questions on the exam.
- Topics covered:
  - 2-dimensional arrays
  - Dynamically-allocated arrays and 2d-arrays
  - Functions with arguments by reference and by value
  - deep and shallow copy
  - Recursive functions

1. Write a function with declaration

   `double pathLength(double** distance, int n, int* path, int m)`

   where

   - `distance` is a $n \times n$ 2d-array such that position `distance[i][j]` stores the road distance in miles from city i to city j;
   - `path` is an integer array with m elements that stores a sequence of cities visited in a trip, i.e., $0 \leq path[i] < n$ for all i such that $0 \leq i < m$
   - n and m are greater than zero.

   The function should return the length in miles of the path.

   For example, for $n = 5$,

   distance

0.0	30.0	10.0	70.0	10.0
30.0	0	45	100.0	50
10	45.0	0	85.0	20
70.0	100.0	85.0	0	100
10.0	50.0	20.0	100.0	0

   and $m = 3$, path

0	1	2

   pathLength(distance, n, path, m) returns $30 + 45 = 75$.

   For same n and distance, but with $m = 6$ and path

0	1	0	3	2	0

   pathLength(distance, n, path, m) returns $30 + 30 + 70 + 85 + 10 = 225$

2. Write a function with declaration

   `void avgMatrix (int** inArray, int m, int n, int** outArray)`

   that gets a 2d-array `inArray` with $m \geq 1$ rows and $n \geq 1$ columns with double elements and calculates `outArray` as the average matrix, i.e., `outArray` is an $m \times n$ matrix where each element is obtained by the average of the (at most 8) neighbor elements in `inArray`.

   Example:

   For the 3×4 matrix:

$$\begin{pmatrix} 0.5 & 2.0 & 1.2 & 3.0 \\ -1.0 & 1.5 & 3.0 & 2.4 \\ 0.0 & 1.0 & 1.5 & 2.0 \end{pmatrix}$$

   the `outArray` would be:

$$\begin{pmatrix} 0.833 & 1.04 & 2.38 & 2.2 \\ 1.0 & 1.025 & 1.8 & 2.14 \\ 0.5 & 1.0 & 1.98 & 2.3 \end{pmatrix}$$

3. Write a function with declaration `void minusOddColumn(int** mat, int n)`

   where $n \geq 1$ and `mat` is an $n \times n$ 2d-array of non-negative integers.

   The function should find the column in `mat` with more odd numbers and replace all of its elements with -1. If multiple columns have the highest presence of odd numbers, the first of them (i.e. the one with the lowest index) should be chosen to have its elements replaced by -1.

   For example, for $n = 5$ and `mat`

0	30	10	70	10
30	0	45	100	50
10	45	0	85	20
70	100	85	0	100
10	50	20	100.0	0

   the function will result in `mat`

0	30	-1	70	10
30	0	-1	100	50
10	45	-1	85	20
70	100	-1	0	100
10	50	-1	100.0	0

   For $n = 2$ and `mat`

0	30
30	0

   the function will result in `mat`

-1	30
-1	0

4. For this question, you will write two functions:

   - write a function that given an $n \times n$ matrix, it returns how many rows of the matrix have all elements positive (≥ 0):
     `int countPositiveRows(int** mat, int n)`
   - write a function that given an $n \times n$ matrix `inMatrix`, it returns the pointer to a new dynamically allocated matrix containing only the rows in `inMatrix` where all elements are positive. It will also return the number of rows in the new matrix:
     `int** newMatrix(int** inMatrix, int n, int& newN)`

     For example, for $n = 4$ and inMatrix

0	-1	10	70	10
30	4	45	100	-50
10	1	0	85	20
70	1	85	0	100
10	-1	-20	100.0	0

     `countPositiveRows(inMatrix,n)` will return 2 and

     `int* ptrMatrix = newMatrix(inMatrix, n, newN)` will result in the variable newN (passed to the function by reference) receiving 2 and the variable ptrMatrixg pointin to the matrix containing the elements

10	1	0	85	20
70	1	85	0	100

     You may find it helpful to use `countPositiveRows` in your solution, but you are not required to.

5. Consider the following sequences of statements:

Sequence A	*Sequence B*

`int *a = new int(11);`
 	
`int *b = new int(*a);`
	`int *x = new int(7);`
	
	`int *y = x;`

(a) Which sequence of statements results in a *deep copy*?

- ◯ Sequence A
- ◯ Sequence B
- ◯ Neither

(b) Which sequence of statements results in a *shallow copy*?

- ◯ Sequence A
- ◯ Sequence B
- ◯ Neither

(c) Briefly describe a shallow copy. Analogies, diagrams, or examples are acceptable as long as the essence of a shallow copy is clearly communicated.

(d) Briefly describe a deep copy. Analogies, diagrams, or examples are acceptable as long as the essence of a deep copy is clearly communicated.

6. Consider the following function definitions:

Definition A

```
void allocate(int *arr)
{
    if(arr != nullptr)
        delete[] arr;
    arr = new int[4];
}
```

Definition B

```
void allocate(int *&arr)
{
    if(arr != nullptr)
        delete[] arr;
    arr = new int[4];
}
```

Given the following main function:

```
int main()
{
    int *ptr = new int[4];
    allocate(ptr);
    delete[] ptr;
}
```

(a) Draw a memory diagram when from `main` through the call to `allocate()` that uses *Definition A*.

(b) Draw a memory diagram when from `main` through the call to `allocate()` that uses *Definition B*.

(c) What is the difference in passing the array by reference (`int *&arr`) vs. by value (`int *arr`) to `allocate`?

(d) Which definition(s) (if any) causes a memory leak?

- ◯ Definition A
- ◯ Definition B
- ◯ Both
- ◯ Neither

(e) Which definition(s) (if any) causes a dangling pointer?

- ◯ Definition A
- ◯ Definition B
- ◯ Both
- ◯ Neither

(f) Which definition(s) (if any) causes `delete` to be called twice on the same address?

- ◯ Definition A
- ◯ Definition B
- ◯ Both
- ◯ Neither

7. Considerations of argument passing.

(a) When passing an argument to a function parameter, how is passing that argument by constant reference different than passing it by reference?

(b) When passing an argument to a function parameter, when should you pass the argument as a reference (and thus not by value or as a constant reference)?

8. Answer the following parts to this question with respect to this code:

```
1  int no_rows = 2;
2  int no_cols = 4;
3  int arr[no_rows][no_cols];
4  int i = 0, j = 0;
5  while (i < no_rows)
6  {
7      arr[i][j] = i * no_cols + j;
8      j += 1;
9      if (j == no_cols) {
10         i += 1;
11     }
12     if (j % no_cols == 0)
13         j = 0;
14 }
```

(a) What is the composition of the two-dimensional array `arr` after the code has been executed? Position `(0,0)` of `arr` is at row 0, column 0 in the table below.

	0	1	2	3
0				
1				

(b) In the space provided below, write C++ code structured as a nested for-statement that performs the same assignments to the two-dimensional array `arr` as does the while-statement above.

9. At your job, you are creating a library. A co-worker brought this test code to you. They expect that the output would be "12 12 12 12 12". However, they are getting "Empty List".

   (a) Describe why the error is occurring.

   (b) Explain how to fix the code.

```
#include <iostream>
using namespace std;

void increaseArray(int* array, int size, int value) {
  int newSize = size + 5;
  if (size==0) {
    size = 5;
  }
  int* newArray = new int[newSize];
  for (int i=0; i<size; ++i) {
    newArray[i] = array[i];
  }
  for (int j=size; j<newSize; ++j) {
    newArray[j] = value;
  }
  delete [] array;
  size = newSize;
  array = newArray;
  newArray = nullptr;
}

void printArray(int* array, int size) {
  if (size==0 || array==nullptr) {
    cout << "Empty List" << endl;
  }
  else {
    for (int i=0; i<size; ++i) {
      cout << array[i] << " ";
    }
    cout << endl;
  }
}

int main() { // test program
  int* testArray = nullptr;
  int testSize = 0;
  int testValue = 12;
  increaseArray(testArray, testSize, testValue);
  printArray(testArray, testSize);
}
```

10. What is the output of the following program?

    - Draw a memory diagram to help you follow what's going on.

```
1  #include <iostream>
2  using namespace std;
3
4  int f1(int, int&);
5  int f2(int&);
6
7  int main() {
8     int a = 2;
9     int b = 9;
10    int c = f1(a,b);
11    cout << "a␣=␣" << a << endl;
12    cout << "b␣=␣" << b << endl;
13    cout << "c␣=␣" << c;
14 }
15
16 int f1(int r, int& s) {
17    r *= 3;
18    s /= 4;
19    int temp = r + f2(s);
20    return temp;
21 }
22
23 int f2(int& p) {
24    p = 12;
25    return p;
26 }
```

11. The question description is on the next page; write your response that question here.

functions.h

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H
bool IsEvenGeneric(char *, char *);
#endif
```

functions.cpp

Question to be answered above is written on the next page

Recall that character arrays (C-style strings) are initialized with one more character that it would appear to have; this is because character arrays are terminated by the null character `'\0'`, which has a value of zero. For example, consider the following representations:

`char bohr[] = "Bohr";` →

B	o	h	r	\0

Suppose there is the function `IsEvenV()` which takes a c-style string as an input, and returns whether the number of vowels it contains is even or not.

For example, `IsEvenV("family of functions")` returns `true` because there are six vowels.

Notice how `isEvenV()` only considers the vowels 'a', 'e', 'i', 'o', and 'u'. Now, I have a friend that counts 'y' among the vowels. I decided that rather than write a second version of `IsEvenV` just for her, I write a generic function, `isEvenGeneric()` that takes two inputs:

- a c-style string `s` storing the string to evaluate
- a c-style string storing the set of characters to check.

This function `isEvenGeneric()` is defined so that it can specify which characters to check for. Given `char s[] = "Some phrase.";`:

- `IsEvenGeneric(s,"aeiou")` checks for vowels not including 'y'
- `IsEvenGeneric(s,"aeiouy")` checks for vowels including 'y'
- `IsEvenGeneric(s,"cdfk")` checks for the characters 'c', 'd', 'f', and 'k'

(a) Given the header file `functions.h`, write the corresponding source file `functions.cpp` containing the necessary `#include`s and the function defintion for `IsEvenGeneric`. `IsEvenGeneric` has the following return type and signature: `bool IsEvenGeneric(char * s, char * v)` and implements the behavior described above. Keep in mind:

- Your function should return a boolean for whether `s` contains an even number of characters listed in `v`.
- `s` and `v` are pointers to the first character in the string of each argument respectively. Use the null terminated character `'\0'` to help you traverse each array.

12. Consider the recursive function below:

```
double f(double x, double y)
{
  if (x >= y) {
   return (x+y)/2;
  }
  double p1 = f(x+2, y - 1);
  double p2 = f(x+1, y - 2);
  return f(p1, p2);
}
```

What is the return value of `f(1, 10)`? How would you write this function without recursion?

13. Write a recursive function with the declaration:

`int countEqual(int* numbers, int n, int x)`

that has as parameters an array `numbers` with $n \geq 0$ elements, and an integer x, and returns how many times x appears in the array.

14. Write a recursive function with the declaration:

`double dist(double* u, double *v, int n)`

that gets two double precision arrays with $n \geq 1$ elements and returns the value:

$$\sqrt{(u[0]-v[0])^2+(u[1]-v[1])^2+\cdots+(u[n-1]-v[n-1])^2}$$

15. Write a recursive function with the declaration:

`double continuousFraction(double* x, int n)`

that has as parameters an array x with $n/geq1$ double precision numbers and returns the value:

$$\left|1+\frac{x_{n-1}}{\left|1+\frac{x_{n-2}}{\ddots\left|1+\frac{x1}{|1+x_0|}\right|}\right|}\right|$$

when it is possible to calculate (no division by zero). If it is not possible to calculate this value the function must return -1.