ECE 220: Computer Systems and Programming

Past Exam

Name:_____

NetID:_____

Room:_____

Question 1 (25 points): _____

Question 2 (40 points): _____

Question 3 (35 points): _____

Total Score: _____

Problem 1 (25 points): Removing Duplicates from an Array

In this problem, you need to remove all the duplicates in an array of integers. Solving this problem involves removing duplicate data items in an sorted array, and returning the number of unique integers. If there are **m** distinct integer values in an array of size **n**, then those **m** values will appear in the first **m** locations.

Assume you have a sorted array of **n** integers in ascending order, **n** > 0. Your algorithm should track the location of the last unique element (int lastUnique) as it scans the array from 0 to n-1. The next unique element is copied into the next available position to the lastUnique slot. The following example illustrates how your algorithm should operate.

```
Example: arr[] = [1,2,2,3,4,5,6,6,8,9]

After each iteration the array and lastUnique value will look like:

[1, 2, 2, 3, 4, 5, 6, 6, 8, 9] i = 0 ; lastUnique = 0
[1, 2, 2, 3, 4, 5, 6, 6, 8, 9] i = 1 ; lastUnique = 1
[1, 2, 2, 3, 4, 5, 6, 6, 8, 9] i = 2 ; lastUnique = 1
[1, 2, 3, 3, 4, 5, 6, 6, 8, 9] i = 3 ; lastUnique = 2
[1, 2, 3, 4, 4, 5, 6, 6, 8, 9] i = 4 ; lastUnique = 3
[1, 2, 3, 4, 5, 5, 6, 6, 8, 9] i = 5 ; lastUnique = 4
[1, 2, 3, 4, 5, 6, 6, 6, 8, 9] i = 6 ; lastUnique = 5
[1, 2, 3, 4, 5, 6, 6, 6, 8, 9] i = 7 ; lastUnique = 5
[1, 2, 3, 4, 5, 6, 8, 6, 8, 9] i = 8 ; lastUnique = 6
[1, 2, 3, 4, 5, 6, 8, 9, 8, 9] i = 9 ; lastUnique = 7
```

```
int RemoveDuplicates(int *arr, int n){

   int lastUnique = 0;

   int i = 0;

   for (i = 0; i<n; i++) {

     if (_____________________________________________) {

          _____________________________________________

          _____________________________________________

          _____________________________________________

          _____________________________________________

     }

   }

   return _____________________________________________;

}



int main(){
    int arr[] = {3, 9, 12, 22, 10, 11, 22, 11, 90};

    int n = sizeof(arr)/sizeof(arr[0]);
    /* n is the number of elements in the array */

    int m;

    BubbleSort(arr, n);
    m = RemoveDuplicates(arr, n);
    printf("The array had %d unique elements\n", m);
}
```


Problem 2 (40 points): Recursive Minimum Path Sum

Given a *m* x *n* grid filled with positive integers numbers, find a path between the top left cell to bottom right cell which *minimizes* the sum of all numbers along its path. Note that you can only move either down or right at any point. The input grid is stored as a 2D array, in which all the elements are positive integers. You have to use ***recursion*** to implement the function **int** min_path_sum(**int** grid[][], **int** m, **int** n, **int** i, **int** j) that finds a minimal path sum from (0,0) to (i,j). If there are multiple minimal paths, you can return the sum of any one.

1	2	3
3	4	1
2	5	1

For the input matrix (m= 3, n=3) on the left, your program should return 8, because the path 1→2→3→1→1 minimizes the sum.

1	3	2	1
1	2	1	2
3	2	1	1

For the input matrix (m= 3, n=4) on the left, the program should return 7, because the path 1→1→2→1→1→1 minimizes the sum.

Algorithm sketch: to find the minimum path between the top left cell and bottom right cell, we will start from the end (bottom right) and traverse backwards.

1. First check if a current cell (i, j) is one of the base cases:
    a. If a current cell (i, j) = (0, 0), that is the path has reached top-left, then return the cost of the current cell (grid[i, j]).
    b. If a current cell is outside range, return a very large cost. For example, you can use the available constant INT_MAX.
2. Otherwise, for the recursive step:
    a. Compute the minimum path sum from (0, 0) to the current cell's left neighbor; call this *mps_l*.
    b. Compute the minimum path sum from (0, 0) to the current cell's top neighbor; call this *mps_u*.
    c. Return the cost of the current cell + min(*mps_l, mps_u*); where the min function returns the smaller of the two arguments.


```
#include <stdio.h>
int min(int a, int b);
int min_path_sum(int grid[][], int m, int n, int i, int j);

int main() {

    /* First test */
    int grid1[3][3]={1, 2, 3, 3, 4, 1, 2, 5, 1};
    int m = 3;
    int n = 3;
    int sum = min_path_sum(grid1, m, n, m-1, n-1);
    printf("min path sum of grid1 is %d \n", sum);

    /* Second test */
    int grid2[3][4]={1, 3, 2, 1, 1, 2, 1, 2, 3, 2, 1, 1};
    m = 3;
    n = 4;
    sum = min_path_sum(grid2, m, n, m-1, n-1);
    printf("min path sum of grid2 is %d \n", sum);
    return 0;
}
```

Part A (10 Points):

```
int min(int a, int b){
/*this function returns the smaller of the two inputs */
```

_____

_____

_____

_____

_____

```
}
```


Part B (30 Points):

```
int min_path_sum(int grid[][], int m, int n, int i, int j){
/* int grid[][]: an mxn grid
   int m, n: the dimension of the 2d grid
   int i, j: current cell index
   return: minimum path sum from (0, 0) to (i, j) */
/* Your Code Starts Here */
```

}

Problem3: Concepts (30 points)

Part A (15 points):
An engineer implemented a program that reverses contents of array using recursion. The correct output for array with 01234 should be 43210. However, the program does not work as intended. Please help this engineer fix the code.

```
#include <stdio.h>
void ReverseArray(int array[], int size) {
   int start = 0, end = size - 1, temp;
   if (start < end) {
      temp = array[start];
      array[start] = array[end];
      array[end] = temp;

      ReverseArray(array, size - 1);
   }
}

int main(){
   int array[5], i;

   for (i = 0; i<5; i++){
      array[i] = i;
   }

   ReverseArray(array, 5);
   printf("Reversed Array: ");
   for (i = 0; i<5; i++){
      printf("%d ", array[i]);
   }
         printf("\n");
   return 0;
}
```

What will be the output of the current program? (7 points)

Your answer:

How should this engineer fix the code to make the program correctly reverses the contents of array? Hint: ReverseArray(array, size-1) is wrong. Please write the entire syntax for the line. No explanation required. (8 points)

Your answer:

Part B (5 points)
A 2-D matrix with ***m*** rows and ***n*** columns (**m ≠ n**) is stored in a 1-D array in row-major order. How to access the element located in row ***i*** and column ***j*** ? (Assume both the row and column indices start from 0)

(a) array[j * m + n]

(b) array[j * n + m]

(c) array[i * m + j]

(d) array[i * n + j]

Your Answer:_____

Part C (5 points)
Which of the followings can randomly generate a number exactly between 7 and 17?

(a) rand() % 7 + 17

(b) rand() % 17 + 7

(c) rand() % 17 + 17

(d) rand() % 7 + 7

(e) rand() % 11 + 7

(f) rand() % 7 + 11

Your Answer:_____

Part D (10 points)
What is the size in bytes of the struct shown here (assume char is 8 bits and int is 32 bits)? (5 points)

```
struct StudentStruct
{
     char Name[100];
     int UIN;
};
```

Your Answer:_____

```
struct StudentStruct student_arr[100];
struct StudentStruct *ptr = student_arr;
ptr = ptr + 1;
```

Where is the pointer "ptr" pointing to now? (5 points)

(a) Name[1] in student_arr[0]

(b) Name[1] in student_arr[1]

(c) UIN in student_arr[0]

(d) Name[0] in student_arr[1]

Your Answer:_____

End of ECE 220 Past Exam 2