GNG1106 Homework Problem Set Solutions 4

The homework is to be submitted electronically before the due time. The submission must be a PDF file with a companion C source code (.c file) for the programming question. The solution to the programming question must be in the format of **LAB REPORT** and included in the PDF file (excepted for the C code, which is separately attached).

Question 1: Explain if the following program will have compiling error, potential run-time error or memory leak? and if not, what will its running result be?

```
#include<stdio.h>

int main ()
{
    int a[6]={1 ,2 , 3, 4, 5, 6};
    printf ("%d\n", *(a+4)* *(a+1));
    return 0;
}
```

No Errors!
Output: 10

Question 2: Explain if the following program will have compiling error, potential run-time error or memory leak? and if not, what will its running result be?

```
# include<stdio.h>

int main ()
{
    int a[4]={1, 2, 0, 3};
    int b[4]={4, 5, 7, 6};
    a=b;
    for (i=0; i <4; i++)
    printf ("%d\n", a[i]);
    return 0;
}
```

Errors!
i: undeclared and a=b cannot be assigned.

Question 3: Explain if the following program will have compiling error, potential run-time error or memory leak? and if not, what will its running result be?

```
# include<stdio.h>
```

```
int main ()
{   int a[]={1 , 2, 5, 4, 5, 6};
    int *p;
    p=(a+3);
    printf ("%d\n", *(p-1)*2);
    return 0;
}
```

No Errors!
Output: 10

Question 4: Explain if the following program will have compiling error, potential run-time error or memory leak? and if not, what will its running result be?

```
#include<stdio.h>

int main ()
{
    int i;
    int a[6]={1 , 2, 3, 4, 5, 6};
    int b[6]={ -1 , -2, -3, -4, -5, -6};
    *b=*a;
    for (i=0;i<3;i++)
        printf ("%d,%d,%d\n", a[i+1], *(a+i), *(b+i));
    return 0;
}
```

No Errors!
Output:
2,1,1
3,2,-2
4,3,-3

Question 5: In the code below, the function func requires an int array as an input, but the calling of the function in main passes in a pointer (to some allocated memory) and yet the code compiles and runs properly. In short answer, what do you learn from this?

```
#include<stdio.h>
#include<stdlib.h>

void func (int a[], int lengthOfA )
{
    int i;
    i=0;
    for (i=0;i< lengthOfA ; i++)
```

```
        a[i]=i;
}

int main ()
{
    int *b, i;
    b=( int *) malloc ( sizeof (int )*5);
    if (b)
    {
        for (i=0; i <5; i++)
            b[i]=10+ i;
        func (b, 5);
        for (i=0; i<5; i++)
            printf ("%d\n", b[i]);
        free(b);
    }
return 0;

}
```

An array and a pointer with allocated memory are passed format and are similar, they both can be used the same as an array.

Question 6: Explain what is wrong with the following program.

```
#include<stdio.h>
#include<stdlib.h>

void getIntegersFromUser (int N, int * userAnswers )
{
    int i;
    userAnswers =( int *) malloc (N* sizeof ( int ));
    if ( userAnswers )
    {
        printf (" Please enter %d integers \n", N);
        for (i=0;i<N; i++)
            scanf ("%d", userAnswers +i);
    }}
int main()
{
    int i,M=4;
    int *p;
    getIntegersFromUser (M,p);
    for (i=0;i <5;i++)
        printf ("%d\n", p[i]);
return 0;
}
```

Trying to print p[i] at index number 4 (i.e. p[4]) while it is out of range.

Question 7: If a program is written to call the following function, will it have compiling error, potential run-time error or memory leak? Provide one valid call for this function?

```
#include<stdio.h>
#include<stdlib.h>

void swap (int *a, int *b)
{
      int *tmp;
      tmp =( int *) malloc ( sizeof (int ));
      *tmp =*a;
      *a=*b;
      *b=*tmp;
}
```

No compiling error, potential run-time error or memory leak. The function swap is basically to switches the contents of a and b. A potential valid call can be:

```
int main()
{
int a=1,b=2;
printf("Before calling swap: a=%d, b=%d\n", a, b);
swap(&a,&b);
printf("After  calling swap: a=%d, b=%d\n", a, b);

return 0;
}
```

Question 8 (Programming): Write a function with name `getEvenNumbers`. The purpose of the function is to process an input array of `any` typed and returns the list of even numbers in the array. The function needs to be able to handle input arrays with arbitrary lengths. The return type of the function must be pointer to int, namely that the function returns a pointer pointing to the memory storing the list of even numbers. No global variables are allowed. Except for these requirements, you are free to design the function. You also need to write a testing program (namely a main function) to show your design and implementations of this function are correct. Your testing program needs to do the following: first it asks the user to enter a positive integer N, and then it generates a length-N array of random integers; then calls your function `getEvenNumbers` to get the list of even numbers in this array, and finally prints the list.

Steps from 1-6 should be included. Here is the programming code:

```
#include <stdio.h>
#include <stdlib.h>

int *getarrayEven(int *, int , int *);

int main()
{
    int i, positiveInt;
    int *randomArray;
    int *randomEvenArray;
    int *sizeofArray;
    do {
        printf("Please enter a positive intger:\n");
        scanf("%d",&positiveInt);
            if (positiveInt<1)
                {    printf("ERROR! Not positive intger\n");
                 } } while (positiveInt<1);

    randomArray = malloc(positiveInt*sizeof(int));
    sizeofArray = malloc(sizeof(int));

    for(i=0; i<positiveInt; i++){
        randomArray[i] = rand()%1000;
    }
    randomEvenArray = getarrayEven(randomArray,positiveInt,sizeofArray);

    printf("The entire random array are: \n");
    for(i=0;i<positiveInt;i++){
        printf("%d\n",randomArray[i]);
    }
    printf("\nThe returned even elements are:\n");
    for(i=0;i<*sizeofArray;i++){
        printf("%d\n",randomEvenArray[i]);
    }   printf("\n"); }

int *getarrayEven(int *randomArray, int positiveInt, int *sizeofArray)
{
    int i, k=0;
    int *arrayEven;
    for(i=0;i<positiveInt;i++)
    {
        if(randomArray[i]%2==0)
            k++;    }
    arrayEven = malloc(k*sizeof(int));
    *sizeofArray = k;
    k=0;
    for(i=0;i<positiveInt;i++)
    {
        if(randomArray[i]%2==0)
        {
            arrayEven[k]=randomArray[i];
            k++;      }   }
    return arrayEven;
}
```

Question 9 (Programming): A matrix (i.e., a rectangular array of numbers) is called binary if its entries are either 0 or 1. A matrix is called sparse if most of its entries are 0. To represent a binary sparse matrix, instead of representing it as a two-dimensional array, a more efficient approach is to record the locations of all entries having value 1. The objective of this programming exercise is for you to develop such a representation for binary sparse matrices using pointer to pointers and manipulate such a representation. More specially, you are to write a program that asks the user to input a binary sparse matrix from the keyboard (by entering the locations of its 1's), the program then prints out the number of 1's in each column. An example output of the program is given below.

```
Enter the number of rows of the matrix:
[3]
Enter the number of columns of the matrix:
[4]
Enter the number of 1's in row 1:
[2]
Enter the column locations of the 1's in row 1:
[1]
[3]
Enter the number of 1's in row 2:
[3]
Enter the column locations of the 1's in row 2:
[1]
[2]
[3]
Enter the number of 1's in row 3:
[2]
Enter the column locations of the 1's in row 3:
[3]
[4]



OK, I got the matrix.
The number of 1's in column 1 is 2.
The number of 1's in column 2 is 1.
The number of 1's in column 3 is 3.
The number of 1's in column 4 is 1.
```

Verify that in this example, the matrix that the user enters is

```
1 0 1 0
1 1 1 0
0 0 1 1
```

This program needs to be implemented using two functions which you must design. The first function is named `getBinarySparseMatrixFromUser`, which must have return

type (int **). The second function is named `printColumnStatisticsOfMatrix`, which must have return type void. The main function should have the following structure:

1. declare variables
2. call getBinarySparseMatrixFromUser
3. call printColumnStatisticsOfMatrix

No global variables are allowed.

Steps from 1-6 should be included. Here is the programming code:

```
#include <stdio.h>
#include <stdlib.h>

int** getBinarySparseMatrixFromUser(int number_of_row, int number_of_col);
void printColumnStatisticsOfMatrix(int** matrix, int number_of_row, int
number_of_col);

void reset(int* p, int length);
void print_matrix(int **matrix, int row, int col);

int main()
{
    int **matrix,number_of_row=0,number_of_col=0;
    /*
    **matrix: the address of the matrix, will be allocated latter
    number_of_row/number_of_col: store the number of rows/columns of the
matrix, to be entered by the user
    */
    int ctr;//counter
    do
    {
        fflush(stdin);
        printf("Please enter the number of rows of the matrix:");
        scanf("%d",&number_of_row);
        if (number_of_row<1) {printf("Invalid input.\n");}
    }while (number_of_row<1);//enter the number of rows and ensure it >= 1
    do
    {
        fflush(stdin);
        printf("Please enter the number of columns of the matrix:");
        scanf("%d",&number_of_col);
        if (number_of_col<1) {printf("Invalid input.\n");}
    }while (number_of_col<1);//enter the number of columns and ensure it >= 1

    matrix=getBinarySparseMatrixFromUser(number_of_row,number_of_col);
    printColumnStatisticsOfMatrix(matrix, number_of_row, number_of_col);
    return 0;
}

int** getBinarySparseMatrixFromUser(int number_of_row, int number_of_col)
{
```

```
    int row_index,col_index,n_of_1_in_the_row=0;
    int** matrix;
    /*
    row_index/col_index: store the index of the row/column
    n_of_1_in_the_row: store the number of 1's in a row, to be entered by the
user
    */
    int ctr;//counter
    matrix=(int**)malloc(number_of_row*sizeof(int*));//allocate places for
the rows of matrix
    if (!matrix) {printf("Memory allocation failed.\n");return -1;}//check if
malloc()'s succeeded
    for (ctr=0;ctr<=number_of_row-1;ctr++)
    {
        matrix[ctr]=(int*)malloc(number_of_col*sizeof(int));//allocate places
for each rows
        if (!matrix[ctr]) {printf("Memory allocation failed.\n");return -
1;}//check if malloc()'s succeeded
    }
    for (ctr=0;ctr<=number_of_row-1;ctr++)
        reset(matrix[ctr],number_of_col);
    printf("\nFor the following input, enter ONLY one number each time and
then press enter. If more than one, the rest will be lost.\n\n");
    for (row_index=0;row_index<=number_of_row-1;row_index++)//row_index runs
from 0 to number_of_row-1, totally number_of_row times, each time, assign 1's
to the current row
    {
        n_of_1_in_the_row=-1;//reset the variable
        do
        {
            fflush(stdin);
            printf("Please enter the number of 1's om row
%d:\n",(row_index+1));
            scanf("%d",&n_of_1_in_the_row);
            if (n_of_1_in_the_row<0 ||
n_of_1_in_the_row>number_of_col){printf("Invalid input.\n");}
        }
        while (n_of_1_in_the_row<0 ||
n_of_1_in_the_row>number_of_col);//enter the n_of_1_in_the_row in the given
range
        if (n_of_1_in_the_row>0)//if n_of_1_in_the_row == 0, there is no 1 in
the row, then the following statement does not need to execute
        {
            printf("Please enter the column locations of the 1's in row
%d:\n",row_index+1);
            for (ctr=0;ctr<=n_of_1_in_the_row-1;ctr++)//counter runs totally
n_of_1_in_the_row times
            {
                col_index=0;//reset variable
                do
                {
                    fflush(stdin);
                    scanf("%d",&col_index);
                    col_index--;
                    if (col_index<0 || col_index>number_of_col-
1){printf("Invalid input.\n");}
                }
```

```
                while (col_index<0 || col_index>number_of_col-1);//enter
col_index, meaning column location of one '1' in current row
                matrix[row_index][col_index]=1;//assign 1 to the
corresponding entry of the matrix
            }
        }
    }
    return matrix;
}

void printColumnStatisticsOfMatrix(int** matrix, int number_of_row, int
number_of_col)
{
    int i,j,ctr=0;
    printf("\nOK I got the matrix.\n");
    for (j=0; j<= number_of_col-1;j++)
    {
        for (i=0; i<= number_of_row-1;i++)
        {
            if (matrix[i][j]==1)
                ctr++;
        }
        printf("The number of 1's in column %d is %d\n", j+1, ctr);
        ctr=0;
    }
    printf("\n");
    print_matrix(matrix, number_of_row, number_of_col);
}

/** set every member in the array to 0 **/
void reset(int* p, int length)
{
    int i;
    for (i=0;i<=length-1;i++)
    {
        p[i]=0;
    }
}

/** Print matrix by rows with '\t' between elements in the same row **/
void print_matrix(int **matrix, int row, int col)
{
    int i,j;
    for (i=0;i<=row-1;i++)
    {
        for (j=0;j<=col-1;j++)
        {
            printf("%d\t",matrix[i][j]);
        }
        printf("\n");
    }
}
```