Output:

```
Enter the number of elements in the array: 5
Enter the array elements:
4
3
1
2
5
Smallest element: 1
```

Date _____

Expt. No. _____ Page No. _____

Lab Sheet 2

1 Write a small function to find the smallest element in an array using pointers. In the main function, create a dynamically allocated array, read the values from the keyboard, and pass the array to the function. Display the result (smallest element) in the main function.

Source Code:

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

int find_small(int * arr, int n) {
    int i;
    int smallest = *arr;

    for(i=1; i < n; i++) {
        if(*(arr+i) < smallest) {
            smallest = *(arr+i);
        }
    }
    return smallest;
}

int main() {
```

Teacher's Signature _____

```
    int n, i;

    printf("Enter the number of elements in
the array: ");
    scanf("%d", &n);

    int* arr = (int*) malloc(n * sizeof(int));

    printf("Enter the array elements:\n");
    for(i=0; i<n; i++) {
        scanf("%d", arr+i);
    }
    .
    printf("Smallest element: %d\n", find_small(arr, n));
    free(arr);
    return 0;
}
```

Teacher's Signature _____

Output:

```
Enter the number of elements in the array: 5
Enter the elements:
4
3
1
2
5
Sorted elements:
1, 2, 3, 4, 5,
```

Date

Expt. No. Page No.

2. Write a recursive C program to implement Selection Sort using pointers.

- The recursive function should sort the array using the Selection Sort algorithm.
- Access and manipulate the array elements using pointers (i.e; avoid using arr[i] style direc-tly).
- The program should read the array from the user in the main function, call the recursive sorting function, and display the sorted array.

Source Code:

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

void selection-sort(int* arr, int n, int i, int
j, int min-idx) {
    if (i >= n-1) return;
    else if (j >= n) {
        if (min-idx != i) {
            *(arr+min-idx) += *(arr+i);
            *(arr+i) = *(arr+min-idx) - *(arr+i);
            *(arr+min-idx) -= *(arr+i);
        }
        selection-sort(arr, n, i+1, i+1, i+1);
    }
```

Teacher's Signature

Date

Expt. No. Page No.

```
    else if (*(arr+j) < *(arr+min_idx)) {
        selection_sort(arr, n, i, j+1, j);
    }
    else {
        selection_sort(arr, n, i, j+1, min_idx);
    }
}

int main() {
    int n, i;
    printf("Enter the number of elements in
the array: ");
    scanf("%d", &n);

    int * arr = (int *) malloc(n * sizeof(int));

    printf("Enter the elements: \n");
    for(i=0; i<n; i++) {
        scanf("%d", arr+i);
    }

    selection_sort(arr, n, 0, 1, 0);

    printf("Sorted elements: \n");
    for(i=0; i<n; i++) {
        printf("%d, ", *(arr+i));
    }
    free(arr);
    return 0;
}
```

Teacher's Signature

Output:

```
Enter the number of rows in matrix A: 3
Enter the number of cols in matrix A: 3
Enter the matrix A elements:
1 2 3
4 5 6
7 8 9
Enter the number of rows in matrix B: 3
Enter the number of cols in matrix B: 3
Enter the matrix B elements:
2 0 0
0 2 0
0 0 2

Matrix A:
1   2   3
4   5   6
7   8   9

Matrix B:
2   0   0
0   2   0
0   0   2

A * B =
2   4   6
8   10  12
14  16  18
```

Date

Expt. No. Page No.

3. Implement a C program to read, display, and find the product of two matrices using functions with appropriate parameters.

- The matrices must be created using dynamic allocation (malloc or calloc).
- Access matrix elements using array dereferencing (i.e; *(*(mat+i)+j) style).

Source Code:

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

int ** matrix_mul(int ** a, int ** b, int m1,
int n1, int m2, int n2) {
    int i, j, k;
    int ** arr = (int **) malloc(m1 * sizeof(int *));

    for(i=0; i<m1; i++) {
        *(arr+i) = (int *) malloc(n2 * sizeof(int));
    }

    for(i=0; i<m1; i++) {
        for(j=0; j<n2; j++) {
            *(*(arr+i)+j) = 0;
            for(k=0; k<n1; k++) {
                *(*(arr+i)+j) += *(*(a+i)+k) *
*(*(b+k)+j);
```

Teacher's Signature

```
        }
    }
}

return arr;
}

int main() {
    int i, j, m1, n1, m2, n2;
    printf("Enter the number of rows in matrix A: "); scanf("%d", &m1);
    printf("Enter the number of cols in matrix A: "); scanf("%d", &n1);

    int ** arr = (int **)malloc(m1 * sizeof(int *));
    for(i=0; i<m1; i++) {
        *(arr+i) = (int *)malloc(n1 * sizeof(int));
    }

    printf("Enter the matrix A elements:\n");
    for(i=0; i<m1; i++) {
        for(j=0; j<n1; j++) {
            scanf("%d", *(arr+i)+j);
        }
    }

    printf("Enter the number of rows in matrix B: "); scanf("%d", &m2);
    printf("Enter the number of cols in matrix
```

Teacher's Signature

```
B: "); scanf("%d", &n2);

    int** arr1 = (int**) malloc(m2 * sizeof(int*));
    for(i=0; i<m2; i++) {
        *(arr1+i) = (int*) malloc(n2 * sizeof(int));
    }

    printf("Enter the matrix B elements: \n");
    for(i=0; i<m2; i++) {
        for(j=0; j<n2; j++) {
            scanf("%d", *(arr1+i)+j));
        }
        printf("\n");
    }

    printf("\nMatrix A: \n");
    for(i=0; i<m1; i++) {
        for(j=0; j<n1; j++) {
            printf("%d\t", *(*(arr1+i)+j));
        }
        printf("\n");
    }

    printf("\nMatrix B: \n");
    for(i=0; i<m2; i++) {
        for(j=0; j<n2; j++) {
            printf("%d\t", *(*(arr1+i)+j));
        }
        printf("\n");
    }
```

Teacher's Signature _____

```
if(n1 != m2) {
    printf("Multiplication is not possible!\n");
    return 1;
}

int** product = matrix_mul(arr, arr1, m1, n1,
m2, n2);

printf("\nA * B = \n");
for(i=0; i<m1; i++) {
    for(j=0; j<n2; j++) {
        printf("%d \t", *(*(product+i)+j));
    }
    printf("\n");
}
for(int i=0; i<m1; i++) {free(*(arr+i)); free(*(product+i));
free(arr);
free(arr1);  <-- for(int i=0; i<m2; i++) free(*(arr1+i));
free(product);

return 0;
}
```

Teacher's Signature _____