Name : Deshmukh Mehmood Rehan

MIS No. : 612303050

SY Computer Science – Div 1

Question: Write a program to accept a file name containing a random number of integers as a command line argument. Sort and display these integers using ADT Heap.

Code:

header.h : This File includes the declarations of structures and function prototypes

```c
#include<stdio.h>
#include<stdlib.h>
#include<limits.h>
#include<errno.h>

/*
This is the initial size of the heap. If the heap is full, the size of the
heap will be doubled.
*/
#define MAX_SIZE 10


/*
This is the structure of the max heap.
it contains:
    - array: a pointer to the array that will store the data.
    - rear: the index of the last element in the heap.
    - size: the size of the heap.
*/

typedef struct max_heap{
    int *array;
    int rear;
    int size;
} max_heap;

/*
This is the structure of the min heap.
it contains:
    - array: a pointer to the array that will store the data.
    - rear: the index of the last element in the heap.
    - size: the size of the heap.
*/
```

```c
typedef struct min_heap{
    int *array;
    int rear;
    int size;
} min_heap;

/*These are the function prototypes */

void init_max_heap(max_heap *h);
void init_min_heap(min_heap *h);

int is_empty_max_heap(max_heap h);
int is_empty_min_heap(min_heap h);

void handle_is_full_max_heap(max_heap *h);
void handle_is_full_min_heap(min_heap *h);

void swap(int *a, int *b);
int get_parent_index(int index);

void insert_max_heap(max_heap *h, int data);
void insert_min_heap(min_heap *h, int data);

void heapify_max_heap(max_heap *h);
void heapify_min_heap(min_heap *h);

int remove_max_heap(max_heap *h);
int remove_min_heap(min_heap *h);

void print_max_heap(max_heap h);
void print_min_heap(min_heap h);

void free_max_heap(max_heap *h);
void free_min_heap(min_heap *h);

void heap_sort_max_heap(int *array, int size);
void heap_sort_min_heap(int *array, int size);
```

logic.c : contains the definition of all the functions declared in the header file along with some helper functions

```c
#include "header.h"


/*
This function initializes the max heap.
It takes a pointer to the max heap as an argument.
```

```
It allocates memory for the array that will store the data.
It sets the rear to -1.
It sets the size to MAX_SIZE.
*/
void init_max_heap(max_heap *h){
    h->array = (int *) malloc(sizeof(int) * MAX_SIZE);
    h->rear = -1;
    h->size = MAX_SIZE;
}

/*
This function initializes the min heap.
It takes a pointer to the min heap as an argument.

It allocates memory for the array that will store the data.
It sets the rear to -1.
It sets the size to MAX_SIZE.
*/
void init_min_heap(min_heap *h){
    h->array = (int *) malloc(sizeof(int) * MAX_SIZE);
    h->rear = -1;
    h->size = MAX_SIZE;
}

/*
This function checks if the max heap is empty.
It returns 1 if the heap is empty, 0 otherwise.
if the rear is -1, then the heap is empty.

*/
int is_empty_max_heap(max_heap h){
    return h.rear == -1;
}

/*
This function checks if the min heap is empty.
It returns 1 if the heap is empty, 0 otherwise.
if the rear is -1, then the heap is empty.
*/

int is_empty_min_heap(min_heap h){
    return h.rear == -1;
}

/*
This function handles the case when the max heap is full.
It takes a pointer to the max heap as an argument.
```

```c
/*
If the rear is equal to the size of the heap - 1, then the heap is full.
In this case, the function reallocates memory for the array and doubles the
size of the heap.
*/

void handle_is_full_max_heap(max_heap *h){
    if(h->rear == h->size - 1){
        h->array = (int *) realloc(h->array, sizeof(int) * h->size * 2);
        h->size = h->size * 2;

    }


    return;
}

/*
This function handles the case when the min heap is full.
It takes a pointer to the min heap as an argument.

If the rear is equal to the size of the heap - 1, then the heap is full.
In this case, the function reallocates memory for the array and doubles the
size of the heap.
*/

void handle_is_full_min_heap(min_heap *h){
    if(h->rear == h->size - 1){
        h->array = (int *) realloc(h->array, sizeof(int) * h->size * 2);
        h->size = h->size * 2;
    }

    return;
}

/*
This is a helper function that swaps the values of two integers.
*/

void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

/*
This is a helper function that returns the index of the parent of a node.
```

```c
A node at index i has its parent at index (i - 1) / 2.
*/
int get_parent_index(int index){
    return (index - 1) / 2;
}

/*
This function inserts a new element into the max heap.
It takes a pointer to the max heap and the data to be inserted as arguments.

It first checks if the heap is full using the handle_is_full_max_heap
function.
It then increments the rear and inserts the data at the rear of the heap.

It then repeatedly swaps the data with its parent until the heap property is
satisfied.
*/

void insert_max_heap(max_heap *h, int data){
    handle_is_full_max_heap(h);

    int i = ++h->rear;
    h->array[i] = data;

    while(i > 0 && h->array[get_parent_index(i)] < h->array[i]){
        swap(&h->array[get_parent_index(i)], &h->array[i]);
        i = get_parent_index(i);
    }

    return;
}

/*
This function inserts a new element into the min heap.
It takes a pointer to the min heap and the data to be inserted as arguments.
It first checks if the heap is full using the handle_is_full_min_heap
function.

It then increments the rear and inserts the data at the rear of the heap.
It then repeatedly swaps the data with its parent until the heap property is
satisfied.
*/

void insert_min_heap(min_heap *h, int data){
    handle_is_full_min_heap(h);

    int i = ++h->rear;
    h->array[i] = data;
```

```c
    while(i > 0 && h->array[get_parent_index(i)] > h->array[i]){
        swap(&h->array[get_parent_index(i)], &h->array[i]);
        i = get_parent_index(i);
    }

    return;
}

/*
This function heapifies the max heap.
It takes a pointer to the max heap as an argument.

It starts at the root of the heap and repeatedly swaps the root with its
largest child until the heap property is satisfied.
*/
void heapify_max_heap(max_heap *h){
    int i = 0;
    int left_child_index;
    int right_child_index;
    int largest_child_index;

    while(1){
        left_child_index = 2 * i + 1;
        right_child_index = 2 * i + 2;
        largest_child_index = i;

        if(left_child_index <= h->rear && h->array[left_child_index] > h-
>array[largest_child_index]){
            largest_child_index = left_child_index;
        }

        if(right_child_index <= h->rear && h->array[right_child_index] > h-
>array[largest_child_index]){
            largest_child_index = right_child_index;
        }

        if(largest_child_index == i){
            break;
        }

        swap(&h->array[i], &h->array[largest_child_index]);
        i = largest_child_index;
    }

    return;
}
```

```c
/*
This function heapifies the min heap.
It takes a pointer to the min heap as an argument.

It starts at the root of the heap and repeatedly swaps the root with its
smallest child until the heap property is satisfied.
*/

void heapify_min_heap(min_heap *h){
    int i = 0;
    int left_child_index;
    int right_child_index;
    int smallest_child_index;

    while(1){
        left_child_index = 2 * i + 1;
        right_child_index = 2 * i + 2;
        smallest_child_index = i;

        if(left_child_index <= h->rear && h->array[left_child_index] < h-
>array[smallest_child_index]){
            smallest_child_index = left_child_index;
        }

        if(right_child_index <= h->rear && h->array[right_child_index] < h-
>array[smallest_child_index]){
            smallest_child_index = right_child_index;
        }

        if(smallest_child_index == i){
            break;
        }

        swap(&h->array[i], &h->array[smallest_child_index]);
        i = smallest_child_index;
    }

    return;
}

/*
This function removes the maximum element from the max heap.
It takes a pointer to the max heap as an argument.

It returns INT_MIN if the heap is empty.
It returns the maximum element in the heap.
It then replaces the root with the last element in the heap and heapifies the
heap.
```

```c
*/

int remove_max_heap(max_heap *h){
    if(is_empty_max_heap(*h)){
        return INT_MIN;
    }

    int data = h->array[0];
    h->array[0] = h->array[h->rear--];
    heapify_max_heap(h);

    return data;
}

/*
This function removes the minimum element from the min heap.
It takes a pointer to the min heap as an argument.

It returns INT_MAX if the heap is empty.
It returns the minimum element in the heap.
It then replaces the root with the last element in the heap and heapifies the
heap.
*/

int remove_min_heap(min_heap *h){
    if(is_empty_min_heap(*h)){
        return INT_MAX;
    }

    int data = h->array[0];
    h->array[0] = h->array[h->rear--];
    heapify_min_heap(h);

    return data;
}

/*
This function prints the max heap.
*/

void print_max_heap(max_heap h){
    if(is_empty_max_heap(h)) return;

    for(int i = 0; i <= h.rear; i++){
        printf("%d ", h.array[i]);
    }

    printf("\n");
```

```c
}

/*
This function prints the min heap.
*/

void print_min_heap(min_heap h){
    if(is_empty_min_heap(h)) return;

    for(int i = 0; i <= h.rear; i++){
        printf("%d ", h.array[i]);
    }

    printf("\n");
}

/*
This function frees the memory allocated for the max heap.
*/

void free_max_heap(max_heap *h){
    free(h->array);
    h->rear = -1;
    h->size = 0;
}

/*
This function frees the memory allocated for the min heap.
*/

void free_min_heap(min_heap *h){
    free(h->array);
    h->rear = -1;
    h->size = 0;
}

/*
This function sorts an array using the max heap.

It first initializes the max heap and inserts all the elements of the array
into the heap.
It then removes the elements from the heap in descending order and stores them
back in the array in
reverse order.
*/

void heap_sort_max_heap(int *array, int size){
    max_heap h;
```

```
    init_max_heap(&h);

    for(int i = 0; i < size; i++){
        insert_max_heap(&h, array[i]);
    }

    for(int i = size - 1; i >= 0; i--){
        array[i] = remove_max_heap(&h);
    }

    free_max_heap(&h);

    return;
}

/*
This function sorts an array using the min heap.

It first initializes the min heap and inserts all the elements of the array
into the heap.
It then removes the elements from the heap in ascending order and stores them
back in the array.
*/

void heap_sort_min_heap(int *array, int size){
    min_heap h;
    init_min_heap(&h);

    for(int i = 0; i < size; i++){
        insert_min_heap(&h, array[i]);
    }

    for(int i = 0; i < size; i++){
        array[i] = remove_min_heap(&h);
    }

    free_min_heap(&h);

    return;
}
```

main.c : This contains the code to test the implementation

```
#include "header.h"

void print_array(int *array, int size);

int main(int argv, char *argc[]){
```

```c
    if(argv < 2){
        printf("Usage: %s <file with numbers to sort>\n", argc[0]);
        exit(1);
    }

    FILE *file = fopen(argc[1], "r");
    if(file == NULL){
        perror("Error opening file: ");
        exit(1);
    }

    int size;
    fscanf(file, "%d", &size);

    int *array = (int *) malloc(sizeof(int) * size);
    for(int i = 0; i < size; i++){
        fscanf(file, "%d", &array[i]);
    }

    fclose(file);

    int *array_copy = (int *) malloc(sizeof(int) * size);
    for(int i = 0; i < size; i++){
        array_copy[i] = array[i];
    }

    printf("Sorting using max heap\n");

    printf("Before sorting: ");
    print_array(array, size);

    heap_sort_max_heap(array, size);

    printf("After sorting: ");
    print_array(array, size);

    printf("\n");

    printf("Sorting using min heap\n");

    printf("Before sorting: ");
    print_array(array_copy, size);

    heap_sort_min_heap(array_copy, size);

    printf("After sorting: ");
    print_array(array_copy, size);
```

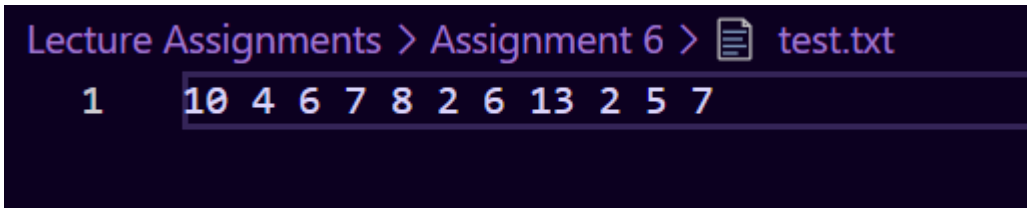
```c
    free(array);
    free(array_copy);

    return 0;
}

void print_array(int *array, int size){
    for(int i = 0; i < size; i++){
        printf("%d ", array[i]);
    }

    printf("\n");
}
```

Textfile:



Output: