Name: Yashwant Chandrakant Bhosale
MIS: 612303039
SY COMP Div 1

Assignment 6- Heap sort

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.

Solution:
heap.h: header file for the heap ADT; contains function prototypes and struct declarations for the heap ADT

```
typedef struct heap {
    int *h;
    int size;
    int rear;
} Heap;

void init_heap(Heap *heap, int size);
void insert_heap(Heap *heap, int data);
void print_heap(Heap *heap);
void heap_sort(Heap *heap);
```

heap.c: contains the logic or function definitions for the functions of the heap ADT

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

// utility functions used throughout the program
int get_parent_index(int index) {
    return (index - 1) / 2;
}

int lchild_index(int index) {
    return 2 * index + 1;
}

int rchild_index(int index) {
    return 2 * index + 2;
}
```

```c
// function to swap two elements in the heap
void swap(Heap *heap, int i, int j) {
    // Make sure that the indices are valid
    if(i >= heap->size || i < 0 || j >= heap->size || j < 0) {
        return;
    }

    int temp = heap->h[i];
    heap->h[i] = heap->h[j];
    heap->h[j] = temp;

    return;
}

// function to initialize heap
void init_heap(Heap *heap, int size) {
    heap->h = (int *)malloc(sizeof(int) * size);
    heap->size = size;
    heap->rear = -1;
    return;
}


// function to print the heap
void print_heap(Heap *heap) {
    int i;

    printf("[  ");
    for(i = 0; i <= heap->rear; i++) {
        printf("%d  ", heap->h[i]);
    }
    printf("]\n");
    return;
}
```

```
// function to insert an element in the heap

void insert_heap(Heap *heap, int d) {
    if(heap->rear == heap->size - 1) {
        // Heap is full
        return;
    }
    /*
        approach for insertion:
        1. insert the element at the end of the Heap
        2. keep comparing the element with its parent
        3. swap the element with its parent if the element is
           greater than its parent

        At the end of this process, the parent of the element will
        be greater than the element
    */
    int i;
    heap->h[++heap->rear] = d;

    i = heap->rear;
    while(i>0 && heap->h[i] > heap->h[get_parent_index(i)]) {
        // swap the element with its parent
        swap(heap, i, get_parent_index(i));
        i = get_parent_index(i); // update the value of i to move
        up the heap and keep checking
    }
    return;
}


// function to heapify the heap: convert the heap into a max heap
void heapify(Heap *heap) {
    // If the heap is empty or contains single element, return as
       the heap is already a max heap
    if(heap->rear <= 0) {
        return;
    }

    int i = 0;
    while(i < heap->rear) {
        int lchild = lchild_index(i);
        int rchild = rchild_index(i);
        // if the lchild index is outof bounds then return
        if(lchild > heap->rear) {
            return;
        }

        // if the rchild index is outof bounds
        if(rchild > heap->rear) {
            // if the lchild is greater than the parent, swap the
               parent and lchild
            if(heap->h[lchild] > heap->h[i]) {
                swap(heap, lchild, i);
            }
            return;
        }

        // if the parent is greater than both the children, return
        if(heap->h[i]>heap->h[lchild] && heap->h[i]>heap->h[rchild]) {
            return;
        }

        // NOTE: Next two conditions most likely be not true as we
           take care of it at the time of insertion
        // if the parent is less than the left child and greater
           than the right child, swap the parent and left child
        if(heap->h[i]<heap->h[lchild] && heap->h[i]>heap->h[rchild]) {
            swap(heap, i, lchild);
            i = lchild;
        }

        // if the parent is less than the right child and greater
           than the left child, swap the parent and right child
        else if
          (heap->h[i]<heap->h[rchild] && heap->h[i]>heap->h[lchild]) {
            swap(heap, i, rchild);
            i = rchild;
        }
        // if the parent is less than both the children, swap the
            parent with the child which is greater
        else {
            if(heap->h[lchild] > heap->h[rchild]) {
                swap(heap, i, lchild);
                i = lchild;
            } else {
                swap(heap, i, rchild);
                i = rchild;
            }
        }
    }
    return;
}

void heap_sort(Heap *heap) {
     // If the heap is empty or contains single element, return as
        there is nothing to sort
     if (heap->rear <= 0) {
          return;
     }

     int old_rear = heap->rear;
     for (int i = heap->rear; i > 0; i--) {
          swap(heap, i, 0);
          /*
          At this point, the heap is a max heap
          1. swap the first element with the last element
          2. reduce the size of the heap by 1 (update the ear pointer)
          3. heapify the heap
          * why swap? : the largest element is considered to be sorted
            at each iteration so we swap the largest element with the
            rear element pointer and reduce the  rear pointer by 1
          * why heapify? : the heap is no longer a max heap after
            swapping the first element with the last element
          */

          // update the rear pointer
          heap->rear--;
          // heapify the unsored heap
          heapify(heap);
     }
     heap->rear = old_rear;
     return;
}
```

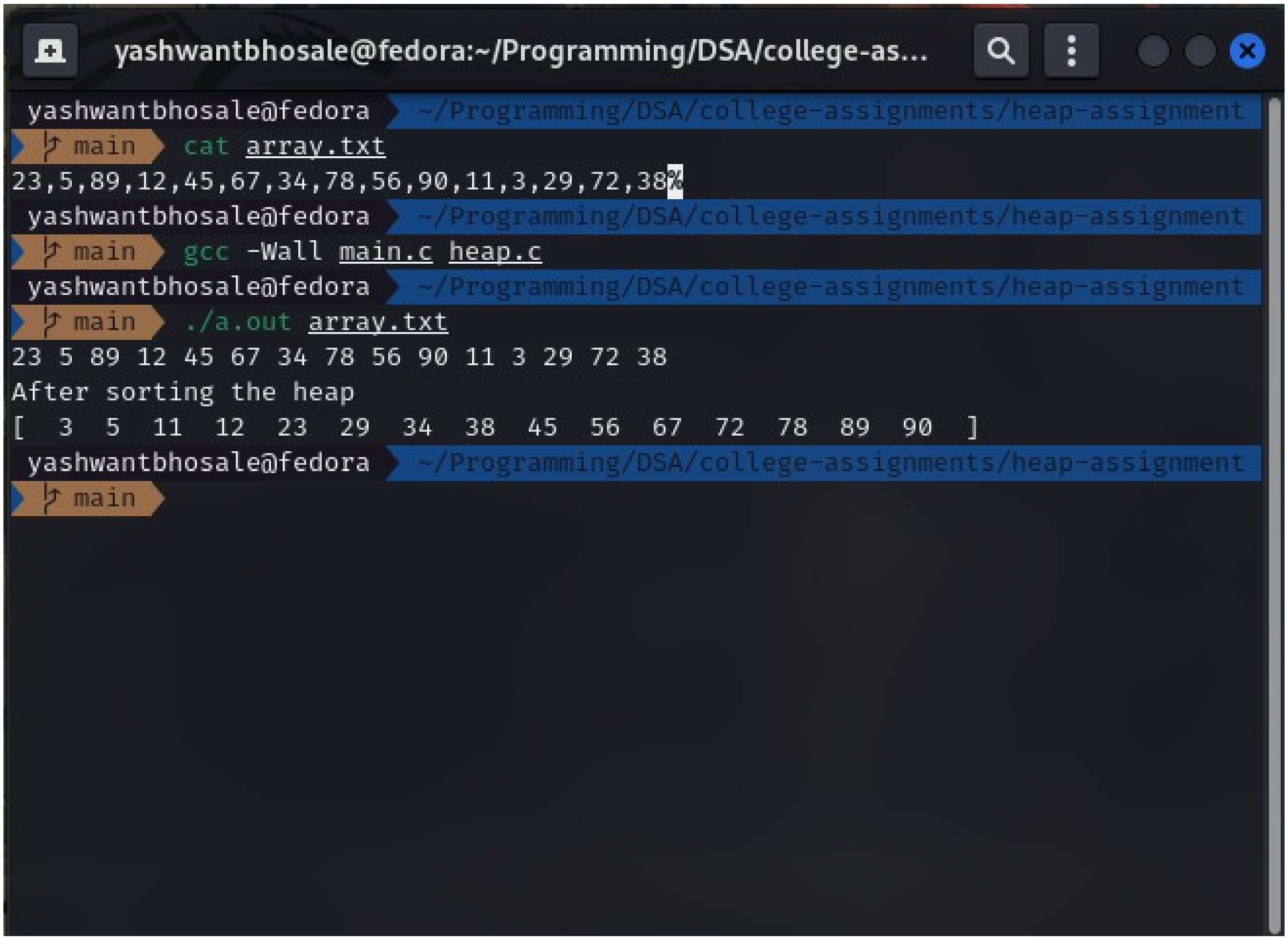
main.c: Contains logic for reading array from the file passed as argument and perform heap sort on it

```
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "heap.h"

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

int main(int argc, char **argv) {
     Heap h1;
     char str[100], *token;
     int arr[100];

     int fd = open(argv[1], O_RDONLY);
     if (fd == -1) {
          perror("Failed to open file\n");
          return 1;
     }

     int bytes = read(fd, str, sizeof(char) * 100);
     if (bytes == -1) {
          perror("Failed to read file\n");
          return 1;
     }

     token = strtok(str, ",");
     int i = 0;
     while (token != NULL) {
          arr[i++] = atoi(token);
          token = strtok(NULL, ",");
     }
```

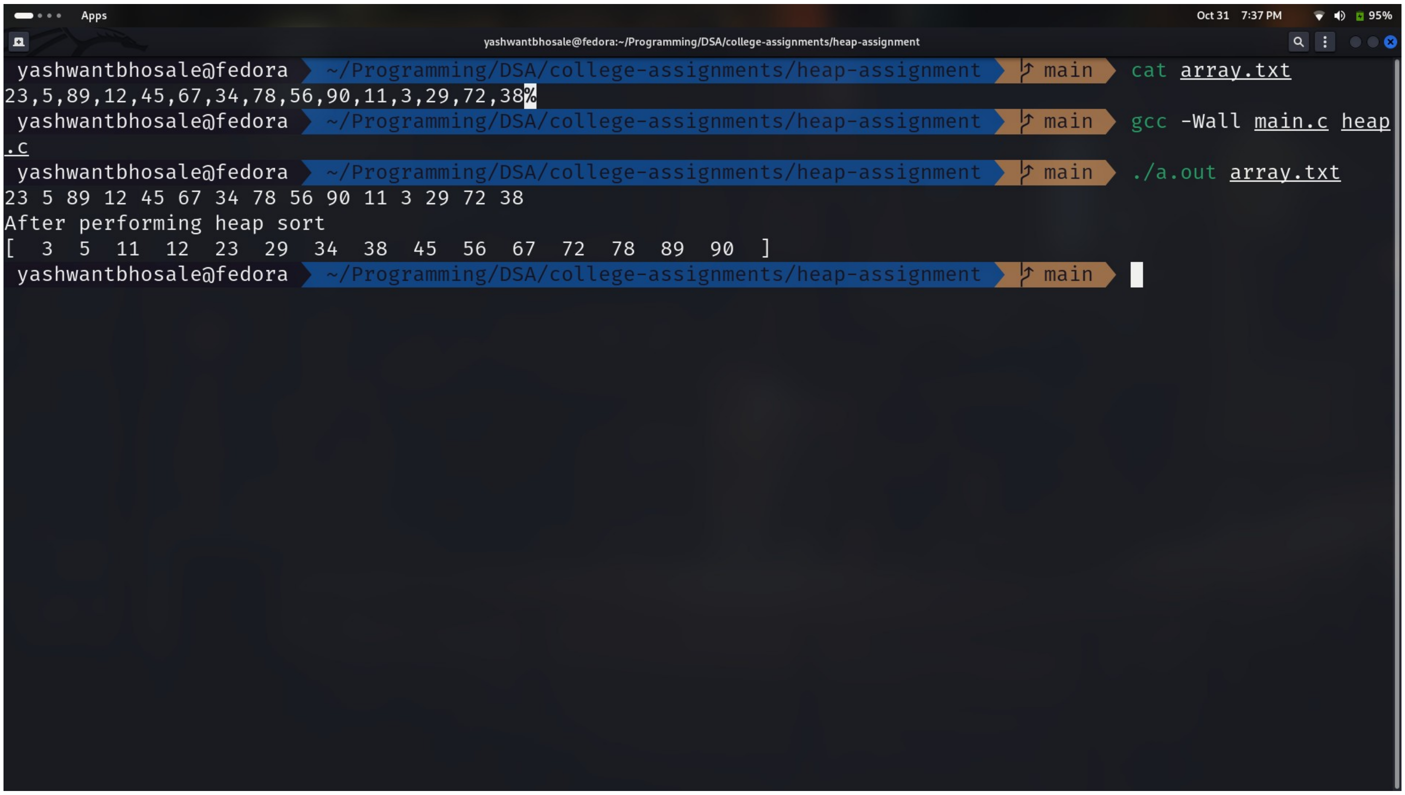
```
    print_array(arr, i);
    init_heap(&h1, i);

    for (int j = 0; j < i; j++) {
        insert_heap(&h1, arr[j]);
    }
    printf("After performing heap sort\n");
    heap_sort(&h1);
    print_heap(&h1);

    return 0;
}
```

Output:

NOTE: array.txt contains the array to be sorted it is passed to a.out so that we can extract data by reading the file



```
yashwantbhosale@fedora  ~/Programming/DSA/college-assignments/heap-assignment
 main  cat array.txt
23,5,89,12,45,67,34,78,56,90,11,3,29,72,38%
yashwantbhosale@fedora  ~/Programming/DSA/college-assignments/heap-assignment
 main  gcc -Wall main.c heap.c
yashwantbhosale@fedora  ~/Programming/DSA/college-assignments/heap-assignment
 main  ./a.out array.txt
23 5 89 12 45 67 34 78 56 90 11 3 29 72 38
After sorting the heap
[  3   5   11   12   23   29   34   38   45   56   67   72   78   89   90   ]
yashwantbhosale@fedora  ~/Programming/DSA/college-assignments/heap-assignment
 main
```



Apps
Oct 31 7:37 PM
95%
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/heap-assignment

```
 yashwantbhosale@fedora  ~/Programming/DSA/college-assignments/heap-assignment  main  cat array.txt
23,5,89,12,45,67,34,78,56,90,11,3,29,72,38%
 yashwantbhosale@fedora  ~/Programming/DSA/college-assignments/heap-assignment  main  gcc -Wall main.c heap
.c
 yashwantbhosale@fedora  ~/Programming/DSA/college-assignments/heap-assignment  main  ./a.out array.txt
23 5 89 12 45 67 34 78 56 90 11 3 29 72 38
After performing heap sort
[  3  5  11  12  23  29  34  38  45  56  67  72  78  89  90  ]
 yashwantbhosale@fedora  ~/Programming/DSA/college-assignments/heap-assignment  main 
```