1.main.c

```
#include <stdio.h>

#include <stdlib.h>

#include "header.h"


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

    if (argc != 2) {

        printf("Usage: %s <filename>\n", argv[0]);

        return 1;

    }


    const char *filename = argv[1];

    Heap heap;

    initHeap(&heap);


    readFromFile(filename, &heap);

    heapSort(&heap, filename);

    freeHeap(&heap);

    return 0;

}
```

2.header.h

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

#define MAX_HEAP_SIZE 100

typedef struct Heap {
   int *a;
   int size;
   int rear;
} Heap;

void initHeap(Heap *heap);
void freeHeap(Heap *heap);
void insert(Heap *heap, int value);
int deleteMin(Heap *heap);
void heapify(Heap *heap);
void heapSort(Heap *heap, const char *filename);
void readFromFile(const char *filename, Heap *heap);
```

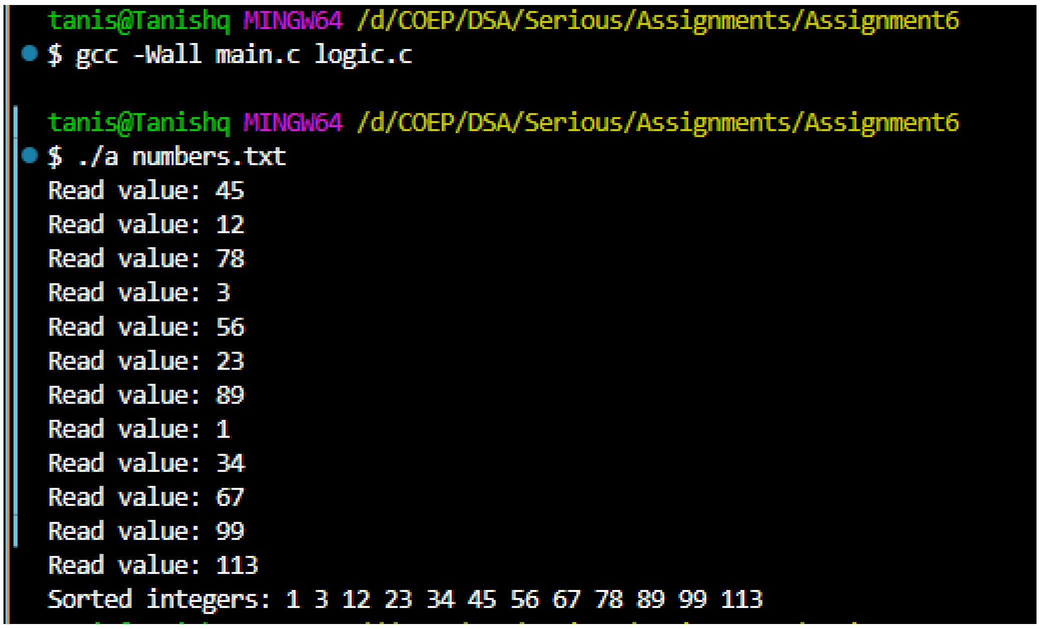
3.logic.c

```
#include "header.h"

// Initialize the heap

void initHeap(Heap *heap) {

   heap->size = MAX_HEAP_SIZE; // Set the total capacity

   heap->rear = -1; // Initialize rear to -1 (empty heap)

   heap->a = (int *)malloc(sizeof(int) * heap->size); // Allocate memory for heap elements

}

// Free the heap memory

void freeHeap(Heap *heap) {

   free(heap->a); // Free allocated memory for heap elements

}

// Insert an element into the heap

void insert(Heap *heap, int value) {

   if (heap->rear >= heap->size - 1) {

      printf("Heap overflow: Cannot insert %d\n", value);

      return;

   }

   heap->a[++heap->rear] = value; // Insert at the end

   int i = heap->rear; // Current index

   int parent = (i - 1) / 2; // Parent index

   // Heapify up

   while (i > 0 && heap->a[i] < heap->a[parent]) { // Min-heap property

      // Swap parent and current element

      int temp = heap->a[i];

      heap->a[i] = heap->a[parent];

      heap->a[parent] = temp;

      i = parent; // Move up to parent

      parent = (i - 1) / 2; // Update parent index

   }

}
```

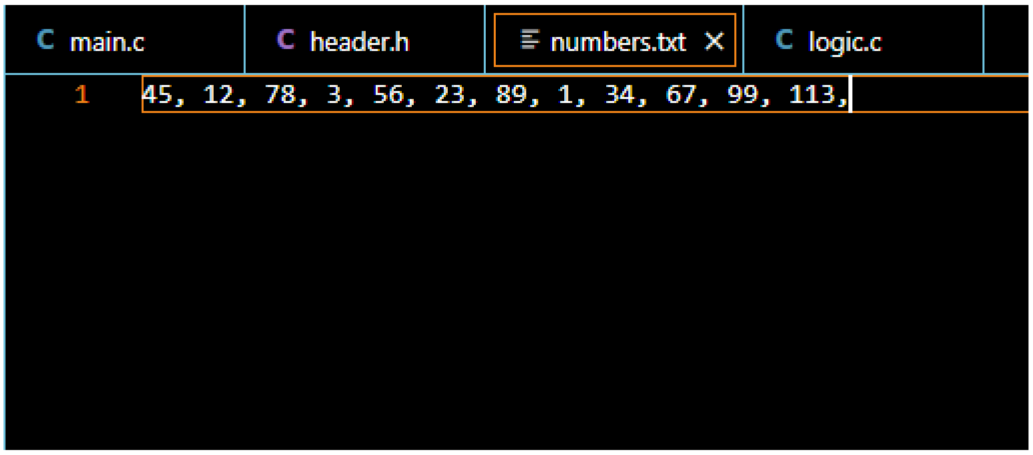
```c
// Delete the minimum element from the heap
int deleteMin(Heap *heap) {
  if (heap->rear < 0) {
    printf("Heap underflow\n");
    return -1; // Heap is empty
  }
  int min = heap->a[0]; // Minimum element
  heap->a[0] = heap->a[heap->rear--]; // Replace root with the last element
  heapify(heap); // Restore heap property
  return min;
}
// Maintain the heap property
void heapify(Heap *heap) {
  int i = 0;
  while (i * 2 + 1 <= heap->rear) { // While has at least one child
    int minChild = i * 2 + 1; // Left child
    if (minChild + 1 <= heap->rear && heap->a[minChild + 1] < heap->a[minChild]) {
      minChild++; // Choose the smaller child
    }
    if (heap->a[i] <= heap->a[minChild]) {
      break; // Heap property satisfied
    }
    // Swap parent and smaller child
    int temp = heap->a[i];
    heap->a[i] = heap->a[minChild];
    heap->a[minChild] = temp;
    i = minChild; // Move down to the child
  }
}
```

```
// Sort the heap and display sorted elements
void heapSort(Heap *heap, const char *filename) {
  int originalRear = heap->rear; // Store the original rear for sorting
  // Open the file in append mode to add sorted and unsorted contents
  FILE *file = fopen(filename, "a");
  if (!file) {
    printf("Error opening file for writing\n");
    return;
  }
  printf("Sorted integers: ");
  fprintf(file, "\nSorted integers: ");
  for (int i = 0; i <= originalRear; i++) {
    int minValue = deleteMin(heap);
    printf("%d ", minValue);
    fprintf(file, "%d ", minValue);  // Write each sorted integer to the file
  }
  fclose(file);
}
void readFromFile(const char *filename, Heap *heap) {
  FILE *file = fopen(filename, "r");
  if (!file) {
    printf("Error opening file\n");
    exit(1);
  }
  int value;
  while (fscanf(file, "%d,", &value) != EOF) {  // Handle comma-separated format
    printf("Read value: %d\n", value); // Debugging statement
    insert(heap, value);
  }
  fclose(file);
}
```

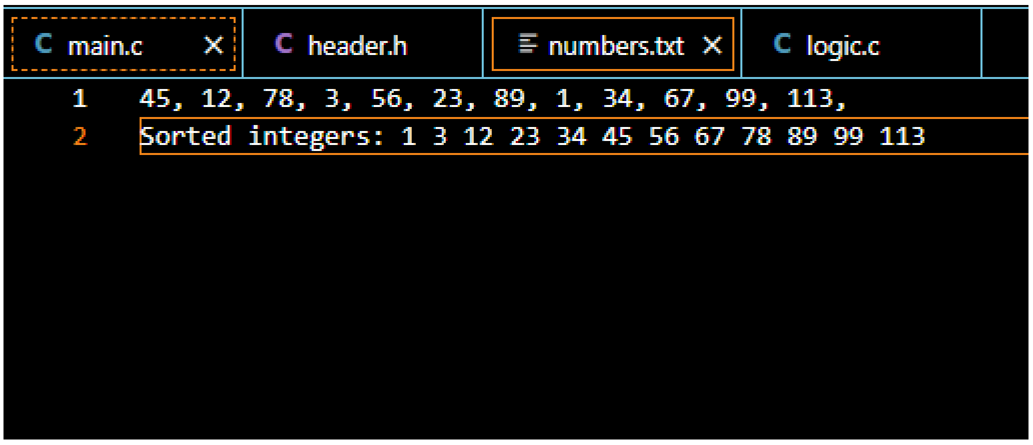
Output:



```
tanis@Tanishq MINGW64 /d/COEP/DSA/Serious/Assignments/Assignment6
$ gcc -Wall main.c logic.c

tanis@Tanishq MINGW64 /d/COEP/DSA/Serious/Assignments/Assignment6
$ ./a numbers.txt
Read value: 45
Read value: 12
Read value: 78
Read value: 3
Read value: 56
Read value: 23
Read value: 89
Read value: 1
Read value: 34
Read value: 67
Read value: 99
Read value: 113
Sorted integers: 1 3 12 23 34 45 56 67 78 89 99 113
```



```
45, 12, 78, 3, 56, 23, 89, 1, 34, 67, 99, 113,
```



```
45, 12, 78, 3, 56, 23, 89, 1, 34, 67, 99, 113,
Sorted integers: 1 3 12 23 34 45 56 67 78 89 99 113
```