Design Specifications

Introduction:

The scope of this design document is to explain the working and specifications of the code delivered as a part of the coding exercise. The problem states to write an ANSI C program which takes as arguments the name of a binary input file and a text output file. The binary input file will contain 12 bit unsigned values. The output file should contain the 32 largest values from the input file. It should also contain the last 32 values read from the input file.

The output file format should be as follows

- Start with "--Sorted Max 32 Values--"
- The 32 largest values in the file, one value per line. Duplicates are allowed if a large value appears multiple times.
- The list should be sorted smallest to largest.
- Output "--Last 32 Values--" on its own line
- The last 32 values read, one per line.
- They should appear in the order they were read.
- The last value read from the file will appear last.

Design Specifications:

The output specifies to read the 12 bit integers from the binary file and store the maximum 32 integers and the last 32 integers to the output file.

1. Maximum 32 integers: The binary file can have 12 bit integers of any number (as duplicates are allowed). Hence reading all the integer values from the file and loading those on memory (either stack or heap) will not be feasible for a low memory device. As we only need the maximum 32 values the following approach consumes the least memory:
    - Globally define 2 unsigned short integer array of length 32.
        - Curr32[] – stores the recently read 32 12-bit integers.
        - Max32[] – stores the maximum 32 12-bit integers.
    - Read 3 bytes of binary data from the file; in case of odd number of 12-bit integers the last nibble will be ignored.
    - Convert the first 3 nibbles to an unsigned short integer and convert the second 3 nibbles to an unsigned short integer.
    - Store these values to a predefined unsigned short integer array of length 32 – **curr32[]**
    - Check if the **curr32[]** is full; if full Sort the values in **curr32[]** array and merge them with **max32[]** in increasing order to update the max32[] with the latest maximum values; Reset the values in **curr32[]** to read the next set of 32 values.
    - Repeat these steps until all the bytes are read from the input bin file.
    - Finally sort and merge the left over values from the **curr32[]** array to **max32[].**
2. Last 32 integers: The idea is to save only the last 32 integers so there is no need to store the data from the entire file to memory. The following approach uses the dynamic memory to save the last 32 values:
    - Read 3 bytes of binary data from the file; in case of odd number of 12-bit integers the last nibble will be ignored.

- Convert the first 3 nibbles to an unsigned short integer and convert the second 3 nibbles to and unsigned short integer.
- These integers are added to a dynamically allocated Queue data structure with a maximum size of 32.
- As the queue size exceeds the earliest read element is de-queued and the latest read element is en-queued.
- At end of the read, the Queue holds 32 last read values or as many integers if they are less than 32 integers in the input bin file.

Memory consumption:

Stack memory: The maximum 32 integers algorithm uses stack.

1. Globally allocated 2 unsigned short integer array of length 32 – 32*2*sizeof(unsigned short) = 128 bytes of constant memory.
2. Temporarily allocated 1 unsigned short integer array of length 64 – 64* sizeof(unsigned short) = 128 bytes of constant memory.

Heap memory: The last 32 integers algorithm uses heap.

1. QueueList: 12 bytes of constant memory.
2. QueueNode: 32*8 = 256 bytes of maximum memory.

Performance:

1. Maximum 32 integers:
    a. Sorting of 32 integers is O(32log32) – constant time.
    b. Merging of 32 + 32 integers O(32+32) – constant time.
    c. Data read from file is O(n) – depends on the file size.
2. Last 32 integers:
    a. The Queue en-queue and de-queue operation are O(1) – constant time.

Tradeoffs:

With low memory constraints storing the maximum 32 values takes more time to read every 32 values from the file, sort then and merge them to the max values array.

With higher flexibility in memory, we can use a counting sort technique to store the values read from the files and save max 32 values to the output file. The time complexity would be O(1) overall but the memory consumption would be array of integers of size 4096 – 4096*4 = 16KB of constant memory on stack.