#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#include <time.h>

#define MAX\_FILENAME\_LENGTH 255

**//Generate the random input file name**

void generateRandomInputFile(char\* filename, int numInts)

{

FILE\* file = fopen(filename, "w");

if (file == NULL) {

perror("Error creating input file");

exit(EXIT\_FAILURE);

}

srand(time(NULL));

for (int i = 0; i < numInts; i++) {

fprintf(file, "%d\n", rand());

}

printf("\nFile created successfully!\n");

fclose(file);

}

**// Simple Merge Sort implementation for sorting each chunk**

void merge(int arr[], int l, int m, int r) {

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++) { L[i] = arr[l + i]; }

for (j = 0; j < n2; j++) { R[j] = arr[m + 1 + j]; }

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

**//Main merge sort using the above MERGE**

void mergeSort(int arr[], int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

**// Function to merge sorted chunks from temporary files**

void mergeSortFiles(char\* outputFile, char\*\* tempFiles, int numFiles) {

FILE\* outFile = fopen(outputFile, "w");

FILE\*\* fps = malloc(numFiles \* sizeof(FILE\*));

int\* values = malloc(numFiles \* sizeof(int));

char line[256];

int activeFiles = numFiles;

for (int i = 0; i < numFiles; i++) {

fps[i] = fopen(tempFiles[i], "r");

if (fgets(line, sizeof(line), fps[i]) != NULL) {

values[i] = atoi(line);

} else {

values[i] = INT\_MAX;

activeFiles--;

}

}

while (activeFiles > 0) {

int minIndex = 0;

int minValue = INT\_MAX;

for (int i = 0; i < numFiles; i++) {

if (values[i] < minValue) {

minValue = values[i];

minIndex = i;

}

}

if (minValue != INT\_MAX) {

fprintf(outFile, "%d\n", minValue);

if (fgets(line, sizeof(line), fps[minIndex]) != NULL) {

values[minIndex] = atoi(line);

} else {

values[minIndex] = INT\_MAX;

activeFiles--;

}

}

}

for (int i = 0; i < numFiles; i++) {

fclose(fps[i]);

remove(tempFiles[i]); // Ensure temporary files are removed.

}

fclose(outFile);

free(fps);

free(values);

}

**//External SORT function.**

void externalSort(char\* inputFileName, char\* outputFileName, int bufferSize) {

clock\_t start = clock();

FILE\* inputFile = fopen(inputFileName, "r");

if (!inputFile) {

perror("Failed to open input file");

exit(EXIT\_FAILURE);

}

int numTempFiles = 0;

int\* buffer = malloc(bufferSize \* sizeof(int));

if (!buffer) {

perror("Failed to allocate memory for buffer");

exit(EXIT\_FAILURE);

}

char\*\* tempFiles = malloc(0); // Dynamically grow this array as needed

char tempFileName[MAX\_FILENAME\_LENGTH];

int bufferIndex = 0;

while (fscanf(inputFile, "%d", &buffer[bufferIndex]) != EOF) {

bufferIndex++;

if (bufferIndex == bufferSize) {

mergeSort(buffer, 0, bufferSize - 1);

snprintf(tempFileName, sizeof(tempFileName), "temp%d.txt", numTempFiles);

FILE\* tempFile = fopen(tempFileName, "w");

for (int i = 0; i < bufferSize; i++) {

fprintf(tempFile, "%d\n", buffer[i]);

}

fclose(tempFile);

tempFiles = realloc(tempFiles, (numTempFiles + 1) \* sizeof(char\*));

tempFiles[numTempFiles] = malloc(MAX\_FILENAME\_LENGTH \* sizeof(char));

strcpy(tempFiles[numTempFiles], tempFileName);

numTempFiles++;

bufferIndex = 0; // Reset index to start filling the buffer again

}

}

**// Sort and write remaining items in buffer if any**

if (bufferIndex > 0) {

mergeSort(buffer, 0, bufferIndex - 1);

snprintf(tempFileName, sizeof(tempFileName), "temp%d.txt", numTempFiles);

FILE\* tempFile = fopen(tempFileName, "w");

for (int i = 0; i < bufferIndex; i++) {

fprintf(tempFile, "%d\n", buffer[i]);

}

fclose(tempFile);

tempFiles = realloc(tempFiles, (numTempFiles + 1) \* sizeof(char\*));

tempFiles[numTempFiles] = malloc(MAX\_FILENAME\_LENGTH \* sizeof(char));

strcpy(tempFiles[numTempFiles], tempFileName);

numTempFiles++;

}

fclose(inputFile);

free(buffer);

mergeSortFiles(outputFileName, tempFiles, numTempFiles);

// Cleanup

for (int i = 0; i < numTempFiles; i++) {

free(tempFiles[i]);

}

free(tempFiles);

clock\_t end = clock();

double timeTaken = ((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("\nElapsed time to sort %s using %d temp files\n", inputFileName, numTempFiles);

printf("%.4f minutes\n", timeTaken / 60);

printf("%.4f seconds\n", timeTaken);

printf("%.4f milli-seconds\n", timeTaken \* 100);

}

**//MAIN function**

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

printf("External sort\n");

char inputFileName[MAX\_FILENAME\_LENGTH] = "data5k.txt";

char outputFileName[MAX\_FILENAME\_LENGTH] = "sorted5k.txt";

int bufferSize = 100;

int numInts = 5000;

printf(" Sorting file: %s\n", inputFileName[MAX\_FILENAME\_LENGTH]);

printf(" Output file: %s\n", outputFileName[MAX\_FILENAME\_LENGTH]);

printf(" Buffer size: %d\n", bufferSize);

generateRandomInputFile(inputFileName, numInts);

externalSort(inputFileName, outputFileName, bufferSize);

printf("\Done\n");

return 0;

} //End of main