```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>


#define NAMESIZE 50

/*
===============================================================================================================
=========

                                    COMMENTS ABOUT FUNCTIONS

-Number of lines: This is not a function, it is an integer which is the number of lines of my champions.txt
file.
I found this value in the main using fgetc function. As we have 6 champions, the value of the number of lines
and also
the size of my heap is 6 in this assignment.

-initializeChampions: It takes the name of the data file containing the names of the champions and their
expected win rates.
It also takes the number of lines. It then reads the file and creates an array of data
structure for the champions while initializing them as mentioned above. Finally, it returns the Champions array.

-getBattleData: It takes the Champions array, and the name of the data file containing the
battles' information. It also takes the number of lines. It then computes the number of battles and wins for
every involved
champion as it reads through a battle.

-computeWinRate: It takes the Champions array and the number of lines. Then traverses the array for every
champion and computes the actual win rate and the expectation skew.

-heapSort: It takes the Champions array, number of lines and sorting criteria. It firstly applies the Build Heap
algorithm to create a max-heap and
then sorts the data based on the heap sort algorithm.

-heapify: It takes the Champions array, the root index, the number of lines and the sorting criteria. This
function is used in the "heapSort"
 function to do heapify  my heap.

 -swaps: There are 6 swap operations for each specific champion information. I used those swaps in "heapSort"
and "heapify" functions.

 -printLeaderBoard: It takes the champions array and the number of lines. Then it prints the sorted Champions
leader board showing all
 the respective data for each champion (battles, wins, expected win rate, actual win rate, expectation skew).


===============================================================================================================
=========
*/

struct ChampionInformation
{
    char champName[NAMESIZE];
    float expectedWinRate;
    int numberOfBattles;
    int numberOfWins;
    float actualWinRate;
    float expectationSkew;
};

struct ChampionInformation *initializeChampions(FILE *outfile,int);
void getBattleData(struct ChampionInformation *Champions,FILE *outfile,int);
void computeWinRate(struct ChampionInformation *Champions,int);
void heapify(struct ChampionInformation *Champions,int,int,int);
void heapSort(struct ChampionInformation *Champions,int,int);
void printLeaderBoard(struct ChampionInformation *Champions,int);

void swapAverageWinRate(float*,float*);
void swapChampionsName(char*,char*);
void swapNumberOfBattle(int*,int*);
void swapNumberOfWin(int*,int*);
void swapExpectedWinRate(float*,float*);
void swapExpectationSkew(float*,float*);

int main(int argc, char *argv[])
{
if(argc == 1){
    printf("No argument has been supplied!");
    exit(-2);
}

int lineNum=1;

FILE *champFile, *battleFile;
champFile=fopen(argv[2],"r");
battleFile=fopen(argv[3],"r");

if(champFile==NULL)
    printf("File does not exist!");
if(battleFile==NULL)
    printf("File does not exist!");

char ch;

while((ch = fgetc(champFile))!= EOF)//this loop counts my lineNumbers of champions.txt file
{
    if(ch == '\n')
        lineNum++;
}



struct ChampionInformation *Champions;

Champions=initializeChampions(champFile,lineNum);
getBattleData(Champions,battleFile,lineNum);
computeWinRate(Champions,lineNum);
heapSort(Champions,atoi(argv[1]),lineNum);
printLeaderBoard(Champions,lineNum);

fclose(champFile);
fclose(battleFile);

    return 0;
}

struct ChampionInformation *initializeChampions(FILE *champFile,int lineNum)
{
    char champName[NAMESIZE];
    float EWR;
    int i;

    struct ChampionInformation *Champions;
    Champions=(struct ChampionInformation*)malloc(lineNum*sizeof(struct ChampionInformation));

    if(Champions==NULL)
        printf("allocation failed!");

    fseek(champFile,0L,SEEK_SET);
    for(i=0;((fscanf(champFile,"%s %f\n",champName,&EWR)!=EOF));i++)//here, I take the information from file and
store them in my array of struct//
    {
        strcpy(Champions[i].champName,champName);
        Champions[i].expectedWinRate=EWR;
        Champions[i].numberOfBattles=0;
        Champions[i].numberOfWins=0;
        Champions[i].actualWinRate=0;
        Champions[i].expectationSkew=0;
    }

return Champions;

}
void getBattleData(struct ChampionInformation *Champions, FILE *battleFile,int lineNum)
{
    int i,compareChamp1,compareChamp2,compareWinner;
    char battleID[NAMESIZE], champ1[NAMESIZE], champ2[NAMESIZE], winner[NAMESIZE];

    for(i=0;i<lineNum;i++)
    {
        fseek(battleFile,0L,SEEK_SET);
        while((fscanf(battleFile,"%s %s %s %s\n",battleID,champ1,champ2,winner)!=EOF))
        {
            compareChamp1 = strcmp(champ1,Champions[i].champName);
            compareChamp2 = strcmp(champ2,Champions[i].champName);
            compareWinner = strcmp(winner,Champions[i].champName);

            if(compareChamp1==0 || compareChamp2==0)
                Champions[i].numberOfBattles++;
            if(compareWinner==0)
                Champions[i].numberOfWins++;
        }
    }
}
void computeWinRate(struct ChampionInformation *Champions,int lineNum)
{
    int i;
    float AWR,WRR,ES;

    for(i=0;i<lineNum;i++)
    {
        AWR=(float)Champions[i].numberOfWins/Champions[i].numberOfBattles;
        WRR=AWR/Champions[i].expectedWinRate;
        ES=fabs(WRR-1.00);
        Champions[i].actualWinRate=AWR;
        Champions[i].expectationSkew=ES;
    }
}

void heapSort(struct ChampionInformation *Champions,int sortingCriteria,int lineNum)
{
    int i;
    int heapSize=lineNum; // as my heap size equals to line number of the txt, I equaled them and use heapSize
in the loops

    for (i=(heapSize/2)-1;i>=0;i--) // This loop applies the Build Heap algorithm to create a max-heap
    {
        heapify(Champions,i,heapSize,sortingCriteria);
    }

    for(i=heapSize-1; i>=1; i--) // This loop makes relevant swap operations, then it calls "heapify" function
    {
        swapAverageWinRate(&Champions[0].actualWinRate,&Champions[i].actualWinRate);
        swapChampionsName(Champions[0].champName,Champions[i].champName);
        swapNumberOfBattle(&Champions[0].numberOfBattles,&Champions[i].numberOfBattles);
```

```c
        swapNumberOfWin(&Champions[0].numberOfWins,&Champions[i].numberOfWins);
        swapExpectedWinRate(&Champions[0].expectedWinRate,&Champions[i].expectedWinRate);
        swapExpectationSkew(&Champions[0].expectationSkew,&Champions[i].expectationSkew);
        heapify(Champions,0,i,sortingCriteria);
    }
}

void heapify(struct ChampionInformation *Champions, int rootIndex, int lineNum ,int sortingCriteria)
{
    int left,right,largest;
    int heapSize=lineNum;


    largest=rootIndex;
    left=2*rootIndex+1;
    right=2*rootIndex+2;

    /*
        -Based on the sorting criteria that comes from Heap Sort function, I made relevant
        swap operations
    */

    if(sortingCriteria==1)
    {
        if (left < heapSize && Champions[left].actualWinRate > Champions[largest].actualWinRate)
            largest = left;
        else
            largest=rootIndex;

        if (right < heapSize && Champions[right].actualWinRate > Champions[largest].actualWinRate)
            largest = right;

        if (largest != rootIndex)
        {
            swapAverageWinRate(&Champions[rootIndex].actualWinRate,&Champions[largest].actualWinRate);
            swapChampionsName(Champions[rootIndex].champName,Champions[largest].champName);
            swapNumberOfBattle(&Champions[rootIndex].numberOfBattles,&Champions[largest].numberOfBattles);
            swapNumberOfWin(&Champions[rootIndex].numberOfWins,&Champions[largest].numberOfWins);
            swapExpectedWinRate(&Champions[rootIndex].expectedWinRate,&Champions[largest].expectedWinRate);
            swapExpectationSkew(&Champions[rootIndex].expectationSkew,&Champions[largest].expectationSkew);
            heapify(Champions,largest,heapSize,sortingCriteria);
        }
    }

    else if(sortingCriteria==2)
    {
        if (left < heapSize && Champions[left].expectedWinRate > Champions[largest].expectedWinRate)
            largest = left;
        else
            largest=rootIndex;

        if (right < heapSize && Champions[right].expectedWinRate > Champions[largest].expectedWinRate)
            largest = right;

        if (largest != rootIndex)
        {
            swapAverageWinRate(&Champions[rootIndex].actualWinRate,&Champions[largest].actualWinRate);
            swapChampionsName(Champions[rootIndex].champName,Champions[largest].champName);
            swapNumberOfBattle(&Champions[rootIndex].numberOfBattles,&Champions[largest].numberOfBattles);
            swapNumberOfWin(&Champions[rootIndex].numberOfWins,&Champions[largest].numberOfWins);
            swapExpectedWinRate(&Champions[rootIndex].expectedWinRate,&Champions[largest].expectedWinRate);
            swapExpectationSkew(&Champions[rootIndex].expectationSkew,&Champions[largest].expectationSkew);
            heapify(Champions,largest,heapSize,sortingCriteria);
        }
    }
    else if(sortingCriteria==3)
    {
        if (left < heapSize && Champions[left].expectationSkew > Champions[largest].expectationSkew)
            largest = left;
        else
            largest=rootIndex;

        if (right < heapSize && Champions[right].expectationSkew > Champions[largest].expectationSkew)
            largest = right;

        if (largest != rootIndex)
        {
            swapAverageWinRate(&Champions[rootIndex].actualWinRate,&Champions[largest].actualWinRate);
            swapChampionsName(Champions[rootIndex].champName,Champions[largest].champName);
            swapNumberOfBattle(&Champions[rootIndex].numberOfBattles,&Champions[largest].numberOfBattles);
            swapNumberOfWin(&Champions[rootIndex].numberOfWins,&Champions[largest].numberOfWins);
            swapExpectedWinRate(&Champions[rootIndex].expectedWinRate,&Champions[largest].expectedWinRate);
            swapExpectationSkew(&Champions[rootIndex].expectationSkew,&Champions[largest].expectationSkew);
            heapify(Champions,largest,heapSize,sortingCriteria);
        }
    }
}

void printLeaderBoard(struct ChampionInformation *Champions, int lineNum)
{
    int i;
    printf("\nChampion\tBattles\tWin\tAWR\tEWR\tSkew\t\n");

    for(i=lineNum-1;i>=0;i--)
    {

printf("%s\t\t%d\t%d\t%.2f\t%.2f\t%.2f\n",Champions[i].champName,Champions[i].numberOfBattles,Champions[i].numbe
rOfWins,Champions[i].actualWinRate,Champions[i].expectedWinRate,Champions[i].expectationSkew);
    }
}

void swapAverageWinRate(float* awr1,float* awr2)
{
    float tempAWR;
    tempAWR=*awr1;
    *awr1=*awr2;
    *awr2=tempAWR;
}
void swapChampionsName(char* name1, char* name2)
{
    char *tempNAME;
    tempNAME=(char*)malloc(NAMESIZE*sizeof(char));
    strcpy(tempNAME, name1);
    strcpy(name1, name2);
    strcpy(name2, tempNAME);
    free(tempNAME);
}
void swapNumberOfBattle(int* nob1, int* nob2)
{
    int tempNOB;
    tempNOB=*nob1;
    *nob1=*nob2;
    *nob2=tempNOB;
}
void swapNumberOfWin(int* now1, int* now2)
{
    int tempNOW;
    tempNOW=*now1;
    *now1=*now2;
    *now2=tempNOW;
}
```

```
void swapExpectedWinRate(float* ewr1, float* ewr2)
{
    float tempEWR;
    tempEWR=*ewr1;
    *ewr1=*ewr2;
    *ewr2=tempEWR;
}
void swapExpectationSkew(float* es1, float* es2)
{
    float tempES;
    tempES=*es1;
    *es1=*es2;
    *es2=tempES;
}
```