```cpp
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <cstdlib>

using namespace std;

//general functions
void showMenu();
void tempIntArray(int[], int[], int);
void tempStringArray(string[], string[], int);

//menu functions 1-5
void showData(int[], string[], int[], int[], int);
void sortAscID(int[], string[], int[], int[], int);
void average(int[], int[], int);
void sortScoreOnly(int[], string[], int[], int[], int);
void topScoreByID(int[], string[], int[], int[], int);
void countTopScore(int[], int[], string[], int);


int main()
{
	//array initialization
	const int size = 100;
	//arrays for ID, CPP, JAVA, and a second ID array respectively
	int ORIGINALID[size], ORIGINALCPP[size], ORIGINALJAVA[size],
ORIGINALNAMEID[size];
	//string array for names from student.txt file
	string ORIGINALNAME[size];

	//file initialization
	string fileName_1;
	string fileName_2;

	//user input for file names
	cout << "Enter file name 1  : ";
	cin >> fileName_1;
	cout << "Enter file name 2  : ";
	cin >> fileName_2;

	//ifstream initialization, using two different
	ifstream inStream_1;
	ifstream inStream_2;
	inStream_1.open(fileName_1);
```

```
	inStream_2.open(fileName_2);

	//open file
	if (inStream_1.fail()) // in case of file failure
	{
		cout << "Input file opening failed.\n";
		exit(1);
	}
	if (inStream_2.fail())
	{
		cout << "Input file opening failed.\n";
		exit(1);
	}


	int index_1 = 0;
	//reading student.txt
	//until end of file, instream data into arrays
	while (!inStream_1.eof())
	{
		inStream_1 >> ORIGINALNAMEID[index_1];
		inStream_1 >> ORIGINALNAME[index_1];
		index_1++; //counts how many inputs there are, used also to index
arrays
	}

	int index_2 = 0;
	//reading score.txt
	//until end of file, instream data into arrays
	while (!inStream_2.eof())
	{
		inStream_2 >> ORIGINALID[index_2];
		inStream_2 >> ORIGINALCPP[index_2];
		inStream_2 >> ORIGINALJAVA[index_2];
		index_2++;

	}

	//nested loop that will compare i and j in arrays then
	//swaps names to correct indexes so that ID, Name, and scores align
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			if (ORIGINALID[i] == ORIGINALNAMEID[j]) {
				swap(ORIGINALNAME[j], ORIGINALNAME[i]);
				swap(ORIGINALNAMEID[j], ORIGINALNAMEID[i]);
			}
		}
```

```
	}

	//initializing array copies
	int ID[size], CPP[size], JAVA[size];
	string NAME[size];

	//creates arrays that we can sort and edit without
	//destroying original data
	tempIntArray(ORIGINALID, ID, size);
	tempStringArray(ORIGINALNAME, NAME, size);
	tempIntArray(ORIGINALCPP, CPP, size);
	tempIntArray(ORIGINALJAVA, JAVA, size);

	// variable that lets user choose menu
	// when select changes to 1-6, will show different menus
	// when more changes to anything besides y, will end function.
	int select = 0;
	char more = 'y';

	// do-while loop that will show menu every time y is inputted
	do {
		showMenu();
		cout << endl;
		cout << "Select an option (1~6) : ";

		//loop that will force user to choose a number 1-6
		//if number is not between 1 and 6, then the loop will continue
		do {
			cin >> select;
			if (select < 1 || select > 6)
				cout << "Please select a valid option (1~6) : ";
		} while (select < 1 || select > 6);	//if number is not between 1
and 6, then the loop will continue

		// clear console (googled this one because I saw it on tiktok and I
think it makes the menu a lot cleaner)
		system("cls");

		//switch that will select menu inputted
		switch (select) {

		case 1: //when select = 1 and more = y
			//menu 1 will show original data, we use original arrays
			cout << "\n                    Menu 1                    ";
			cout << "\n-----------------------------------------------" <<
endl;
			showData(ORIGINALID, ORIGINALNAME, ORIGINALCPP, ORIGINALJAVA,
```

```
index_1);
                break;
            case 2: // when select = 2 and more = y
                // menus 2-5 rearrange the data, so we use array copies to
                // sort our data while keeping the originals untouched
                cout << "\n                    Menu 2                    ";
                cout << "\n-----------------------------------------------" <<
endl;
                sortAscID(ID, NAME, CPP, JAVA, index_1);
                break;
            case 3: //when select = 3 and more = y
                cout << "\n                    Menu 3                    ";
                cout << "\n-----------------------------------------------" <<
endl;
                average(ORIGINALCPP, ORIGINALJAVA, index_1);
                break;
            case 4: //when select = 4 and more = y
                cout << "\n                    Menu 4                    ";
                cout << "\n-----------------------------------------------" <<
endl;
                sortScoreOnly(ID, NAME, CPP, JAVA, index_1);
                showData(ID, NAME, CPP, JAVA, index_1);
                break;
            case 5: //when select = 5 and more = y

                cout << "\n                    Menu 5                    ";
                cout << "\n-----------------------------------------------" <<
endl;

                cout << "\n         Top Scores C++        ";
                cout << "\n==============================" << endl;
                topScoreByID(ID, NAME, CPP, JAVA, index_1);
                countTopScore(CPP, ID, NAME, index_1);

                cout <<
"\n============================================================" << endl;

                cout << "\n        Top Scores Java        ";
                cout << "\n==============================" << endl;
                topScoreByID(ID, NAME, JAVA, CPP, index_1);
                countTopScore(JAVA, ID, NAME, index_1);
                break;

            case 6: // exit function
                cout << "Exited! " << endl;
                return 0;
            }
```

```
			//give the user the option to return to the menu, or exit
			cout << "Return to menu ? (y/n) : ";
			cin >> more;

			system("cls"); // clear console again so that the menu shows on its
own without the
			// other menus getting in the way

		} while (more == 'y' || more == 'Y');

		return 0;
}

//general functions
void showMenu()
{
		//menu interface
		cout << "\n-----------------------------------------------";
		cout << "\n                      Menu                     ";
		cout << "\n-----------------------------------------------";
		cout << "\n1. Show original data";
		cout << "\n2. Show data in order of ascending ID";
		cout << "\n3. Show average for each course";
		cout << "\n4. Show C++ score in descending order";
		cout << "\n5. Show top three scores for C++ and Java tests";
		cout << "\n6. Exit ";
}

//A[] is the original array
//B[] is an empty array copy
//function will copy integers to the array copy
//so we can edit data and still display original data after
void tempIntArray(int A[], int B[], int size)
{
		for (int i = 0; i < size; i++) {
			B[i] = A[i];
		}
}

//same as above but for string
//A[] is the original array
//B[] is an empty array copy
//function will copy string to the array copy
//so we can edit data and still display original data after
void tempStringArray(string A[], string B[], int size)
{
```

```
	for (int i = 0; i < size; i++) {
		B[i] = A[i];
	}
}


//array print
//A[] is ID
//B[] is NAME
//C[] and D[] are score data
//size is how many units there are in the file
void showData(int A[], string B[], int C[], int D[], int size)
{
	//loop goes through each index of each array and outputs it
	//different setw are used to align the data properly
	for (int i = 0; i < siAze; i++) {
		cout << setw(3) << A[i] << setw(12) << B[i] << setw(4) << C[i] <<
setw(4) << D[i] << endl;
	}
}

//menu 2
//A[] is ID
//B[] is NAME
//C[] and D[] are score data
//size is how many units there are in the file
void sortAscID(int A[], string B[], int C[], int D[], int size)
{
	//bubble sort for ID, swaps all arrays at the same index
	//if A[j] is greater than the number next in order, they swap places
	//this puts the highest data value at the end, sorting from least to
greatest (ascending)
	for (int i = 0; i < size - 1; i++) {
		for (int j = 0; j < size - i - 1; j++) {
			if (A[j] > A[j + 1]) {
				swap(A[j], A[j + 1]);
				swap(B[j], B[j + 1]);
				swap(C[j], C[j + 1]);
				swap(D[j], D[j + 1]);
			}
		}
	}
	showData(A, B, C, D, size);
}

//menu 3
//A[] is CPP data
```

```
//B[] is JAVA data
//size is the number of files in the function
//function will calculate the average
void average(int A[], int B[], int size)
{
	double averageCpp = 0;
	double averageJava = 0;

	//adds all the scores together
	//every i, the variables are incremented by the score.
	for (int i = 0; i < size; i++) {
		averageCpp += A[i];
		averageJava += B[i];
	}

	//divides the all the scores added up by the number of things in the file
	//gives the average
	averageCpp = averageCpp / size;
	averageJava = averageJava / size;

	//show up to two decimal places
	cout.setf(ios::fixed);
	cout.setf(ios::showpoint);
	cout.precision(2);


	cout << "Average of C++ : " << averageCpp << endl;
	cout << "Average of Java : " << averageJava << endl;
}

//menu 4
//A[] is ID
//B[] is NAME
//C[] and D[] are score data
//C[] is the primary score, and after C[] is sorted, D[] will be sorted as the
secondary score
// depending on if CPP or JAVA needs to be sorted first
//size is how many units there are in the file
void sortScoreOnly(int A[], string B[], int C[], int D[], int size)
{
	//bubble sort for main score
	//if C[j] is less than the number next in order, they swap places
	//this puts the lowest data value at the end, sorting from greatest to least
(descending)
	for (int i = 0; i < size - 1; i++) {
		//we swap ALL arrays to keep data together
		for (int j = 0; j < size - i - 1; j++) {
```

```
                        if (C[j] < C[j + 1]) {
                                swap(A[j], A[j + 1]);
                                swap(B[j], B[j + 1]);
                                swap(C[j], C[j + 1]);
                                swap(D[j], D[j + 1]);
                        }
                }
        }
        //bubble sort for secondary score if main scores match
        //if C[j] matches the thing next to it
        //compare the D[j] scores and sort descending so that the highest scorer is
listed
        for (int i = 0; i < size - 1; i++) {
                //we swap ALL arrays to keep data together
                for (int j = 0; j < size - i - 1; j++) {
                        if (C[j] == C[j + 1] && D[j] < D[j + 1]) {
                                swap(A[j], A[j + 1]);
                                swap(B[j], B[j + 1]);
                                swap(C[j], C[j + 1]);
                                swap(D[j], D[j + 1]);
                        }
                }
        }

}

//menu 5
//A[] is ID
//B[] is NAME
//C[] and D[] are score data; //C[] is the primary score that will be sorted
//size is the number of units in the file
void topScoreByID(int A[], string B[], int C[], int D[], int size)
{
        //organizes ID from greatest to least so that countTopScore
        //function can print backwards ((see countTopScoreFunction))
        //uses the sortScoreOnly function to sort C[] in descending order
        sortScoreOnly(A, B, C, D, size);
        for (int i = 0; i < size - 1; i++) {
                //goes through each score, and if two scores are the same
                //nested loop will bubble sort by ID in descending order
                for (int j = 0; j < size - i - 1; j++) {
                        if (C[j] == C[j + 1] && A[j] < A[j + 1]) {
                                swap(A[j], A[j + 1]);
                                swap(B[j], B[j + 1]);
                                swap(C[j], C[j + 1]);
                                swap(D[j], D[j + 1]);
                        }
```

```
			}
		}
}

//menu 5
//A[] is the primary score being sorted
//B[] is the ID
//C[] is the name
//size is the number of units in the file
void countTopScore(int A[], int B[], string C[], int size)
{
		//these variables are used as limiters so that our function does not
		//produce an error
		//count will count how many people get the same score so that they can be
grouped together
		//max will stop outputting score groups and ranks once it hits 3
		int count = 1;
		int max = 0;

		//function counts how many of the same score to output, then
		//prints that many indicies in the array, starting at i, the counting
backwards
		//until the scores compared are different for A[i] and A[i+1], count will
continue
		for (int i = 0; i < size; i++) {
			if (A[i] == A[i + 1]) {
				count++;
			}
			else
			{
				cout << "[ " << A[i] << " (" << count << " student(s)) ]" <<
endl;
				//since loop goes backwards in the array, the scores are
ordered greatest
				//to least so they print least to greatest
				for (int j = i; j >= i - count + 1; j--) {
					if (count == size) {
						//if there are as many top scores as there are
scores, then there is no top rank
						//since everyone gets the same score

						cout << "\nThere is no top rank. " << endl;

					}
					else
						cout << "<" << B[j] << "> " << setw(9) << C[j] <<
" " << endl;
```

```
                        }
                        count = 1;
                        cout << endl;
                        ++max;
                        if (max == 3) // only prints the top three scores
                              break;
                  }
            }
}
```