```cpp
#include "Employee.h"

const int RECORD = 40;


Employee::Employee() : employee_id(""), lastName(""), firstName(""), address(""),
                       city(""), province(""), postalCode(""), phone(""), gender('0'),
                       age(-1), dependents(-1), dept(""), uni('0'), hourlyRate(0.0) {}

Employee::~Employee() {}

// assign values to the Employee objects
Employee Employee::operator<<(string data) {

    stringstream ss;
    ss << data;

    getline(ss, employee_id, '\t');
    getline(ss, lastName, '\t');
    getline(ss, firstName, '\t');
    getline(ss, address, '\t');
    getline(ss, city, '\t');
    getline(ss, province, '\t');
    getline(ss, postalCode, '\t');
    getline(ss, phone, '\t');
    ss >> gender;
    ss >> age;
    ss >> dependents;
    ss >> dept;
    ss >> uni;
    ss >> hourlyRate;

    return *this;
}

// override << operator
ostream& operator<<(ostream& output, Employee& emp) {

    output << emp.employee_id << "\t" << emp.lastName << "\t" << emp.firstName << "\t" <<
emp.address << "t" << emp.city <<
           "\t" << emp.province << "\t" << emp.postalCode << "\t" << emp.phone << "\t" <<
emp.gender << "\t" << emp.age << "\t" <<
           emp.dependents << "\t" << emp.dept << "\t" << emp.uni << "\t" << emp.hourlyRate
 << endl;

    return output;

}

// write array elements to a file
void Employee::SaveToFile(Employee* emp) {

    ofstream outputFile;
    string outPath = "..\\output\\Output.txt";
    outputFile.open(outPath);

    for (int i = 0; i < RECORD; i++) {
        outputFile << emp[i];
    }

    outputFile << "\n";

}
```

```cpp
#include <iostream>
#include <fstream>
#include <sstream>

using namespace std;

#ifndef ASSIGNMENT4_EMPLOYEE_H
#define ASSIGNMENT4_EMPLOYEE_H

class Employee {

public:

    string employee_id;
    string lastName;
    string firstName;
    string address;
    string city;
    string province;
    string postalCode;
    string phone;
    char gender;
    int age;
    int dependents;
    string dept;
    char uni;
    double hourlyRate;

    Employee();
    virtual ~Employee();

    Employee operator<<(string data);
    friend ostream& operator<<(ostream& output, Employee& emp);
    void SaveToFile(Employee* emp);

};

#endif //ASSIGNMENT4_EMPLOYEE_H
```

File - C:\NSCC\2nd Year\2nd Semester\PROG2400 - Data Structure\assignment-4-Paraga2mp\src\main.cpp

```cpp
#include <iostream>
#include <sstream>
#include "MergeSort.h"

using namespace std;

const int RECORD = 40;

// get the field number from command line and convert to an integer
int GetFieldNo(string fieldNo) {

    string initialStr = "-field=";
    string num = "";
    int number;

    for(int i = 0; i < fieldNo.length(); i++) {

        if(i < initialStr.length()) {
            // check if initialStr id equal to fieldNo
            if(initialStr[i] != fieldNo[i]) {

                cout << "Wrong field no" << endl;
            }
        }
        else {
            // check if fieldNo is digit
            if(!isalpha(fieldNo[i])) {

                num += fieldNo[i];
            }
            else {

                cout << "Wrong field no" << endl;
            }
        }
    }
    // convert string num to integer value
    if((num != "") && stoi(num) >= 0 && stoi(num) <= 13) {

        number = stoi(num);
    }
    else {

        cout << "Wrong field number" << endl;
    }

    return number;
}



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

    string fileName;
    string line;

    //gets the file name from the main function argument
    if(argc > 1) {

        fileName = argv[1];
    }

    // get and convert the field number from command line
    string field = argv[2];
    int fieldNo = GetFieldNo(field);

    Employee emp;
    Employee obj[RECORD];
    MergeSort mst;
    int i = 0;
```

```
    //stream to read from the file
    fstream readFile(fileName);

    if(readFile.is_open()) {
        // read the 1st line (heading) from the file
        getline(readFile, line);

        while (getline(readFile, line)) {
            // read and store the file content into an array
            emp << line;
            obj[i++] << line;
        }
    }
    else {

        cout << "Unable to open the file" << endl;
    }

    // call the Mergesort function
    mst.Mergesort(obj, RECORD, fieldNo);

    // call the function to write into a file
    emp.SaveToFile(obj);


    return 0;
}
```

```cpp
#include "MergeSort.h"

// compare Employee's object members value, appropriate member value
// is less than equal to or not
bool MergeSort::IsLessThanEqualTo (Employee &emp1, Employee &emp2, int index) {

    switch (index) {

        case 0:
            return (emp1.employee_id <= emp2.employee_id);
        case 1:
            return (emp1.lastName <= emp2.lastName);
        case 2:
            return (emp1.firstName <= emp2.firstName);
        case 3:
            return (emp1.address <= emp2.address);
        case 4:
            return (emp1.city <= emp2.city);
        case 5:
            return (emp1.province <= emp2.province);
        case 6:
            return (emp1.postalCode <= emp2.postalCode);
        case 7:
            return (emp1.phone <= emp2.phone);
        case 8:
            return (emp1.gender <= emp2.gender);
        case 9:
            return (emp1.age <= emp2.age);
        case 10:
            return (emp1.dependents <= emp2.dependents);
        case 11:
            return (emp1.dept <= emp2.dept);
        case 12:
            return (emp1.uni <= emp2.uni);
        case 13:
            return (emp1.hourlyRate <= emp2.hourlyRate);
    }

    return false;
}

// compare Employee's object members value, appropriate member value
// is greater than equal to or not
bool MergeSort::IsGreaterThan (Employee &emp1, Employee &emp2, int index) {

    switch (index) {

        case 0:
            return (emp1.employee_id > emp2.employee_id);
        case 1:
            return (emp1.lastName > emp2.lastName);
        case 2:
            return (emp1.firstName > emp2.firstName);
        case 3:
            return (emp1.address > emp2.address);
        case 4:
            return (emp1.city > emp2.city);
        case 5:
            return (emp1.province > emp2.province);
        case 6:
            return (emp1.postalCode > emp2.postalCode);
        case 7:
            return (emp1.phone > emp2.phone);
        case 8:
            return (emp1.gender > emp2.gender);
        case 9:
            return (emp1.age > emp2.age);
        case 10:
            return (emp1.dependents > emp2.dependents);
        case 11:
            return (emp1.dept > emp2.dept);
```

```
        case 12:
            return (emp1.uni > emp2.uni);
        case 13:
            return (emp1.hourlyRate > emp2.hourlyRate);
    }

    return false;
}

// split the employee objects into two arrays
void MergeSort::Split(Employee *inarray, int len, Employee *outarray1, int *outsize1,
                      Employee *outarray2, int *outsize2, int index) {

    int subFiles = 0;
    int in = 0;
    *outsize1 = 0;
    *outsize2 = 0;
    string strValue;
    Employee curr;
    Employee next = inarray[in++];

    // while there are objects available, keep looping
    while(in <= len) {

        // extract one ordered sublist and put into correct array
        do {
            curr = next;
            // get the next record
            if(in < len) {
                next = inarray[in];
            }
            in++;

            // put objects in the correct array
            if(subFiles % 2 == 0) {
                outarray1[(*outsize1)++] = curr;
            }
            else {
                outarray2[(*outsize2)++] = curr;
            }
            // keep looping as long as the current objects index value
            // is less than next object's index value
        } while((in <= len) && (IsLessThanEqualTo(curr , next, index)));

        subFiles++;
    }
}

// merge the two arrays into an ordered array
int MergeSort::Merge(Employee *outarray, int *outsize, Employee *inarray1, int insize1,
                     Employee *inarray2, int insize2, int index) {

    int subFiles = 0;
    int in1 = 0;
    int in2 = 0;
    *outsize = 0;
    Employee curr1;
    Employee curr2;
    Employee prev1;
    Employee prev2;

    // start at the beginning of each subarray
    curr1 = inarray1[in1++];
    curr2 = inarray2[in2++];

    // keep going while both arrays have elements
    while(in1 <= insize1 && in2 <= insize2) {

        bool endOfSub1 = false;
        bool endOfSub2 = false;
```

```
        // move through both arrays until an ordered sublist is completed
        while(!endOfSub1 && !endOfSub2) {

            // use the element from array 1, if it is lower
            if(IsLessThanEqualTo(curr1 , curr2, index)) {

                // store the element in the merged array
                outarray[(*outsize)++] = curr1;

                // move to next object
                prev1 = curr1;
                if(in1 < insize1) {

                    curr1 = inarray1[in1];
                }
                in1++;

                // check the end of the sublist
                if((in1 > insize1) || IsGreaterThan(prev1 , curr1, index)) {

                    endOfSub1 = true;
                }
            }

            // use the object from array 2, if it is lower
            if(IsLessThanEqualTo(curr2 , curr1, index)) {

                // store the object in the merged array
                outarray[(*outsize)++] = curr2;

                // move to next object
                prev2 = curr2;
                if(in2 < insize2) {

                    curr2 = inarray2[in2];
                }
                in2++;

                // check end of the sublist
                if((in2 > insize2) || IsGreaterThan(prev2, curr2, index)) {

                    endOfSub2 = true;
                }
            }
        }

        // we finished with array 2, dump the remaining ordered
        // elements from array 1 into the merged array
        while(!endOfSub1) {

            // add to merged array
            outarray[(*outsize)++] = curr1;

            // move to next object
            prev1 = curr1;
            if(in1 < insize1) {

                curr1 = inarray1[in1];
            }
            in1++;

            // check for the end of the sublist
            if((in1 > insize1) || IsGreaterThan(prev1, curr1, index)) {

                endOfSub1 = true;
            }
        }

        // we finished with array 1, dump the remaining ordered
        // elements from array 2 into the merged array
        while(!endOfSub2) {
```

```
            // add to merged array
            outarray[(*outsize)++] = curr2;

            // move to next object
            prev2 = curr2;
            if(in2 < insize2) {

                curr2 = inarray2[in2];
            }
            in2++;

            // check for the end of the sublist
            if((in2 > insize2) || IsGreaterThan(prev2, curr2, index)) {

                endOfSub2 = true;
            }
        }

        // move to the next sublist
        subFiles++;
    }

    // dump remaining objects from array 1
    while(in1 <= insize1) {

        outarray[(*outsize)++] = curr1;

        // move to next object
        prev1 = curr1;
        if(in1 < insize1) {

            curr1 = inarray1[in1];
        }
        in1++;

        //keep track of sublists within the remaining elements
        if((in1 > insize1) || IsGreaterThan(prev1, curr1, index)) {

            subFiles++;
        }
    }

    // dump the remaining objects from array 2
    while(in2 <= insize2) {

        outarray[(*outsize)++] = curr2;

        // move to next object
        prev2 = curr2;
        if(in2 < insize2) {

            curr2 = inarray2[in2];
        }
        in2++;

        // keep track of sublists within the remaining elements
        if((in2 > insize2) || IsGreaterThan(prev2, curr2, index)) {

            subFiles++;
        }
    }

    return subFiles;
}

// the function perform mergesort, by merging and splitting elements
void MergeSort::Mergesort(Employee* array, int len, int index) {

    int subFiles = 0;
    int tempsize1;
```

```
    int tempsize2;

    Employee *temp1 = new Employee[len];
    Employee *temp2 = new Employee[len];


    // keep splitting and merging until we have 1 subfile
    do {
        Split(array, len, temp1, &tempsize1, temp2, &tempsize2, index);

        subFiles = Merge(array, &len, temp1, tempsize1, temp2, tempsize2, index);

    }while(subFiles != 1);

    delete [] temp1;
    delete [] temp2;
}
```

```
#ifndef ASSIGNMENT4_MERGESORT_H
#define ASSIGNMENT4_MERGESORT_H

#include "Employee.h"

class MergeSort {

public:

    bool IsLessThanEqualTo (Employee &emp1, Employee &emp2, int index);

    bool IsGreaterThan (Employee &emp1, Employee &emp2, int index);

    void Split(Employee *inarray, int len, Employee *outarray1, int *outsize1,
               Employee *outarray2, int *outsize2, int index);

    int Merge(Employee *outarray, int *outsize, Employee *inarray1, int insize1,
              Employee *inarray2, int insize2, int index);

    void Mergesort(Employee* array, int len, int index);

};


#endif //ASSIGNMENT4_MERGESORT_H
```