323-34 Project 3: Radix Sort **C++**

Student: Seratul Ambia

Project Due Date: 2/28/2021

Algorithm's Steps:

1. The first step is to create the first reading. All it is doing is creating an offset and shows the highest number of digits you have in the radix sort.
2. You want to load the data into a stack. This will help you input all the data into the hashtable to be sorted.
3. Once you have put the data into the hashtables, you use rSort to sort all the data based on the value of each of their digits. The algorithm will be able to use radix sort and compare the digits to put into their respective buckets.
4. Everytime a new digit is looked at, it is sent to a different table until all the digits have been looked at. By the end, they should all be sorted.

Source Code:

```
#include <iostream>
#include <fstream>
#include <string>


using namespace std;

class listNode
{
public:
   int data;
   listNode *next;
   listNode()
   {
      next = NULL;
   }
```

```
    listNode(int d)
    {
        data = d;
        next = NULL;
    }
    void printNode(listNode *node, ofstream &outFile)
    {
        if (node->next == NULL)
        {
            outFile << "(" << node->data << ", NULL) --> NULL" << endl;
        }
        else
        {
            outFile << "(" << node->data << "," << node->next->data << ") --> ";
        }
    }
};

class LLStack
{
public:
    friend class RadixSort;
    listNode *top = NULL;
    listNode *dummy = new listNode(-9999);

    LLStack()
    {
        top = dummy;
    }

    void push(listNode *d)
    {
        d->next = top->next;
        top->next = d;
    }

    listNode *pop()
    {
        if (top->next == NULL)
        {
```

```
            cout << "Cannot remove from empty stack." << endl;
            return NULL;
        }
        else
        {
            listNode *temp;
            temp = top->next;
            top->next = top->next->next;
            temp->next = NULL;
            return temp;
        }
    }

    bool isEmpty()
    {
        if (top->next == NULL)
            return true;
        else
            return false;
    }



    void printStack(LLStack &S, ofstream &outFile1)
    {
        outFile1 << "Top --> ";
        listNode *temp = S.top;

        while (temp != NULL)
        {
            temp->printNode(temp, outFile1);

            temp = temp->next;
        }
    }
};

class LLQueue
{
public:
```

```
friend class RadixSort;
listNode *head;
listNode *tail;
listNode *dummy = new listNode(-9999);

LLQueue()
{
   head = dummy;
   tail = head;
}

void insertQ(listNode *node)
{
   if (head->next == NULL)
   {
      head->next = node;
      tail = node;
   }
   else
   {
      tail->next = node;
      tail = node;
   }
}
listNode *deleteQ()
{

   if (head->next== NULL)
   {
      cout << "Cannot remove from empty stack." << endl;
      tail = head;
      return NULL;
   }
   else
   {
      listNode *temp = new listNode();
      temp = head->next;


      head->next = head->next->next;
```

```
        temp->next = NULL;

        return temp;
    }
}
bool isEmpty()
{
    if (head->next == NULL)
        return true;
    else
        return false;
}
void buildQueue(ifstream &inFile, ofstream &outFile2)
{
    char op;
    int data;

    listNode *junk;

    while (!inFile.eof())
    {

        inFile >> op >> data;

        listNode *temp = new listNode(data);
        if (op == '+')
        {
            insertQ(temp);
        }
        else if (op == '-')
        {
            junk = deleteQ();
            free(junk);
        }
        else
        {
            outFile2 << "The Stack is Empty" << endl;
        }
    }
```

```
    }
    void printQueue(int whichTable, int index, ofstream& outFile2){
        outFile2 << "Table[" << whichTable << "][" << index << "]: ";

        listNode* t;
        t = head;

        while(t != tail){
            t->printNode(t, outFile2);
            t = t->next;

        }
        t->printNode(t,outFile2);
        outFile2 << endl;

    }
};



class RadixSort {
public:
    int tableSize = 10;
    LLQueue ***HashTable;
    int currentTable = 0;
    int previousTable = 1;
    int numDigits;
    int offSet;
    int currentPosition;
    int currentDigit = 0;
    int currentQueue;

    RadixSort() {
        HashTable = new LLQueue **[1];

        for (int i = 0; i <= 1; ++i) {
            HashTable[i] = new LLQueue *[9];

            for (int j = 0; j <= 9; ++j) {
                HashTable[i][j] = new LLQueue();
```

```
        }
    }
}

void firstReading (ifstream &inFile, ofstream &outFile2) {
    int data;
    int negativeNum = 0;
    int positiveNum = 0;
    outFile2 << "*** Performing firstReading" << endl;

    while (!inFile.eof()) {
        inFile >> data;

        if (data < negativeNum) {
            negativeNum = data;

        }
        if (data > positiveNum) {
            positiveNum = data;

        }
    }
    if (negativeNum < 0) {
        offSet = abs(negativeNum);
    }
    else {
        offSet = 0;
    }

    positiveNum += offSet;
    numDigits = getLength(positiveNum);
    outFile2 << "The biggest positive integer is: " << positiveNum - offSet << endl;
    outFile2 << "The biggest negative integer is: " << negativeNum << endl;
    outFile2 << "The offset is: " << offSet << endl;
    outFile2 << "The biggest digit of a number is: " << numDigits << endl;

}
int getLength (int data) {
    string str = to_string(data);
    return str.length();
```

```
}

LLStack * loadStack(ifstream &inFile1, ofstream &outFile2) {
   int data;
   outFile2 << "*** Performing loadStack" << endl;
   LLStack temp;
   LLStack* S = new LLStack();

   while (!inFile1.eof()) {
      inFile1 >> data;
      data += offSet;
      listNode *newNode = new listNode(data);
      S->push(newNode);
   }

   return S;
}

void RSort (LLStack S, ofstream& outFile1, ofstream& outFile2) {
   outFile2 << "*** Performing Rsort" << endl;
   currentPosition = 0;
   currentTable = 0;
   outFile2 << "*** Performing moveStack" << endl;
   moveStack(S, currentPosition, currentTable);
   printTable(currentTable, outFile2);
   currentPosition++;
   currentTable = 1;
   previousTable = 0;
   currentQueue = 0;
   listNode *newNode = new listNode();
   int hashIndex;

   while (currentPosition < numDigits) {
      while (currentQueue <= tableSize -1) {
         while (HashTable[previousTable][currentQueue] ->head->next != NULL) {
            newNode = HashTable[previousTable][currentQueue]->deleteQ();
            hashIndex = getDigit(newNode->data, currentPosition);
            HashTable[currentTable][hashIndex]->insertQ(newNode);
         }
         currentQueue++;
```

```
        }
        printTable(currentTable, outFile2);
        previousTable = currentTable;
        currentTable = (currentTable + 1) %2;
        currentQueue = 0;
        currentPosition++;

    }
    printSortedData(previousTable, outFile1);

}

void moveStack (LLStack S, int currentPosition, int currentTable) {

    while (!S.isEmpty()) {
        listNode *newNode = S.pop();
        int hashIndex = getDigit(newNode->data, currentPosition);
        HashTable[currentTable][hashIndex]->insertQ(newNode);
    }
}

int getDigit(int data, int currentPosition) {
    if (data == 0) {
        return 0;
    }
    else {

        string str = to_string(data);
        if (numDigits == 4) {

            if (str.length() == 3) {
                str = "0" + str;
            }
            if (str.length() == 2) {
                str = "00" + str;
            }
            if (str.length() == 1) {
                str = "000" + str;
            }
        }
```

```
        if (numDigits == 3) {

            if (str.length() == 2) {
                str = "0" + str;
            }
            if (str.length() == 1) {
                str = "00" + str;
            }

        }

        string holder;
        holder = str[str.length() - (currentPosition + 1)];
        data = stoi(holder);
        return data;
    }
}

void printTable(int whichTable, ofstream& outFile2){
    outFile2 << "Printing Table: " << endl;
    for(int i=0; i<10;i++){
        if(!HashTable[whichTable][i]->isEmpty()){
            HashTable[whichTable][i]->printQueue(whichTable, i, outFile2);
        }
    }
    outFile2 << endl;
}

void printSortedData(int whichTable, ofstream& outFile1){
    outFile1 << "Sorted Data: " << endl;

    for(int i = 0; i <= 9;i++){

        while(!HashTable[whichTable][i]->isEmpty()){

            while(HashTable[whichTable][i]->head->next != NULL){
                listNode* newNode = HashTable[whichTable][i]->deleteQ();


                outFile1 << newNode->data - offSet << endl;
```

```
            }

        }
    }
}


};

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

    string inputName = argv[1];
    ifstream input;
    input.open(inputName);

    string outputName1 = argv[2];
    ofstream output1;
    output1.open(outputName1);

    string outputName2 = argv[3];
    ofstream output2;
    output2.open(outputName2);



    RadixSort input1;
    input1.firstReading(input, output1);
    input.seekg(0, ios::beg); // makes it so you don't need to close, and you can just go back to the
beginning.

    LLStack *S = input1.loadStack(input,output1);
    S->printStack(*S, output1);
    input1.RSort(*S, output1, output2);



    input.close();
    output1.close();
    output2.close();

    return 0;
```

}

OutFiles:

OUTFILE 1 FROM DATA 1:

*** Performing firstReading
The biggest positive integer is: 999
The biggest negative integer is: 0
The offset is: 0
The biggest digit of a number is: 3
*** Performing loadStack
Top --> (-9999,388) --> (388,971) --> (971,40) --> (40,6) --> (6,61) --> (61,95) --> (95,8) --> (8,702) --> (702,816) --> (816,95) --> (95,4) --> (4,49) --> (49,22) --> (22,91) --> (91,48) --> (48,123) --> (123,85) --> (85,64) --> (64,67) --> (67,730) --> (730,296) --> (296,538) --> (538,37) --> (37,714) --> (714,653) --> (653,152) --> (152,999) --> (999,18) --> (18,402) --> (402,328) --> (328,191) --> (191, NULL) --> NULL
Sorted Data:
4
6
8
18
22
37
40
48
49
61
64
67
85
91
95
95
123
152
191
296
328
388

402
538
653
702
714
730
816
971
999

OUTFILE 2 FROM DATA 1:

*** Performing Rsort
*** Performing moveStack
Printing Table:
Table[0][0]: (-9999,40) --> (40,730) --> (730, NULL) --> NULL

Table[0][1]: (-9999,971) --> (971,61) --> (61,91) --> (91,191) --> (191, NULL) --> NULL

Table[0][2]: (-9999,702) --> (702,22) --> (22,152) --> (152,402) --> (402, NULL) --> NULL

Table[0][3]: (-9999,123) --> (123,653) --> (653, NULL) --> NULL

Table[0][4]: (-9999,4) --> (4,64) --> (64,714) --> (714, NULL) --> NULL

Table[0][5]: (-9999,95) --> (95,95) --> (95,85) --> (85, NULL) --> NULL

Table[0][6]: (-9999,6) --> (6,816) --> (816,296) --> (296, NULL) --> NULL

Table[0][7]: (-9999,67) --> (67,37) --> (37, NULL) --> NULL

Table[0][8]: (-9999,388) --> (388,8) --> (8,48) --> (48,538) --> (538,18) --> (18,328) --> (328, NULL) --> NULL

Table[0][9]: (-9999,49) --> (49,999) --> (999, NULL) --> NULL

Printing Table:
Table[1][0]: (-9999,702) --> (702,402) --> (402,4) --> (4,6) --> (6,8) --> (8, NULL) --> NULL

Table[1][1]: (-9999,714) --> (714,816) --> (816,18) --> (18, NULL) --> NULL

Table[1][2]: (-9999,22) --> (22,123) --> (123,328) --> (328, NULL) --> NULL

Table[1][3]: (-9999,730) --> (730,37) --> (37,538) --> (538, NULL) --> NULL

Table[1][4]: (-9999,40) --> (40,48) --> (48,49) --> (49, NULL) --> NULL

Table[1][5]: (-9999,152) --> (152,653) --> (653, NULL) --> NULL

Table[1][6]: (-9999,61) --> (61,64) --> (64,67) --> (67, NULL) --> NULL

Table[1][7]: (-9999,971) --> (971, NULL) --> NULL

Table[1][8]: (-9999,85) --> (85,388) --> (388, NULL) --> NULL

Table[1][9]: (-9999,91) --> (91,191) --> (191,95) --> (95,95) --> (95,296) --> (296,999) --> (999, NULL) --> NULL

Printing Table:
Table[0][0]: (-9999,4) --> (4,6) --> (6,8) --> (8,18) --> (18,22) --> (22,37) --> (37,40) --> (40,48) --> (48,49) --> (49,61) --> (61,64) --> (64,67) --> (67,85) --> (85,91) --> (91,95) --> (95,95) --> (95, NULL) --> NULL

Table[0][1]: (-9999,123) --> (123,152) --> (152,191) --> (191, NULL) --> NULL

Table[0][2]: (-9999,296) --> (296, NULL) --> NULL

Table[0][3]: (-9999,328) --> (328,388) --> (388, NULL) --> NULL

Table[0][4]: (-9999,402) --> (402, NULL) --> NULL

Table[0][5]: (-9999,538) --> (538, NULL) --> NULL

Table[0][6]: (-9999,653) --> (653, NULL) --> NULL

Table[0][7]: (-9999,702) --> (702,714) --> (714,730) --> (730, NULL) --> NULL

Table[0][8]: (-9999,816) --> (816, NULL) --> NULL

Table[0][9]: (-9999,971) --> (971,999) --> (999, NULL) --> NULL

OUTFILE 1 FROM DATA 2:

*** Performing firstReading
The biggest positive integer is: 999
The biggest negative integer is: -532
The offset is: 532
The biggest digit of a number is: 4
*** Performing loadStack
Top --> (-9999,1531) --> (1531,1145) --> (1145,1338) --> (1338,1129) --> (1129,946) --> (946,865) --> (865,539) --> (539,1217) --> (1217,810) --> (810,1208) --> (1208,1442) --> (1442,537) --> (537,524) --> (524,1234) --> (1234,588) --> (588,623) --> (623,539) --> (539,1227) --> (1227,571) --> (571,548) --> (548,437) --> (437,541) --> (541,1071) --> (1071,856) --> (856,1287) --> (1287,575) --> (575,1208) --> (1208,893) --> (893,1272) --> (1272,541) --> (541,626) --> (626,557) --> (557,0) --> (0,659) --> (659,758) --> (758,540) --> (540,1356) --> (1356,538) --> (538,1353) --> (1353,555) --> (555,745) --> (745,1014) --> (1014,546) --> (546,1440) --> (1440,942) --> (942,666) --> (666,534) --> (534, NULL) --> NULL
Sorted Data:
-532
-95
-8
2
5
6
7
7
8
9
9
14
16
23
25
39
43
56
91
94

127
134
213
226
278
324
333
361
410
414
482
539
597
613
676
676
685
695
702
740
755
806
821
824
908
910
999

OUTFILE 2 FROM DATA 2

*** Performing Rsort
*** Performing moveStack
Printing Table:
Table[0][0]: (-9999,810) --> (810,0) --> (0,540) --> (540,1440) --> (1440, NULL) --> NULL

Table[0][1]: (-9999,1531) --> (1531,571) --> (571,541) --> (541,1071) --> (1071,541) --> (541, NULL) --> NULL

Table[0][2]: (-9999,1442) --> (1442,1272) --> (1272,942) --> (942, NULL) --> NULL

Table[0][3]: (-9999,623) --> (623,893) --> (893,1353) --> (1353, NULL) --> NULL

Table[0][4]: (-9999,524) --> (524,1234) --> (1234,1014) --> (1014,534) --> (534, NULL) --> NULL

Table[0][5]: (-9999,1145) --> (1145,865) --> (865,575) --> (575,555) --> (555,745) --> (745, NULL) --> NULL

Table[0][6]: (-9999,946) --> (946,856) --> (856,626) --> (626,1356) --> (1356,546) --> (546,666) --> (666, NULL) --> NULL

Table[0][7]: (-9999,1217) --> (1217,537) --> (537,1227) --> (1227,437) --> (437,1287) --> (1287,557) --> (557, NULL) --> NULL

Table[0][8]: (-9999,1338) --> (1338,1208) --> (1208,588) --> (588,548) --> (548,1208) --> (1208,758) --> (758,538) --> (538, NULL) --> NULL

Table[0][9]: (-9999,1129) --> (1129,539) --> (539,539) --> (539,659) --> (659, NULL) --> NULL

Printing Table:
Table[1][0]: (-9999,0) --> (0,1208) --> (1208,1208) --> (1208, NULL) --> NULL

Table[1][1]: (-9999,810) --> (810,1014) --> (1014,1217) --> (1217, NULL) --> NULL

Table[1][2]: (-9999,623) --> (623,524) --> (524,626) --> (626,1227) --> (1227,1129) --> (1129, NULL) --> NULL

Table[1][3]: (-9999,1531) --> (1531,1234) --> (1234,534) --> (534,537) --> (537,437) --> (437,1338) --> (1338,538) --> (538,539) --> (539,539) --> (539, NULL) --> NULL

Table[1][4]: (-9999,540) --> (540,1440) --> (1440,541) --> (541,541) --> (541,1442) --> (1442,942) --> (942,1145) --> (1145,745) --> (745,946) --> (946,546) --> (546,548) --> (548, NULL) --> NULL

Table[1][5]: (-9999,1353) --> (1353,555) --> (555,856) --> (856,1356) --> (1356,557) --> (557,758) --> (758,659) --> (659, NULL) --> NULL

Table[1][6]: (-9999,865) --> (865,666) --> (666, NULL) --> NULL

Table[1][7]: (-9999,571) --> (571,1071) --> (1071,1272) --> (1272,575) --> (575, NULL) --> NULL

Table[1][8]: (-9999,1287) --> (1287,588) --> (588, NULL) --> NULL

Table[1][9]: (-9999,893) --> (893, NULL) --> NULL

Printing Table:
Table[0][0]: (-9999,0) --> (0,1014) --> (1014,1071) --> (1071, NULL) --> NULL

Table[0][1]: (-9999,1129) --> (1129,1145) --> (1145, NULL) --> NULL

Table[0][2]: (-9999,1208) --> (1208,1208) --> (1208,1217) --> (1217,1227) --> (1227,1234) --> (1234,1272) --> (1272,1287) --> (1287, NULL) --> NULL

Table[0][3]: (-9999,1338) --> (1338,1353) --> (1353,1356) --> (1356, NULL) --> NULL

Table[0][4]: (-9999,437) --> (437,1440) --> (1440,1442) --> (1442, NULL) --> NULL

Table[0][5]: (-9999,524) --> (524,1531) --> (1531,534) --> (534,537) --> (537,538) --> (538,539) --> (539,539) --> (539,540) --> (540,541) --> (541,541) --> (541,546) --> (546,548) --> (548,555) --> (555,557) --> (557,571) --> (571,575) --> (575,588) --> (588, NULL) --> NULL

Table[0][6]: (-9999,623) --> (623,626) --> (626,659) --> (659,666) --> (666, NULL) --> NULL

Table[0][7]: (-9999,745) --> (745,758) --> (758, NULL) --> NULL

Table[0][8]: (-9999,810) --> (810,856) --> (856,865) --> (865,893) --> (893, NULL) --> NULL

Table[0][9]: (-9999,942) --> (942,946) --> (946, NULL) --> NULL

Printing Table:
Table[1][0]: (-9999,0) --> (0,437) --> (437,524) --> (524,534) --> (534,537) --> (537,538) --> (538,539) --> (539,539) --> (539,540) --> (540,541) --> (541,541) --> (541,546) --> (546,548) --> (548,555) --> (555,557) --> (557,571) --> (571,575) --> (575,588) --> (588,623) --> (623,626) --> (626,659) --> (659,666) --> (666,745) --> (745,758) --> (758,810) --> (810,856) --> (856,865) --> (865,893) --> (893,942) --> (942,946) --> (946, NULL) --> NULL

Table[1][1]: (-9999,1014) --> (1014,1071) --> (1071,1129) --> (1129,1145) --> (1145,1208) --> (1208,1208) --> (1208,1217) --> (1217,1227) --> (1227,1234) --> (1234,1272) --> (1272,1287) --> (1287,1338) --> (1338,1353) --> (1353,1356) --> (1356,1440) --> (1440,1442) --> (1442,1531) --> (1531, NULL) --> NULL