Class : CSC 313 (Fall 2024)
Name: Baldwin Cepeda
Project: Project 6
Language: (C++)
Due date: Nov 6, 2024

Top Level algorithm Step

Step 0: check argc count is correct

inFile, outFile1, logFile
open input file from argv[]

check all files can be opened.

Step 1: S
define as LLStack

Step 2: outFile1
“** calling buildStack ()”

buildStack (S->top, inFile, logFile)

outFile1
“** printing the stack after buildStack”

printStack(S->top, outFile1)

Step 3: Q
define as LLQ

Step 4: outFile1
“calling buildQueue ()”

buildQueue (S->top, Q->tail, logFile)

outFile1
“** printing the stack after buildQueue”

printQueue (Q->head, outFile1)

Step 5: close all files

Source Code:

```
/*
* Name: Baldwin Cepeda
* GitHub: https://github.com/BaldwinCepeda
* Website: http://baldwincepeda.me/
*
* Description:
* This program implements a stack and queue using linked lists in C++.
* The `LLStack` class provides standard stack operations (LIFO), while
* the `LLQueue` class implements a queue with FIFO behavior.
*
* Key Features:
* - `LLStack`: Supports `push`, `pop`, and `printStack`, with a dummy node as the top.
* - `LLQueue`: Includes `insertQ`, `deleteQ`, and `printQueue`, with indicators for
*   `head` and `tail` positions in the queue.
* - Demonstrates the transfer of elements from a stack to a queue, maintaining
*   respective order for each data structure.
*
* The program reads input from a file, logs operations to a log file, and
* outputs the stack and queue structures to an output file for verification.
*/
#include <iostream>
#include <fstream>
using namespace std;

class listNode
{
public:
  int data;
  listNode *next;

  listNode(int data) : data(data), next(nullptr) {}

  void printNode(ofstream &file)
  {
     file << "(" << data << ", ";
     if (next)
     {
        file << next->data;
```

```
        }
        else
        {
            file << "null"; // Indicate that there's no next node
        }
        file << ") ";
    }
};

class LLStack
{
public:
    listNode *top;

    LLStack()
    {
        top = new listNode(-1); // Dummy node
    }

    bool isEmpty()
    {
        return top->next == nullptr;
    }

    void push(int data)
    {
        listNode *newNode = new listNode(data);
        newNode->next = top->next;
        top->next = newNode;
    }

    listNode *pop()
    {
        if (isEmpty())
            return nullptr;

        listNode *tmp = top->next;
        top->next = tmp->next;
        tmp->next = nullptr;
        return tmp;
    }

    void buildStack(ifstream &inFile, ofstream &logFile)
    {
```

```
        logFile << "** Entering buildStack() **\n";

        int data;
        while (inFile >> data)
        {
            logFile << "Input data is " << data << endl;
            push(data);
            logFile << "** Printing Stack after push **\n";
            printStack(logFile);
        }

        logFile << "** Leaving buildStack() **\n";
    }

    void printStack(ofstream &file)
    {
        file << "Stack (top): ";
        listNode *current = top->next;
        while (current != nullptr)
        {
            current->printNode(file);
            current = current->next;
        }
        file << "NULL\n";
    }
};
class LLQueue
{
public:
    listNode *head; // Dummy head node
    listNode *tail; // Points to the last node of the queue

    // Constructor: Initializes the queue with a dummy node
    LLQueue()
    {
        head = new listNode(-999); // Dummy node with data -999
        tail = head;               // Initially, tail points to the head
    }

    // insertQ: Inserts a new node at the end of the queue
    void insertQ(listNode *newNode)
    {
        tail->next = newNode;    // Link current tail to new node
        tail = newNode;          // Update tail to the new node
```

```
    newNode->next = nullptr; // Ensure new node's next is null
}

// deleteQ: Removes and returns the first node after the dummy node (FIFO behavior)
listNode *deleteQ()
{
    if (head->next == nullptr) // Queue is empty
        return nullptr;

    listNode *tmp = head->next; // First real node in the queue
    head->next = tmp->next;     // Update head to skip over tmp
    if (head->next == nullptr)  // If queue becomes empty, reset tail to head
        tail = head;

    tmp->next = nullptr; // Detach tmp from the queue
    return tmp;
}

// buildQueue: Pops nodes from the stack and inserts them into the queue
void buildQueue(LLStack &stack, ofstream &logFile)
{
    logFile << "** Entering buildQueue() **\n";

    while (!stack.isEmpty())
    {
        listNode *newNode = stack.pop(); // Pop from stack
        logFile << "** Node popped from stack: " << newNode->data << " **\n";
        logFile << "** Printing stack after pop **\n";
        stack.printStack(logFile);

        insertQ(newNode); // Insert at the end of the queue
        logFile << "** Printing Queue after insert **\n";
        printQueue(logFile);
    }

    logFile << "** Leaving buildQueue() **\n";
}
void printQueue(ofstream &file)
{
    file << "Queue (head): ";
    listNode *current = head;
    while (current != nullptr)
    {
        current->printNode(file);
```

```
            if (current == tail)
            {
                file << "(tail) "; // Indicate the tail position
            }
            current = current->next;
        }
        file << "NULL\n";
    }
};

int main(int argc, char *argv[])
{
    if (argc != 4)
    {
        cerr << "Usage: " << argv[0] << " <inFile> <outFile1> <logFile>\n";
        return 1;
    }

    ifstream inFile(argv[1]);
    ofstream outFile1(argv[2]);
    ofstream logFile(argv[3]);

    if (!inFile.is_open() || !outFile1.is_open() || !logFile.is_open())
    {
        cerr << "Error opening files.\n";
        return 1;
    }

    LLStack stack;
    outFile1 << "** calling buildStack() **\n";
    stack.buildStack(inFile, logFile);

    outFile1 << "** Printing the stack after buildStack **\n";
    stack.printStack(outFile1);
    // outFile1 << "** TESTING **\n";
    // stack.pop();
    // outFile1 << "** After  pop ] **\n";
    // stack.printStack(outFile1);// outFile1 << "** TESTING **\n";
    // stack.pop();
    // outFile1 << "** After  pop2 ] **\n";
    // stack.printStack(outFile1);
    LLQueue queue;
    outFile1 << "** calling buildQueue() **\n";
    queue.buildQueue(stack, logFile);
```

```
  outFile1 << "** Printing the queue after buildQueue **\n";
  queue.printQueue(outFile1);

  // outFile1 << "** TESTING **\n";
  // stack.pop();
  // outFile1 << "** After  pop3 ] **\n";
  // stack.printStack(outFile1);

  inFile.close();
  outFile1.close();
  logFile.close();

  return 0;
}
```

Illustrions:

Program Output

Data1:

“799 44
702 838 91 32 333 888 999
“

Output:

** calling buildStack() **
** Printing the stack after buildStack **
Stack (top): (999, 888) (888, 333) (333, 32) (32, 91) (91, 838) (838, 702) (702, 44) (44, 799) (799, null) NULL
** calling buildQueue() **

** Printing the queue after buildQueue **
Queue (head): (-999, 999) (999, 888) (888, 333) (333, 32) (32, 91) (91, 838) (838, 702) (702, 44) (44, 799) (799, null) (tail) NULL

Data2:

“32 333 888 999
79 444
702 95 588 12 16 666
88 192 40 555
58 327 955 349 325
307 111 222
613 777
637 255
838 91
“

Output:
** calling buildStack() **
** Printing the stack after buildStack **
Stack (top): (91, 838) (838, 255) (255, 637) (637, 777) (777, 613) (613, 222) (222, 111) (111, 307) (307, 325) (325, 349) (349, 955) (955, 327) (327, 58) (58, 555) (555, 40) (40, 192) (192, 88) (88, 666) (666, 16) (16, 12) (12, 588) (588, 95) (95, 702) (702, 444) (444, 79) (79, 999) (999, 888) (888, 333) (333, 32) (32, null) NULL
** calling buildQueue() **
** Printing the queue after buildQueue **
Queue (head): (-999, 91) (91, 838) (838, 255) (255, 637) (637, 777) (777, 613) (613, 222) (222, 111) (111, 307) (307, 325) (325, 349) (349, 955) (955, 327) (327, 58) (58, 555) (555, 40) (40, 192) (192, 88) (88, 666) (666, 16) (16, 12) (12, 588) (588, 95) (95, 702) (702, 444) (444, 79) (79, 999) (999, 888) (888, 333) (333, 32) (32, null) (tail) NULL

Logs 1:
** Entering buildStack() **
Input data is 799
** Printing Stack after push **
Stack (top): (799, null) NULL
Input data is 44
** Printing Stack after push **
Stack (top): (44, 799) (799, null) NULL
Input data is 702
** Printing Stack after push **
Stack (top): (702, 44) (44, 799) (799, null) NULL
Input data is 838
** Printing Stack after push **
Stack (top): (838, 702) (702, 44) (44, 799) (799, null) NULL
Input data is 91
** Printing Stack after push **
Stack (top): (91, 838) (838, 702) (702, 44) (44, 799) (799, null) NULL
Input data is 32
** Printing Stack after push **
Stack (top): (32, 91) (91, 838) (838, 702) (702, 44) (44, 799) (799, null) NULL
Input data is 333
** Printing Stack after push **
Stack (top): (333, 32) (32, 91) (91, 838) (838, 702) (702, 44) (44, 799) (799, null) NULL
Input data is 888
** Printing Stack after push **
Stack (top): (888, 333) (333, 32) (32, 91) (91, 838) (838, 702) (702, 44) (44, 799) (799, null) NULL
Input data is 999
** Printing Stack after push **
Stack (top): (999, 888) (888, 333) (333, 32) (32, 91) (91, 838) (838, 702) (702, 44) (44, 799) (799, null) NULL
** Leaving buildStack() **
** Entering buildQueue() **
** Node popped from stack: 999 **

** Printing stack after pop **
Stack (top): (888, 333) (333, 32) (32, 91) (91, 838) (838, 702) (702, 44) (44, 799) (799, null) NULL
** Printing Queue after insert **
Queue (head): (-999, 999) (999, null) (tail) NULL
** Node popped from stack: 888 **
** Printing stack after pop **
Stack (top): (333, 32) (32, 91) (91, 838) (838, 702) (702, 44) (44, 799) (799, null) NULL
** Printing Queue after insert **
Queue (head): (-999, 999) (999, 888) (888, null) (tail) NULL
** Node popped from stack: 333 **
** Printing stack after pop **
Stack (top): (32, 91) (91, 838) (838, 702) (702, 44) (44, 799) (799, null) NULL
** Printing Queue after insert **
Queue (head): (-999, 999) (999, 888) (888, 333) (333, null) (tail) NULL
** Node popped from stack: 32 **
** Printing stack after pop **
Stack (top): (91, 838) (838, 702) (702, 44) (44, 799) (799, null) NULL
** Printing Queue after insert **
Queue (head): (-999, 999) (999, 888) (888, 333) (333, 32) (32, null) (tail) NULL
** Node popped from stack: 91 **
** Printing stack after pop **
Stack (top): (838, 702) (702, 44) (44, 799) (799, null) NULL
** Printing Queue after insert **
Queue (head): (-999, 999) (999, 888) (888, 333) (333, 32) (32, 91) (91, null) (tail) NULL
** Node popped from stack: 838 **
** Printing stack after pop **
Stack (top): (702, 44) (44, 799) (799, null) NULL
** Printing Queue after insert **
Queue (head): (-999, 999) (999, 888) (888, 333) (333, 32) (32, 91) (91, 838) (838, null) (tail) NULL
** Node popped from stack: 702 **
** Printing stack after pop **

Stack (top): (44, 799) (799, null) NULL
** Printing Queue after insert **
Queue (head): (-999, 999) (999, 888) (888, 333) (333, 32) (32, 91) (91, 838) (838, 702) (702, null) (tail) NULL
** Node popped from stack: 44 **
** Printing stack after pop **
Stack (top): (799, null) NULL
** Printing Queue after insert **
Queue (head): (-999, 999) (999, 888) (888, 333) (333, 32) (32, 91) (91, 838) (838, 702) (702, 44) (44, null) (tail) NULL
** Node popped from stack: 799 **
** Printing stack after pop **
Stack (top): NULL
** Printing Queue after insert **
Queue (head): (-999, 999) (999, 888) (888, 333) (333, 32) (32, 91) (91, 838) (838, 702) (702, 44) (44, 799) (799, null) (tail) NULL
** Leaving buildQueue() **

Logs 2:** Entering buildStack() **
Input data is 32
** Printing Stack after push **
Stack (top): (32, null) NULL
Input data is 333
** Printing Stack after push **
Stack (top): (333, 32) (32, null) NULL
Input data is 888
** Printing Stack after push **
Stack (top): (888, 333) (333, 32) (32, null) NULL
Input data is 999
** Printing Stack after push **
Stack (top): (999, 888) (888, 333) (333, 32) (32, null) NULL
Input data is 79
** Printing Stack after push **
Stack (top): (79, 999) (999, 888) (888, 333) (333, 32) (32, null) NULL
Input data is 444

** Printing Stack after push **
Stack (top): (444, 79) (79, 999) (999, 888) (888, 333) (333, 32) (32, null) NULL
Input data is 702
** Printing Stack after push **
Stack (top): (702, 444) (444, 79) (79, 999) (999, 888) (888, 333) (333, 32) (32, null) NULL
Input data is 95
** Printing Stack after push **