**Lab 7: Program 1 Documentation**

1. **Problem Statement**

We are reading an array of n elements. We create a template function that takes in the array of n elements, and a char value to determine integer sorting order (ascending or descending). We check the file for errors, then set two stacks to evaluate and order the data. One stack will be used as a sorted linked list stack, and the other will be used as a temporary array to pass in values. Once the sroted linked lists exists, you can print out all the values. The data in these stacks can be manipulated by the template function’s *pop, push, isFull,* & *isEmpty* methods.

1. **Requirements**
   1. **Assumptions**

* Input set through the source code
* Output will be handled by the command line
  1. **Specifications**
* Template Function mixMaxFunc()
  + It will contain two stacks: one sorted linked list stack & one temporary array
  + *push()*
  + *pop()*
  + *peek()*
  + *isFull()*
  + *isEmpty()*
* Print functions:
  + After it is sorted, it will print the linked list

1. **Decomposition Diagram**

* minMaxFunc (Template Class)
  + Input
    - There are checks in place to ensure all the data is valid
    - The user will input an array via a file
    - The user inputs values using the push() function
  + Process
    - Functions listed under specifications can be called in order to manipulate the data
    - Program has checks in place to process request and follows them accordingly
  + Output
    - Program will print out the values from the linked lists via usage of the print functions

|  |  |
| --- | --- |
| minMaxFunc() | |
| Responsibilities:  +push()  +pop()  +peek()  +isFull()  +isEmpty() | Collaborators: N/A |

|  |
| --- |
| minMaxFunc() |
| push(), pop(), peek(), isFull(), isEmpty() |
| * push()   + push a value into the stack * pop()   + pop a value from the stack * peek()   + check if the value is in the stack * isFull()   + check if the stack is full * isEmpty()   + check if the stack is empty |

1. **Test Strategy**

* Valid Data
* Invalid Data
* Empty File
* Nonexistent File

1. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid Data, File Exists | 1 | Pass valid file with data (ascending) |  |  |  |  |
| Valid Data, File Exists | 2 | Pass valid file with data (descending) |  |  |  |  |
| Valid Data, File Exists | 3 | Test pop function |  |  |  |  |
| Invalid Data, File Exists | 4 | Test non integer values |  |  |  |  |

1. **Initial Algorithm**

* template Function minMaxFunc()
* fstream *filename*
* Private:
  + Set Linked List
  + <typename>array[5] – array holding only 5 elements
* Public:
  + constructor
    - read the file, sorts it based on given order, and prints the data
  + void readFile(file)
    - Checks file values and set to type defined
    - Read first 5 values into array
    - Output clarification which states if array is full
  + void printData()
    - Prints out values of array in correct format
  + void push()
    - Confirms the data type of the element and checks if the array is full or not
    - It then finds the first null space and set the element to that location
    - Else it will throw an error
  + void pop()
    - if isEmpty is false, then it sets the last filled element in the array as null
    - else it throw an error message
  + peek()
    - if isEmpty is false, it will check the array to see if the value appears
  + isEmpty
    - if array[0] contains a value, it will return true
    - Else, it will be false
  + isFull
    - if array contains values in array[4], it will return true
    - Else, it will be false
  + void sortAscending()
    - Passes values of array into the linked list in ascending order after sorting (bubble sort)
  + void sortDescending()
    - Passes values of array into the linked list in descending order after sorting (bubble sort)

1. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid Data, File Exists | 1 | Pass valid file with data (ascending) | 5 3 2 4 1 | 1 2 3 4 5 |  |  |
| Valid Data, File Exists | 2 | Pass valid file with data (descending) | 5 3 2 4 1 | 5 4 3 2 1 |  |  |
| Valid Data, File Exists | 3 | Test pop function | 5 3 2 4 1 | 5 4 3 2 1 |  |  |
| Invalid Data, File Exists | 4 | Test non integer values | a b c e d | ERROR- “Please save file” |  |  |

1. **Code**

//Program: Lab 7 - Program 1 [CIS200, Steiner]

//Programmer Name: Srinivas Simhan

//Purpose: The purpose of the program was to learn how to use stacks to

//manipulate values into a linked list

//while reading and writing the values from a data file

//Date Due: 11/8/17

#include <iostream>

#include <fstream>

#include <string>

#include <typeinfo>

#include <cctype>

using namespace std;

//TEMPLATE NODE

template <typename T>

struct node

{

T val;

node \*previous;

node \*next;

};

//TEMPLATE LINKEDLIST

template <class listType>

class LinkedList

{

//Setting pointers for linked list

public:

node<listType> \*head = NULL;

node<listType> \*curr = NULL;

node<listType> \*tail = NULL;

//Create header, curr, and tail nodes, and attach them to each other

void newNode(listType val)

{

if (head == NULL)

{

head = new node<listType>;

head->previous = NULL;

curr = head;

curr->val = val;

curr->previous = head;

head->next = curr;

tail = curr;

curr = NULL;

}

else

{

curr = new node<listType>;

curr->previous = tail;

tail->next = curr;

curr->val = val;

tail = curr;

curr = NULL;

}

}

LinkedList()

{

//constructor

}

void reset()

{

head = NULL;

curr = NULL;

tail = NULL;

}

void readList()

{

if (head != NULL)

{

while (curr != NULL)

{

curr = head;

cout << curr->val;

curr = curr->next;

}

}

else

{

cout << "Make a head node, linked list doesn't exist";

}

}

listType checkTailVal()

{

return tail->val;

}

};

//TEMPLATE STACK

template <class stackType>

class Stack

{

private:

LinkedList<stackType> list;

stackType a[5];

int itr = 0;

public:

void readFile(ifstream &file)

{

//File doesn't open

if (file.fail())

{

throw "Error, file won't open.";

}

//File is empty

if (file.peek() == EOF)

{

throw "Error, file empty.";

}

while (file.peek() != EOF)

{

if (itr > 5)

{

cout << "Too many values in file, decrease size to 5.";

break;

}

else

{

file >> a[itr];

}

itr++;

}

}

//Checks file

Stack(ifstream &file)

{

readFile(file);

}

//Sort values in ascending order

void sortAscending()

{

bool isSorted = false;

stackType curr;

// while isSorted == false

// go through whole list

// if finding an element out of order, fix it

// if after going through the whole list, nothing was moved, then we are done

//Bubble Sort

while (isSorted == false)

{

isSorted = true;

for (int i = 0; i < itr; i++)

{

if (i != 0)

{

if (a[i] < a[i - 1])

{

isSorted = false;

curr = a[i];

a[i] = a[i - 1];

a[i - 1] = curr;

}

}

}

}

}

//Sort values in descending order

void sortDescending()

{

bool isSorted = false;

stackType curr;

//Bubble Sort

while (isSorted == false)

{

isSorted = true;

for (int i = 0; i < itr; i++)

{

if (i != 0)

{

if (a[i] > a[i - 1])

{

isSorted = false;

curr = a[i];

a[i] = a[i - 1];

a[i - 1] = curr;

}

}

}

}

}

//Prints Highest Value

void minMaxFunc()

{

cout << "Last value in this set: " << list.checkTailVal;

}

void putInList()

{

for (int i = 0; i < itr; i++)

{

list.newNode(a[i]);

}

}

void printArray()

{

for (int i = 0; i < itr; i++)

{

cout << a[i];

}

}

bool isFull()

{

if (itr == 4)

{

return true;

}

else

{

return false;

}

}

bool isEmpty()

{

if (itr == 0)

{

return true;

}

else

{

return false;

}

}

void push(stackType value)

{

if (isFull() == false)

{

a[itr] = value;

itr++;

node<stackType>\* newNode= new node<stackType>;

newNode->val = value;

node<stackType>\* oldHead = list.head;

node<stackType>\* newHead = newNode;

newHead->next = oldHead;

oldHead->previous = newHead;

list.head = newHead;

}

else

{

cout << "Error! Full List!" << endl;

}

sortAscending();

}

void reset()

{

list.reset();

}

void readList()

{

list.readList();

}

void pop()

{

if (isEmpty() == false)

{

a[itr] = 0;

itr--;

list.tail->previous->next = NULL;

}

else

{

cout << "Error! No values in list" << endl;

}

}

};

//MAIN FUNCTION

int main()

{

ifstream readFile("Data.dat");

Stack<int> listStack(readFile);

listStack.sortAscending();

listStack.printArray();

cout << endl;

listStack.sortDescending();

listStack.printArray();

cout << endl;

listStack.putInList();

listStack.printArray();

cout << endl << endl << endl << endl;

listStack.pop();

listStack.printArray();

cout << endl;

listStack.push(4);

listStack.printArray();

cout << endl;

listStack.readList();

listStack.printArray();

cout << endl;

system("pause");

return 0;

}

1. **Updated Algorithm**

* template<typename T>
  + struct node
    - T val
    - Node \*previous
    - Node \*next
* Template<class listType>
  + Public:
  + Set node head, temp, tail
  + newNode(listType val)
    - Reset head, temp, and tail
  + LinkedList()
    - Constructor for object
  + Void reset()
    - Reset head, temp, and tail to nullptr
  + Void readList()
    - Read values of list
    - Else linked list doesn’t exist
  + listType checkTailVal()
    - returns tail value
* Template<class stackType>
  + Private:
    - Set Linked List
    - <typename>array[5] – array holding only 5 elements
  + Public:
    - void readFile(ifstream &file)
      * Checks file values and set to type defined
      * Read first 5 values into array
      * Output clarification which states if array is full
    - void sortAscending()
      * Passes values of array into the linked list in ascending order after sorting (bubble sort)
    - void sortDescending()
      * Passes values of array into the linked list in descending order after sorting (bubble sort)
    - void minMaxFunc()
      * print last tail value
    - void putInList()
      * iterate through list and put into array
    - void printArray()
      * Prints out values of array in correct format
    - bool isFull()
      * if iterator is 4, then it returns true
      * else false
    - bool isEmpty()
      * if iterator is 0, then it returns true
      * else false
    - void push()
      * Confirms the data type of the element and checks if the array is full or not
      * It then resets the value as the new head of the list
      * Else it will throw an error
    - void reset()
      * resets list
    - void readList()
      * reads list
    - void pop()
      * if isEmpty is false, then it sets the last filled element in the array as null
      * else it throw an error message

1. **Test Plan Version 3**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid Data, File Exists | 1 | Pass valid file with data (ascending) | 5 3 2 4 1 | 1 2 3 4 5 | \*see screenshot\* | Pass |
| Valid Data, File Exists | 2 | Pass valid file with data (descending) | 5 3 2 4 1 | 5 4 3 2 1 | \*see screenshot\* | Pass |
| Valid Data, File Exists | 3 | Test pop function | 5 3 2 4 1 | 5 4 3 2 1 | \*see screenshot\* | Pass |
| Invalid Data, File Exists | 4 | Test non integer values | a b d g e | ERROR | \*see screenshot\* | Fail |

1. **Screenshots**

Test Cases 1-3



Test Case 4



1. **Error Log**
   1. Need to double check the source code because needs some fine tuning in the code. Need to test all the functions more.
2. **Status**
   1. The final status of my program was that it worked but still needs some finer tuning.