Table of Contents

[Part I 2](#_Toc525848180)

[Data Structure Documentation 2](#_Toc525848181)

[UML Document 3](#_Toc525848182)

[Detailed Design 4](#_Toc525848183)

[Client 4](#_Toc525848184)

[**Class:** LinkedList 5](#_Toc525848185)

[Test Plan 6](#_Toc525848186)

[Client Code 6](#_Toc525848187)

[Class: LinkedList 7](#_Toc525848188)

[Class Specification Files 8](#_Toc525848189)

[LinkedList.h 8](#_Toc525848190)

[Class Implementation Files 9](#_Toc525848191)

[LinkedList.cpp 9](#_Toc525848192)

[Source Code 12](#_Toc525848193)

[Output 13](#_Toc525848194)

[List Of Scenarios 14](#_Toc525848195)

[Scenario 1 14](#_Toc525848196)

[Part II 15](#_Toc525848197)

[Part III 16](#_Toc525848198)

[a) 16](#_Toc525848199)

[b) 16](#_Toc525848200)

[c) 16](#_Toc525848201)

# Part I

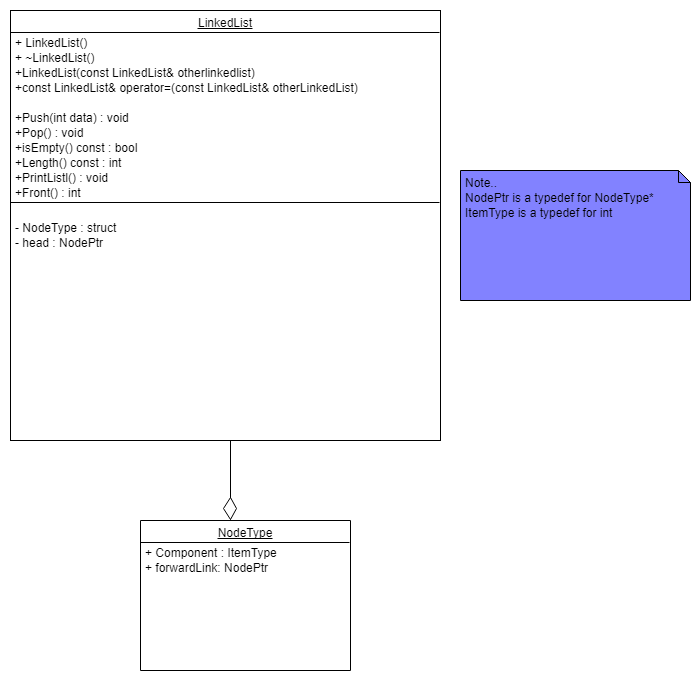
## Data Structure Documentation

(let n be the number of men and women in each list)

Data Structures used in this program:

* LinkedList(Stack)
  + freewoman
    - This holds the list of women that still need to be engaged
* 3 Two-Dimensional Arrays size [n][n]
  + manPref
    - This holds the preference list of all men for all women
  + womanPref
    - This holds the preference list of all women for all men
  + Ranking
    - This gives the ranking of each woman in the preference list of each man
* 2 One Dimensional Arrays size [n]
  + Next
    - Gives the index of the next man to propose to in the preference list of each woman
  + Current
    - Gives the current (and ending) engagement of each man

## UML Document



## Detailed Design

### Client

**Data Structures Used:**

*Enums*

*enum menName { VICTOR = 0, WYATT = 1, XAVIER = 2, YANCEY = 3, ZEUS = 4 };*

*enum womanName { AMY = 0, BERTHA = 1, CLARE = 2, DIANE = 3, ERIKA = 4 };*

*Global Variable*

*Int n ( the number of men & women in the lists)*

**Function: main()**

*int manPref[n][n]*

*int womanPref[n][n]*

*int matches[n]*

*try {*

*TestLList();*

*ReadPreferancesFromFile(manPref, womanPref)*

*StableMatch(manPref, womanPref, matches)*

*PrintMatches(matches)*

*}*

*Catch exception{*

*Print error to screen and kill program*

*}*

**Function:** void StandardInitializePreferances(int manPref[n][n], int womanPref[n][n])

Narrative: This is a function that creates a standard test case for the program. It initializes both manPref and womanPref

Pre-condition: None

Post-condition: ManPref and womanPref will have been initialized

*Hardcode each value of the arrays.*

**Function:**  *void ReadPreferancesFromFile(int manPref[n][n], int womanPref[n][n]);*

Narrative: Read in preferances from file.

Pre-condition: The input has the correct number of lines and is formatted correctly

Post-condition: manPref and womanPref will be initialized with the values in the file

ifstream inFile;

string filename;

int temp;

Ask user for filename

If file exists

For (n1 number of women)

For (n2 preferences of men for the women)

Read in number

Store in temp

womanPref[n1][n2] = temp

For (n1 number of men)

For (n2 preferences of women for the men)

Read in number

Store in temp

manPref[n1][n2] = temp

Else

the file does not exist, throw the error that the file was not found

**Function:** *void TestLList();*

Narrative:

Pre-condition:

Post-condition:

*local variable declarations*

*algorithm*

**Function:** *void StableMatch(int manPref[n][n], int womanPref[n][n], int matches[n]);*

Narrative:

Pre-condition:

Post-condition:

*local variable declarations*

*algorithm*

**Function:** *void PrintMatches(int matches[n]);*

Narrative:

Pre-condition:

Post-condition:

*local variable declarations*

*algorithm*

### **Class:** LinkedList

*private data*

**Method:** *method header with parameters*

Narrative:

Pre-condition:

Post-condition:

*algorithm*

## Test Plan

### Client Code

**Function Prototype: main()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
|  | Exception thrown in try block | Wrong file | Wrong filename | Error 404 program shutdown |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Function Prototype:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

### Class: LinkedList

**Method Prototype: Push()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
|  | Item Pushed onto stack | Empty list | 1 | 1 pushed onto stack |  |
|  |  | Non-Empty list | 2 | 2 pushed onto stack |  |

**Method Prototype: Pop()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
|  | Item Poped off top of list | Empty list |  | Noting happens |  |
|  |  | Non-Empty list | 3 | Item on top is popped off |  |

**Method Prototype:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Method Prototype: isEmpty()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
|  | List Empty | True |  | True |  |
|  |  | False |  | False |  |

**Method Prototype: Length()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
|  | Length of the list | Empty list |  | 0 |  |
|  |  | Non-Empty list | 3 | 3 |  |

**Method Prototype: Front()**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
|  | Returns item on top of list | Empty list |  | -1 |  |
|  |  | Non-Empty List | 3 | 3 |  |

## Class Specification Files

### LinkedList.h

#ifndef LinkedList\_H

#define LinkedList\_H

class LinkedList {

private:

typedef int ItemType;

struct NodeType;

typedef NodeType\* NodePtr;

public:

LinkedList();

~LinkedList();

LinkedList(const LinkedList& otherlinkedlist);

const LinkedList& operator=(const LinkedList& otherLinkedList);

void Push(int data);

void Pop();

bool Search(int data);

bool isEmpty() const;

int Length() const;

void PrintList();

int Front();

private:

struct NodeType

{

ItemType component;

NodePtr forwardLink;

};

NodePtr head;

int length;

};

#endif

## Class Implementation Files

### LinkedList.cpp

#include "LinkedList.h"

#include <iostream>

LinkedList::LinkedList() {

head = nullptr;

length = 0;

}

LinkedList::~LinkedList() {

NodePtr temp;

NodePtr curr = head;

while (curr != nullptr) {

temp = curr;

curr = curr->forwardLink;

delete temp;

}

head = nullptr;

return;

}

LinkedList::LinkedList(const LinkedList & otherLinkedList) {

NodePtr from;

NodePtr to;

length = otherLinkedList.length;

if (otherLinkedList.head == nullptr) {

head = nullptr;

}

else {

from = otherLinkedList.head;

head = new NodeType;

head->component = from->component;

to = head;

from = from->forwardLink;

while (from != nullptr) {

to->forwardLink = new NodeType;

to = to->forwardLink;

to->component = from->component;

from = from->forwardLink;

}

to->forwardLink = nullptr;

}

}

const LinkedList & LinkedList::operator=(const LinkedList & otherLinkedList)

{

NodePtr from;

NodePtr to;

length = otherLinkedList.length;

if (otherLinkedList.head == nullptr) {

head = nullptr;

}

else {

from = otherLinkedList.head;

head = new NodeType;

head->component = from->component;

to = head;

from = from->forwardLink;

while (from != nullptr) {

to->forwardLink = new NodeType;

to = to->forwardLink;

to->component = from->component;

from = from->forwardLink;

}

to->forwardLink = nullptr;

}

return \*this;

}

void LinkedList::Push(int data) {

NodePtr tmp = new NodeType;

tmp->component = data;

if (head == nullptr)

{

tmp->forwardLink = nullptr;

head = tmp;

length++;

}

else

{

tmp->forwardLink = head;

head = tmp;

length++;

}

}

void LinkedList::Pop() {

NodePtr tmp;

if (head != nullptr)

{

tmp = head;

head = head->forwardLink;

length--;

delete tmp;

}

}

bool LinkedList::Search(int data) {

NodePtr current = head;

while (current != nullptr) {

current = current->forwardLink;

if (data == current->component)

return true;

}

return false;

}

bool LinkedList::isEmpty() const {

if (length == 0)

return true;

return false;

}

int LinkedList::Length() const {

return length;

}

void LinkedList::PrintList() {

NodePtr head = this->head;

if (length == 0)

std::cout << "This list is empty";

while (head) {

std::cout << head->component << " ";

head = head->forwardLink;

}

return;

}

int LinkedList::Front() {

return head->component;

}

## Source Code

## Output

## List Of Scenarios

### Scenario 1



### Scenario 2



### Scenario 3



# Part II

What general observations can you make from the algorithm you implemented about the matches that are made based on the women’s preference lists? How about the matches made based on the men’s preference lists?

# Part III

## a)

## b)

## c)