Table of Contents

[Part I 2](#_Toc525847983)

[Data Structure Documentation 2](#_Toc525847984)

[UML Document 3](#_Toc525847985)

[Detailed Design 4](#_Toc525847986)

[Client 4](#_Toc525847987)

[**Class:** LinkedList 5](#_Toc525847988)

[Test Plan 6](#_Toc525847989)

[Client Code 6](#_Toc525847990)

[Class: LinkedList 7](#_Toc525847991)

[Class Specification Files 8](#_Toc525847992)

[LinkedList.h 8](#_Toc525847993)

[Class Implementation Files 9](#_Toc525847994)

[LinkedList.cpp 9](#_Toc525847995)

[Source Code 12](#_Toc525847996)

[Output 13](#_Toc525847997)

[List Of Scenarios 14](#_Toc525847998)

[Scenario 1 14](#_Toc525847999)

[Part II 15](#_Toc525848000)

# Part I

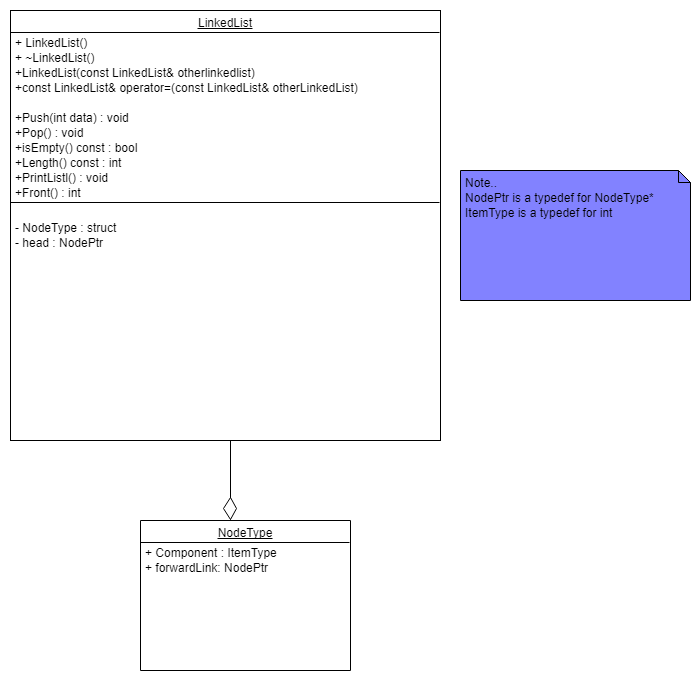
## Data Structure Documentation

(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*

*typedefs*

*structs*

**Function: main()**

*local variable declarations*

*algorithm*

**Function:** *function header with parameters here*

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** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Function Prototype:**

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

### Class: LinkedList

**Method Prototype:**

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

## 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();

bool HeadNull();

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;

}

bool LinkedList::HeadNull() {

if (head == nullptr) {

return true;

}

return false;

}

## Source Code

## Output

## List Of Scenarios

### Scenario 1

# 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)