Table of Contents

[Part I 2](#_Toc525895919)

[Data Structure Documentation 2](#_Toc525895920)

[UML Document 3](#_Toc525895921)

[Detailed Design 4](#_Toc525895922)

[Client 4](#_Toc525895923)

[**Class:** LinkedList 9](#_Toc525895924)

[Test Plan 15](#_Toc525895925)

[Client Code 15](#_Toc525895926)

[Class: LinkedList 16](#_Toc525895927)

[Class Specification Files 18](#_Toc525895928)

[LinkedList.h 18](#_Toc525895929)

[Class Implementation Files 19](#_Toc525895930)

[LinkedList.cpp 19](#_Toc525895931)

[Source Code 22](#_Toc525895932)

[Output 29](#_Toc525895933)

[List Of Scenarios 34](#_Toc525895934)

[Scenario 1 34](#_Toc525895935)

[Scenario 2 35](#_Toc525895936)

[Scenario 3 36](#_Toc525895937)

[Part II 37](#_Toc525895938)

[Part III 38](#_Toc525895939)

[a) Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample 38](#_Toc525895940)

[b) Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample. 38](#_Toc525895941)

[c) Television Network question 38](#_Toc525895942)

# Part I

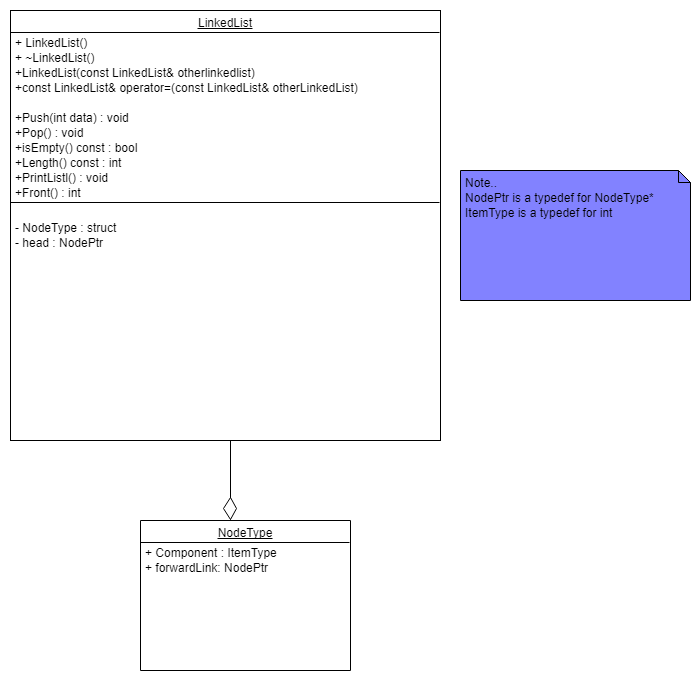
## Data Structure Documentation

(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: This function is solely to test the LinkedList class and make sure all of the functions being used are working correctly

Pre-condition: none

Post-condition: LinkedList will be thoroughly tested.

*Linked List Test*

*Test isEmpty on empty list*

*Test pop on empty list*

*Test Front on empty list*

*Add items to list*

*Test is empty on list*

*Test front on list*

*Test length on list*

*Test pop on list*

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

Narrative: Men and women are represented by integers 0...n-1

ManPref is the preference list of all men for all women.

ManPref[m][i] = w is at ith position in the preference list of m

WomanPref is the preference list of all women for all men.

WomanPref[w][i] = m is at ith position in the preference list of w

Ranking gives the ranking of each woman in the preference list of each man

Ranking[w][m] = manPref[w][i] = m

Current gives the current engagement of each man

Current[w] = m is currently engaged to w

Next gives the index of next man to propose to in the preference list of each woman

Next[w] = w has proposed to all m s.t. womanPref[w][j] = m for j = 1...i-1 but not womanPref[w][i]

Pre-condition:

Post-condition:

**Int** *Ranking [n][n], Next[n], Current[n], m, w*

*Linkedlist freew*

*Initialize*

*For(I = 0; I < n; i++)*

*Next[i] = 0*

*Current[i] = -1*

*freewoman.Push(i)*

*for(j = 0;j<n;j++)*

*Ranking[i][manPref[i][j]] = j + 1 🡨* ***(n x n array ranking where***

***Ranking [M,W] contains the rank of woman W in sorted order of M's preferences).***

*End for*

End for

While(there are woman in the linkedList)

W = freew.front

M = womanPref[w][next++] 🡨 this will start at 0 and each time the same woman needs the next

one in their list it will increment

if(Current[m] == -1)

Current[m] = w

freewoman.pop

end if

else if

Ranking[m][Current[m]] > Ranking[m][w]

Freewoman.pop

Freewoman.push(current[m])

Current[m] = w

End else

End while

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

Narrative: This function prints out the matches in (M,W) order

Pre-condition: The StableMatch function has been ran and matches is containing the women that are married to the men in order

Post-condition: The matches of the lists are printed to the screen in (M,W) order

*Output “ (Man , Woman)” so that the user knows the ordering of the pairs*

*For(I = 0; I < n; i++)*

*Output (I , match[i])*

### **Class:** LinkedList

*Struct Nodetype{*

*ItemType component*

*NodePtr forwardLink*

*}*

*NodePtr head*

*Int length*

**Method:** *Push(int data)*

Narrative: Place an item at the front of the list

Pre-condition: None

Post-condition: If there are items in the list, this will push them back and add the item onto the front of the list

*NodePtr tmp = new NodeType*

*tmp->component = data*

*if (head == nullptr)*

*{*

*tmp->forwardLink = nullptr*

*head = tmp*

*length++*

*}*

*else*

*{*

*tmp->forwardLink = head*

*head = tmp*

*length++*

*}*

**Method:** *void Pop()*

Narrative: This will delete the item at the front of the list

Pre-condition: None

Post-condition: If there are items in the list, the front item will be deleted. Otherwise nothing happens

*NodePtr tmp*

*if (head != nullptr)*

*{*

*tmp = head*

*head = head->forwardLink*

*length--*

*delete tmp*

*}*

**Method:** *bool isEmpty() const*

Narrative: Will return true if list is empty, otherwise false

Pre-condition: none

Post-condition: will return a true or false

*if (length == 0)*

*return true*

*return false*

**Method:** *int Length() const*

Narrative: This returns the current length of the list

Pre-condition: None

Post-condition: The length of the list is returned

*return length*

**Method:** *int Front()*

Narrative: If there are items in the list, this will return the component of the head pointer, otherwise it will return a -1

Pre-condition: none

Post-condition: an integer will be returned

*if(head!= nullptr)*

*return head->component*

*return -1*

**Method:** *LinkedList()*

Narrative: Constructor

Pre-condition: None

Post-condition: Variables Initialized

*head = nullptr*

*length = 0*

**Method:** *~LinkedList()*

Narrative: Destructor

Pre-condition: none

Post-condition: Link List Destroyed

*NodePtr temp*

*NodePtr curr = head*

*while (curr != nullptr) {*

*temp = curr*

*curr = curr->forwardLink*

*delete temp*

*}*

*head = nullptr*

*return*

**Method:** *LinkedList(const LinkedList & otherLinkedList)*

Narrative: Copy Constructor

Pre-condition: None

Post-condition: Will have created a deep copy of the list

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

}

**Method:** *operator=(const LinkedList & otherLinkedList)*Narrative:

Pre-condition:

Post-condition:

*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*

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **Testing for** | **Test Case** | **Input/Test value** | **Expected Outcome** | **Observed Result** |
|  | if(Current[m] == -1) |  |  | Add current woman proposing to the list as a match |  |
|  | Ranking[m][Current[m]] > Ranking[m][w] |  |  | Switch the current woman proposing and the one that had already proposed |  |
|  | Ranking[m][Current[m]] < Ranking[m][w] |  |  | Go back through list and propose to next man in woman’s list |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

### 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: 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

#include <iostream>

#include <fstream>

#include <string>

#include <vld.h>

#include "LinkedList.h"

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* The Stable Matching Program \*/

/\* Developed by \*/

/\* Michael Crispen \*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Global Enumerations

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

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

//Constant number in both lists

const int n = 7; //Number of men & Women

//Function Declerations

void StandardInitializePreferances(int manPref[n][n], int womanPref[n][n]);

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

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

void PrintMatches(int matches[n]);

void TestLList();

int main() {

int manPref[n][n];

int womanPref[n][n];

int matches[n];

try {

//StandardInitializePreferances(manPref, womanPref);

TestLList();

ReadPreferancesFromFile(manPref, womanPref);

StableMatch(manPref, womanPref, matches);

PrintMatches(matches);

}

catch (string n) {

cout << endl << endl;

cout << "Program Shutting down due to error: " << n << endl << endl;

system("pause");

return 1;

}

cout << endl << endl;

system("pause");

return 0;

}

void StandardInitializePreferances(int manPref[n][n], int womanPref[n][n]) {

//This function is a hardcoded preferance initializer so to test with a standard input.

//We use names in this as it is easier to read than numbering each man and woman.

//Women are numbers 0 to N-1, Men are numbered 0 to N-1

//Amy W0

womanPref[0][0] = ZEUS;

womanPref[0][1] = VICTOR;

womanPref[0][2] = WYATT;

womanPref[0][3] = YANCEY;

womanPref[0][4] = XAVIER;

//Bertha W1

womanPref[1][0] = XAVIER;

womanPref[1][1] = WYATT;

womanPref[1][2] = YANCEY;

womanPref[1][3] = VICTOR;

womanPref[1][4] = ZEUS;

//Clare W2

womanPref[2][0] = WYATT;

womanPref[2][1] = XAVIER;

womanPref[2][2] = YANCEY;

womanPref[2][3] = ZEUS;

womanPref[2][4] = VICTOR;

//Diane W3

womanPref[3][0] = VICTOR;

womanPref[3][1] = ZEUS;

womanPref[3][2] = YANCEY;

womanPref[3][3] = XAVIER;

womanPref[3][4] = WYATT;

//Erika W4

womanPref[4][0] = WYATT;

womanPref[4][1] = YANCEY;

womanPref[4][2] = ZEUS;

womanPref[4][3] = XAVIER;

womanPref[4][4] = VICTOR;

//Victor M0

manPref[0][0] = BERTHA;

manPref[0][1] = AMY;

manPref[0][2] = DIANE;

manPref[0][3] = ERIKA;

manPref[0][4] = CLARE;

//Wyatt M1

manPref[1][0] = BERTHA;

manPref[1][1] = AMY;

manPref[1][2] = DIANE;

manPref[1][3] = ERIKA;

manPref[1][4] = CLARE;

//Xavier M2

manPref[2][0] = BERTHA;

manPref[2][1] = ERIKA;

manPref[2][2] = CLARE;

manPref[2][3] = DIANE;

manPref[2][4] = AMY;

//Yancey M3

manPref[3][0] = AMY;

manPref[3][1] = DIANE;

manPref[3][2] = CLARE;

manPref[3][3] = BERTHA;

manPref[3][4] = ERIKA;

//Zeus M4

manPref[4][0] = BERTHA;

manPref[4][1] = DIANE;

manPref[4][2] = AMY;

manPref[4][3] = ERIKA;

manPref[4][4] = CLARE;

}

void ReadPreferancesFromFile(int manPref[n][n], int womanPref[n][n]) {

//Read in preferances from file.

//open file

//Read in n lines of preferances (This will be the womans preferances)

//Read in n lines of preferances (This will be the mans preferances)

//This is always assuming that the input is the correct number of lines

//If the file does not exist it will throw the error that the file was not found

ifstream inFile;

string filename;

int temp;

cout << "Please Enter The Filename You Wish To Use: ";

cin >> filename;

inFile.open(filename);

if (inFile.is\_open()) {

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

//cout << endl;

for (int j = 0; j < n; j++) {

inFile >> temp;

womanPref[i][j] = temp;

//cout << temp << " " ;

}

}

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

//cout << endl;

for (int j = 0; j < n; j++) {

inFile >> temp;

manPref[i][j] = temp;

//cout << temp << " ";

}

}

}

else {

string error = "Error 404 File Not Found";

throw(error);

}

return;

}

void StableMatch(int manPref[n][n], int womanPref[n][n], int matches[n]) {

/\*

Men and women are represented by integers 0...n-1

ManPref is the preference list of all men for all women.

ManPref[m][i] = w is at ith position in the preference list of m

WomanPref is the preference list of all women for all men.

WomanPref[w][i] = m is at ith position in the preference list of w

Ranking gives the ranking of each woman in the preference list of each man

Ranking[w][m] = manPref[w][i] = m

Current gives the current engagement of each man

Current[w] = m is currently engaged to w

Next gives the index of next man to propose to in the preference list of each woman

Next[w] = w has proposed to all m s.t. womanPref[w][j] = m for j = 1...i-1 but not womanPref[w][i]

Steps

1. Identify Free Woman

2. For a woman W identify the highest-ranked man to whom hasnt proposed

3. For a man M, we need to decide if M is currently engaged, if he is, identify current partner

4. For a man M and two Women W and W', we need to be able to decide (In constant time), whether W or W' is preferred by M

Start by creating n x n array ranking where Ranking [M,W] contains the rank of woman W in sorted order of M's preferances.

\*/

LinkedList freeWoman;

int Ranking[n][n], Next[n], Current[n];

int m , w; //Man and woman working with

//Init step

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

Next[i] = 0;

Current[i] = -1;

freeWoman.Push(i);

for (int j = 0; j < n; j++)

Ranking[i][manPref[i][j]] = j +1;

}

//BLAB

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

cout << endl << "Woman " << i << "'s Preferance List: ";

for (int y = 0; y < n; y++) {

cout << womanPref[i][y] << " ";

}

}

cout << endl;

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

cout << endl << "Man " << i << "'s Preferance List: ";

for (int y = 0; y < n; y++) {

cout << manPref[i][y] << " ";

}

}

cout << endl << endl;

cout << "Ranking List: ";

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

cout << endl;

for (int j = 0; j < n; j++)

cout << Ranking[i][j] << " ";

}

cout << endl << endl << "Next List: " << endl;

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

cout << Next[i] << " ";

}

cout << endl << endl << "Current List: " << endl;

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

cout << Current[i] << " ";

}

cout << endl << endl << "freeWoman LinkedList: " << endl;

freeWoman.PrintList();

cout << endl << endl;

//End Blab

//End Initialization

//Algorithm

cout << "Proposals: " << endl;

while (!freeWoman.isEmpty()) {

w = freeWoman.Front();

m = womanPref[w][Next[w]++];

cout << "Next: " << Next[w] << endl;

cout << w << " Proposes to " << m << endl;

if (Current[m] == -1) {

cout << w << " Married unmarried man " << m << endl;

Current[m] = w;

freeWoman.Pop();

}

else {

//cout << "Checking if " << w << " " << m << " is better than " << w << " " << Current[m] << endl << endl; ;

cout << "w: " << w << endl << "m: " << m << endl << endl;

cout << "Current : ";

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

cout << Current[i] << " ";

}

cout << endl;

cout << "Current[m] :" << Current[m];

cout << endl << endl;

cout << "else if (Ranking[Current[m]][m] < Ranking[m][w])" << endl;

cout << "if ( " << Ranking[m][Current[m]] << " < " << Ranking[w][m] << " ) " << endl;

if (Ranking[m][Current[m]] > Ranking[m][w]) {

cout << "This woman is better for the man.. Replacing " << endl;

freeWoman.Pop();

freeWoman.Push(Current[m]);

Current[m] = w;

}

}

cout << endl << endl << endl;

}

cout << endl << endl;

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

matches[i] = Current[i];

return;

}

void PrintMatches(int matches[n]) {

//Matches are printed to the screen in the form (w,m)

cout << "( Man , Woman )" << endl;

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

//cout << i << "Has Married " << Current[m] << endl;

cout << "( " << i << ", " << matches[i] << " )" << endl;

}

cout << endl;

return;

}

void TestLList() {

cout << endl << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << " Starting Linked List Tests " << endl;

cout << endl;

cout << "Creating List" << endl;

LinkedList test;

if (test.isEmpty())

cout << "The list is empty";

else

{

cout << "The list has items";

}

cout << endl;

cout << "Head data of empty list: " << test.Front() << endl;

cout << "Pushing 1, 2, and 3" << endl;

test.Push(1);

test.Push(2);

test.Push(3);

cout << "Length of list : " << test.Length() << endl;

if (test.isEmpty())

cout << "The list is empty";

else

{

cout << "The list has items";

}

cout << endl;

cout << "Item at the front of list : " << test.Front() << endl;

cout << "Popping items 1 2 and 3 off the list" << endl;

test.Pop();

test.Pop();

test.Pop();

test.Pop();

if (test.isEmpty())

cout << "The list is empty";

else

{

cout << "The list has items";

}

cout << endl;

cout << "Ending Linked List Tests " << endl;

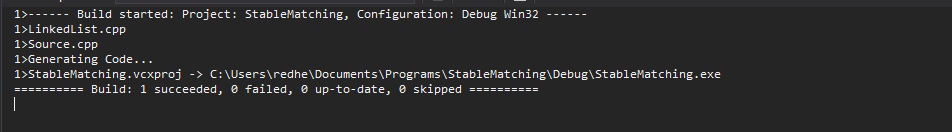
cout << endl << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

return;

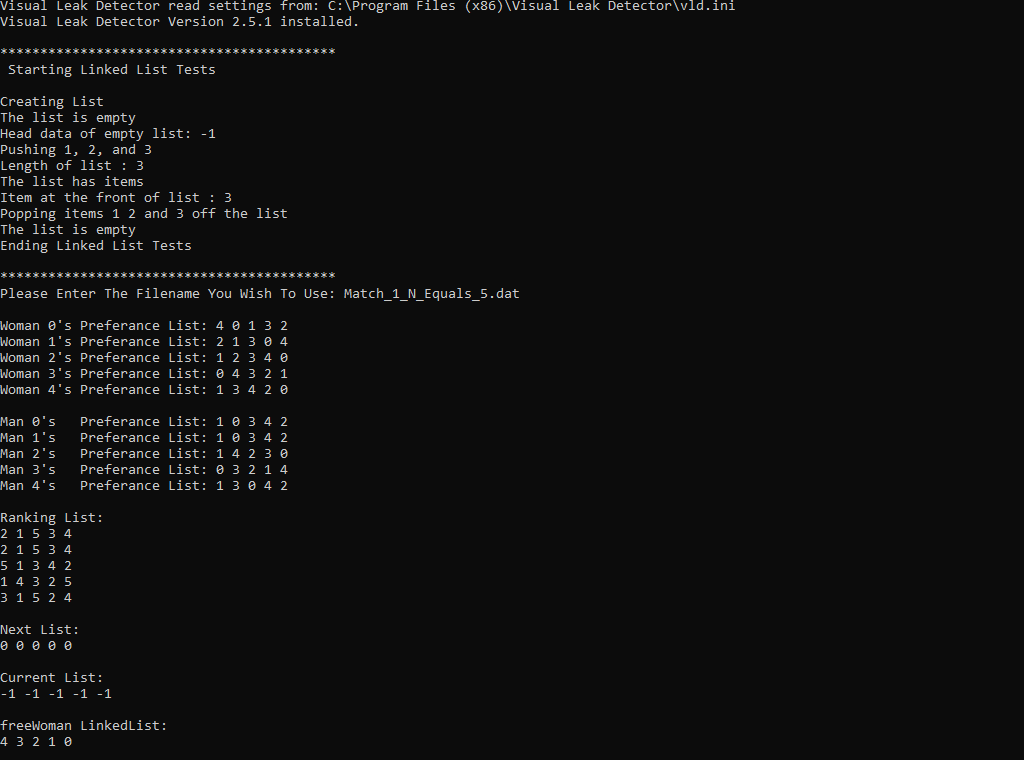
}

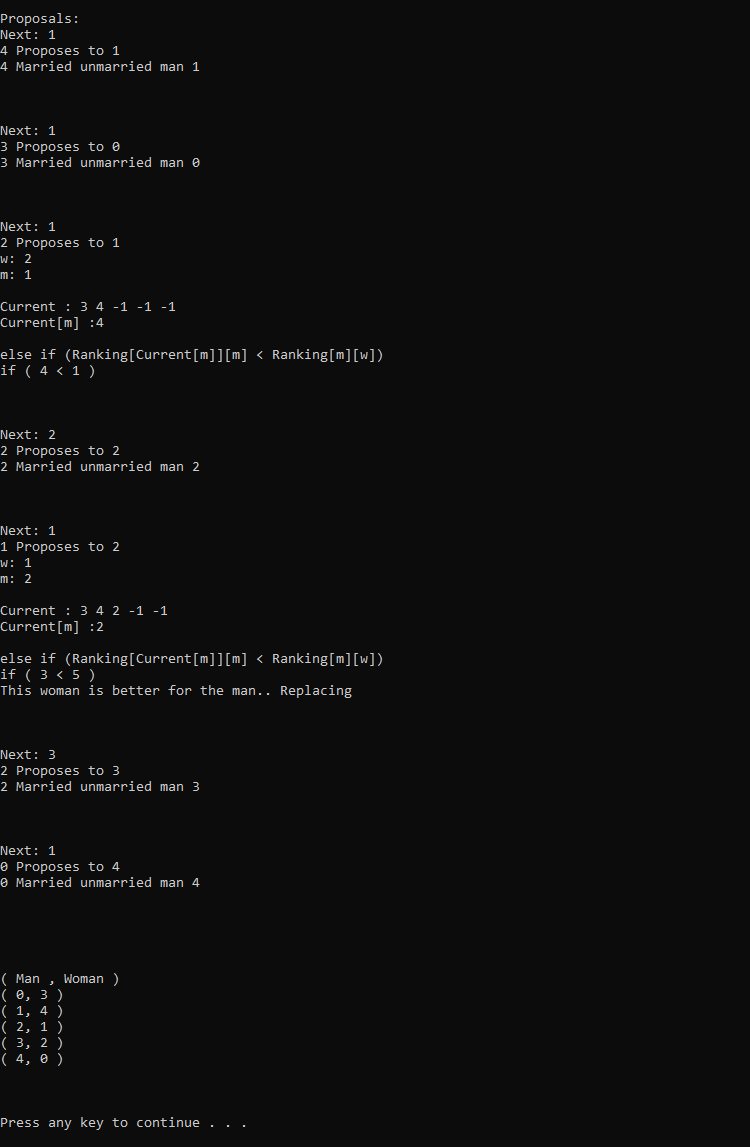
## Output

Build

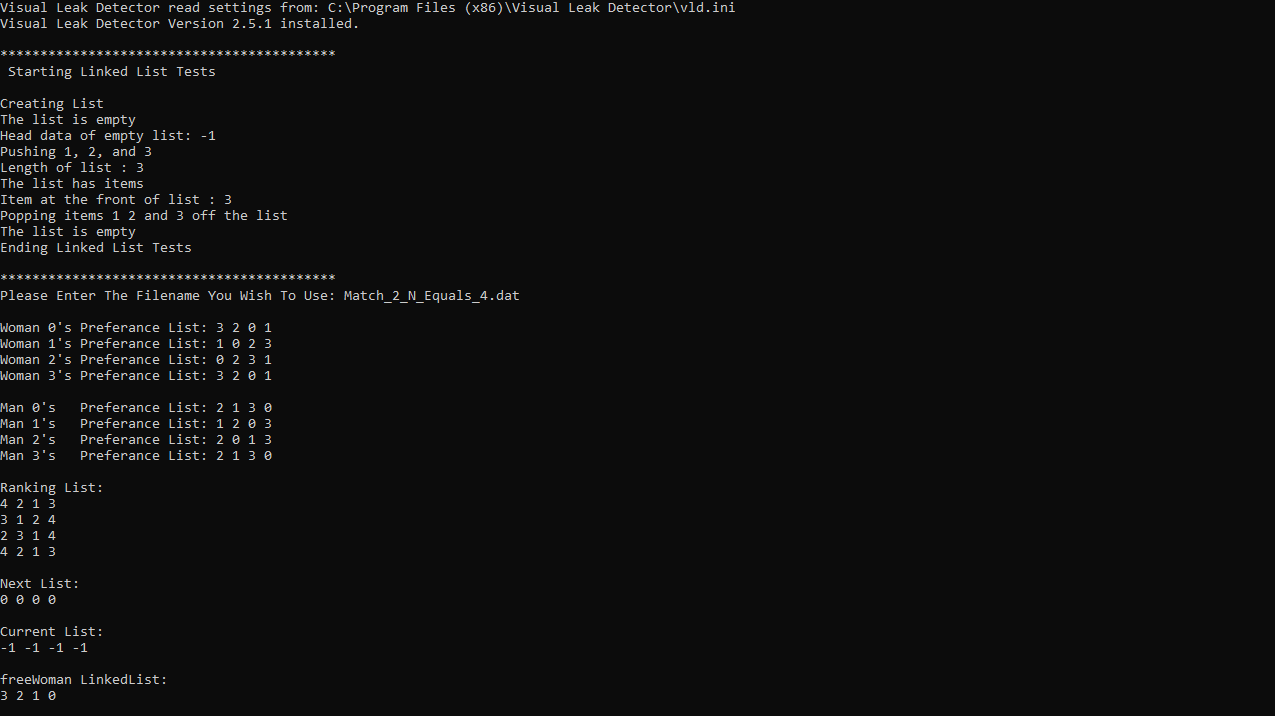


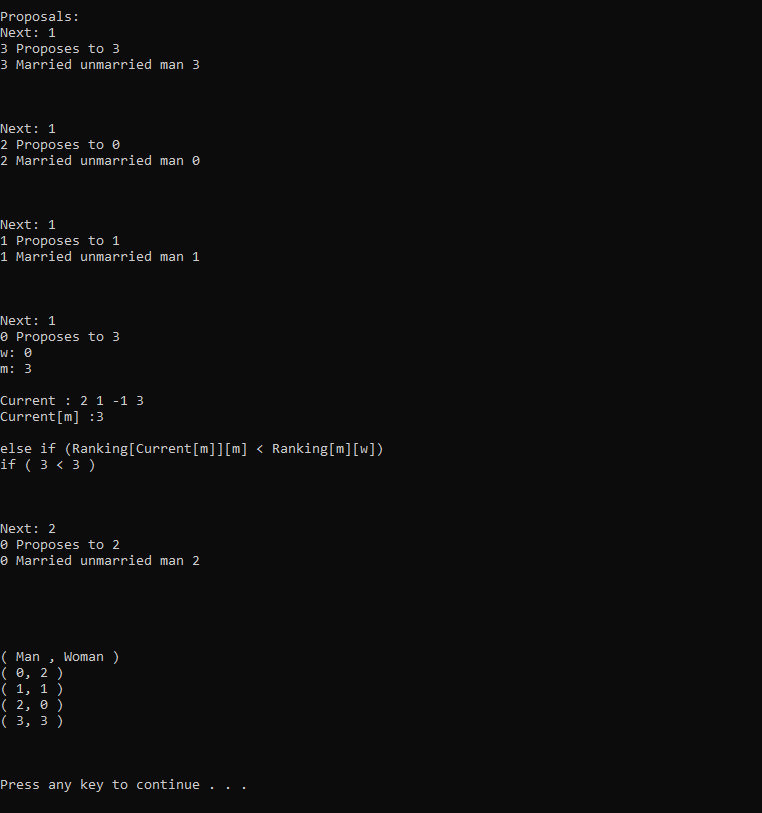
Scenario 1



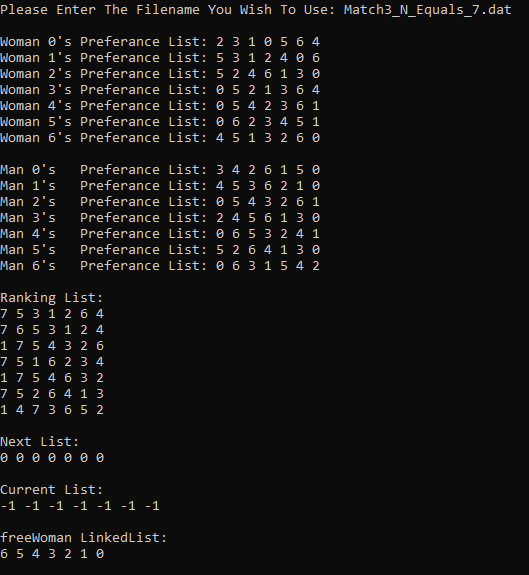


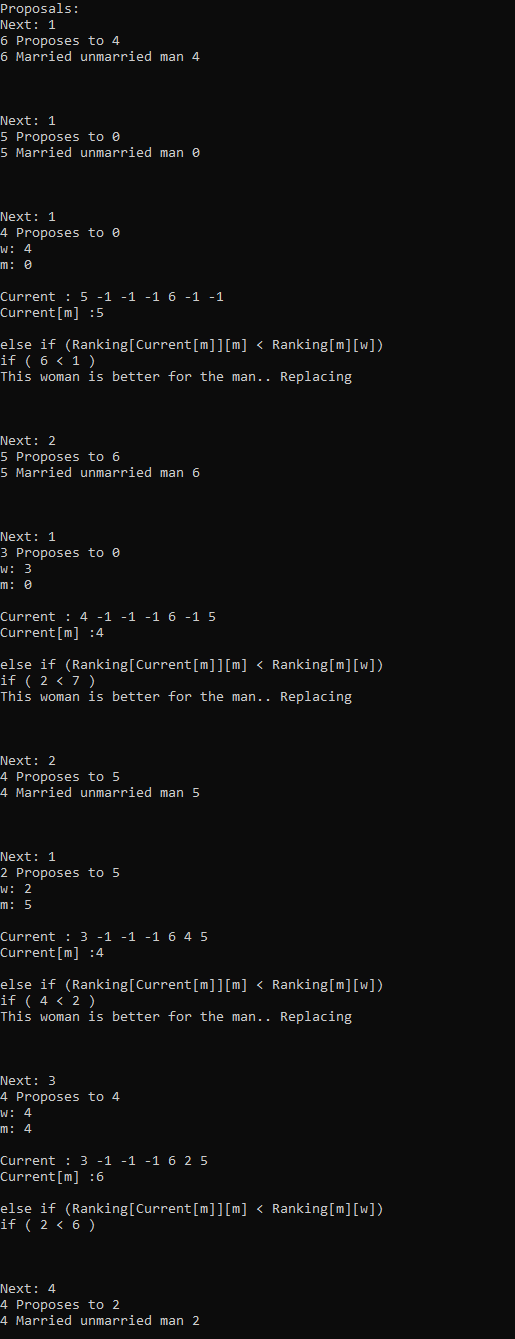
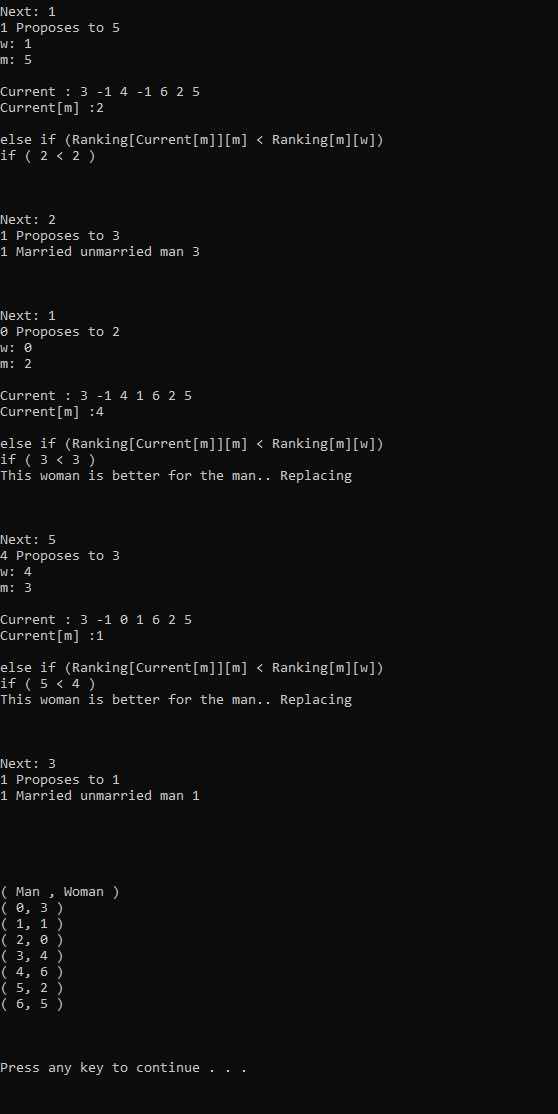
Scenario 2





Scenario 3





## 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?

One of the most general observations that I can make is that when selecting a partner, whomever is doing the selecting will always trade partners down while whomever is being selected will always trade up. This is to say that based on the woman’s preference list the woman would always trade down so the longer the algorithm runs the worse off the women are. It would be the exact same for the men if they were doing the choosing.

# Part III

## Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample

True or false? In every instance of the Stable Matching Problem, there is a stable matching containing a pair (m, w) such that m is ranked first on the preference list of w and w is ranked first on the preference list of m.

This is false

Say you have the Preferances as follows

M = { W , W’}

M’ = {W , W’}

W = {M , M’}

W’ = {M , M’}

In this example you would have the only stable matching pair of {(M, W)(M’,W’)}. This contradicts that m and w are ranked first on each other’s preference list.

## Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

True or false? Consider an instance of the Stable Matching Problem in which there exists a man m and a woman w such that m is ranked first on the preference list of w and w is ranked first on the preference list of m. Then in every stable matching S for this instance, the pair (m, w) belongs to S.

True, this would happen because when the man proposes to the woman she will always say yes. The man will always ask this woman first because she is first on his priority list. The woman will also always deny any other man that proposes to her after being proposed to by man m.

## Television Network question

There is not always a stable pair of schedules. Stability in this case is designed as if neither network can unilaterally change its own schedule and win more time slots.

Say we have the following schedule for n = 2 slots

Tv A Tv B

Slot 1 5 10

Slot 2 15 20

Slot 1 & 2 B wins

A will want to change the schedule so that they can at least win one slot. This will change the schedule to

Tv A Tv B

Slot 1 15 10

Slot 2 5 20

Slot 1 A Wins

Slot 2 B Wins

But now tv B will want to change its schedule to win back the slots so,

Tv A Tv B

Slot 1 15 20

Slot 2 5 10

This is a continuous cycle that will have no stable match.