PSET 1

/*

Problem 1:

We need to create a modified form of PrefW to be an array indexed on machinist's numbers (given input of a machinist number into a specific welder's preference list we want to get out its preference number). Essentially, we want to sort PrefW on the output and make the information held the index of the output. For example:

```
PrefW[0][i] = [4 3 2 5 1 0]
               0 1 2 3 4 5 (order of importance)

PrefWRev[0][i] = [5 4 2 1 0 3]
                  0 1 2 3 4 5 (machinist indices)
```

so PrefWRev[0][3] gives preference of Welder_0 to Machinist_3 which is 1.

A modified form of PrefM will also be created to make writing out the algorithm simpler (it's not necessary though)

A modfied form of Match will also need to be created based on Welder's indices. So we will have a MatchM (original match) and MatchW (modified to be indexed on Welder's indices).

*/

```
//To initialize PrefWRev and PrefMRev
int *PrefWRev[n];
for (int i = 0; i < n; ++i)
{
        PrefWRev[i] = (int *) malloc(n*sizeof(int));
        PrefMRev[i] = (int *) malloc(n*sizeof(int));
}

//To create PrefWRev
for (int i = 0; i < n; ++i)
{
        for (int j = 0; i < n; ++j)
        {
                PrefWRev[i][PrefW[i][j]] = j;
                PrefMRev[i][PrefM[i][j]] = j;
        }
}a

int MatchW[n];
int *MatchM = Match; //essentially renaming Match
//To Create MatchW
for (int i = 0; i < n; ++i)
{
        MatchW[Match[i]] = i;
}

//To Test for Unstable Pairs
/*
Go through every pair.
For a given (M, W) pair, check all the other Welders that M prefers more than W (these are the
only welders that could cause an instability)
Loop through these Welders (a size of at most size n). If one of these welders prefer M over its
current partner then there is an instability. This is because M prefers these welders over its
partner W.
*/
for (int i = 0; i < n; ++i)
{
        int W = MatchM[i] //Mi ismatched with a W
        int importanceW = PrefMRev[i][W] //how much Mi prefers W (there are an
importanceW number of Welders that M prefers to its current partner W). 0 means most
important.

        for (int j = 0; j < importanceW ; ++j) //all W's to work with
        {
                int wCurr = PrefM[i][j] //current W to test (M prefers wCurr more than its current
partner)
                int mCurr = MatchW[wCurr] //wCurr is currently paired with mCurr

                if (PrefWRev[wCurr][i] > PrefWRev[wCurr][mCurr]) {
                //if wCurr prefers Mi to who the mCurr (the machinist it is currently paired with)
then there is an instability because as shown above Mi prefers wCurr more than its current pair.
                        return unstable; //or false
                }
        }
}

return stable; //or true
```

/*

PrefWRev and PrefMRev Run Time take n^2 time to create. There are two encased for loops with only constant time calls.

MatchW takes n time to create. One for loop with a size of n.

To Test for Unstable Pairs Run Time:
importanceW can be at most size n. (M, W) is the current pair. In the worst case, W is M's least favorite Welder and importanceW takes a value of n (M prefers n-1 other welders).
There are two encased for loops with max size n and in each loop there are only constant time lookups (array accesses).
This leads to n^2 time.

*/

/*
Question 2:
2.1 Counterexample:

This is a stable perfect matching S proven in class:
(A, X)
(B, Z)
(C, Y)
A prefers X over every other welder and X prefers A over every other machinist so that pair can't be unstable nor could either A or X cause an instability.
B only prefers X over Z but X prefers A over B (still stable). C prefers Z and X over Y but Z prefers A and B over C and X prefers A and B over C (still stable). Therefore, this is a stable perfect matching.

Machinist	Preferences	Welder	Preferences
A	X Y Z	X	A B C
B	X Z Y	Y	A C B
C	Z X Y	Z	A B C

if the welder A quits and machinist X is free:
(B, Z) B prefers free machinist X to its current partner Z. (B, X) is an instability.
(C, Y)

We have shown an example where S - {(m,w)} is unstable so it can't always be stable.

2.2a Example:
We will use the same original stable perfect matching S. Welder w is hired.
The original matching S can still be stable if w is the least preferred welder by all the machinists.

Machinist	Preferences	Welder	Preferences
A	X Y Z W	X	A B C
B	X Z Y W	Y	A C B
C	Z X Y W	Z	A B C
		W	A B C

Stable Matching Still Holds:
(A, X)
(B, Z)
(C, Y)

This holds for the traditional stable definition because it was already proven that this matching was stable.
This holds for the free welder condition because every machinist would prefer their current welder pair over W (W is least preferred by all).

2.2b Counter Example:

Cascading effect where w is most preferred by one machinist which displaces machinist and welder pairs further down.

Machinist	Preferences				Welder	Preferences		
A	W	X	Y	Z	X	A	B	C
B	X	Z	Y	W	Y	A	C	B
C	Z	X	Y	W	Z	A	B	C
					W	A	B	C

No Longer Stable:
(A, X)
(B, Z)
(C, Y)

A prefers free W to its current partner X.
(A, W) becomes a pair.
X is now free.
(B, Z) is now unstable because B prefers X to its partner Z.
(B, X) becomes a pair.
Z is now free.
(C, Y) is now unstable because C prefers Z to its partner Y.
(C, Z) becomes a pair.

Updated Stable Pairs:
(A, W)
(B, X)
(C, Z)

Proof of stability:
A prefers nothing over W and W prefers nothing over A (stable pair that can’t cause any instabilities). B prefers nothing over X. C prefers nothing over Z. A, B, and C all have their top choice so there can’t be an instability.

*/

/*
Question 3:
Counter Example:
A,B on Team 1
X,Y on Team 2

skill order: A > X > B > Y

Team 1: 1 win, Team 2: 1 win
A Y
B X

Team 1: 2 wins, Team 2: 0 wins
B Y
A X

Knowing any sequence from one team, the other team has an incentive to change their current sequence to win one more game.
This leads to no stable sequence existing.

/*

Question 4:
4.1 counterexample

Example Situation:

Machinists	Preferences
a	W X
b	W X
e	W X
f	W X

Welders	Preferences
W	e f b a
X	e f b a

a and b are in Union 1
e and f are in Union 2

Possible Pairs that meet the requirements of the trade unions:

Pair 1: unstable because w prefers free machinist f
(w,a)
(x,e)

Pair 2: unstable because w prefers free machinist e
(w,b)
(x,f)

Pair 3: unstable because w prefers free machinist e
(w,a)
(x,f)

Pair 4: unstable because w prefers free machinist f
(w,b)
(x,e)

Pair 5: unstable because w prefers free machinist f
(x,a)
(w,e)

Pair 6: unstable because w prefers free machinist e
(x,b)
(w,f)

Pair 7: unstable because w prefers free machinist e
(x,a)
(w,f)

Pair 8: unstable because w prefers free machinist f
(x,b)

(w,e)

These 8 pairs are all possible pairs that meet the trade unions requirement that half the welders are paired with one union of machinists and the other half are paired with the other union of machinists.
None of them are stable pairs because the Welders (W and X) dominantly prefer Machinists in one union (Union 2). This leads to a free machinist (either e or f) who is always prefered by one welder.

4.2 Proof + Algorithm
Algorithm:
Number the machinists. M_1 to M_n are machinists in Union 1. M_{n+1} to M_{2n} are machinists in Union 2.
Number the welders and split them into two even groups. W_1 to $W_{(n/2)}$ (Group 1) and $W_{(n/2)+1}$ to W_n

Run Gayle-Shapley twice with Welders inviting Machinists:
One run of Gayle-Shapley with Group 1 Welders inviting machinists in Union 1.
Another run of Gayle-Shapley with Group 2 Welders inviting machinists in Union 2.

Proof of algorithm always finding a stable matching:
This is a proof by contradiction.

Assume (m,w) is an instability so there is two ways this can happen:
1) m and m' are in the same group. There exists (m, w') and (m', w) where m prefers w to w' and w prefers m to m'.
This can't happen in Gayle Shapley. Gayle Shapley does not allow this standard definition of an instability to happen.
This is a contradiction.

2) m and m' are in the same group. m is free, w is paired with m'
If m is free, m never got invited.
If w is paired with m', w sent an invitation to m'.
But if w prefers m over m', w would have invited m first.
This is a contradiction.

There since (m, w) was arbitrary. There are no instabilities.

Run Time:
Numbering the machinists and splitting the welders into two even groups (can take O(n) time or constant time depending on how it's implemented)
Running Gayle-Shapley twice takes $O(n^2)$ time
This leads to the algorithm taking $O(n^2)$ time overall

*/