Name: _____

Directions: Do NOT turn in this sheet of paper (unless you lack a laptop or have a laptop failure during the Exam). You will submit electronic files to **handin**.

INSTRUCTIONS FOR SUBMISSION:

- Submit to the CSIF **handin**, under my account, directory **158quiz7** using the alphabetically earliest UCD e-mail address among your group members.
- Submit a file named **Who.txt** that lists the UCD e-mail addresses (without the @ucdavis.edu) of your group members, one per line.
- Submit ONLY the files **transpose.c** (Problem 1, no **main()**) and **Transgraph.cpp** (Problem 2, including **main()**).
- Get your files into **handin** by 2 minutes after the exam. After that, you may be penalized.
- You will receive a 10-point bonus if you comply fully with the specs.

1. (50) Here you will write an OpenMP program to do matrix transpose, with the following specs:

- The matrix is square, nxn, with **int** entries.
- The transposition is done in-place. Do not create any auxiliary arrays.
- The signature of your function must be

```
void transp(int *m, int n)
```

- Here is my test code:

```
#include <stdio.h>
#include <stdlib.h>

void printmat(int *m, int n)
{  int i,j,k=0;
   for (i = 0; i < n; i++)  {
      for (j = 0; j < n; j++)
         printf("%d ",m[k++]);
      printf("\n");
   }
}

int main(int argc, char **argv)
{  int i;
   int n = atoi(argv[1]), n2 = n*n;
   int *testm = malloc(n2*sizeof(int));
   for (i = 0; i < n2; i++)
      testm[i] = rand() % 16;
   printmat(testm,n);
   transp(testm,n)
   printmat(testm,n);
}
```

My test script will run the commands

```
gcc transposemain.c transpose.c -fopenmp
setenv OMP_NUM_THREADS 3
a.out 5
```

and will expect the output

```
7 6 9 3 1
15 10 12 9 13
10 11 2 11 3
6 12 2 4 8
11 8 7 13 6

7 15 10 6 11
6 10 11 12 8
9 12 2 2 7
3 9 11 4 13
1 13 3 8 6
```

2. (50) Consider the adjacency graph transformation method we've seen before. You will write Thrust code to transform an adjacency matrix for a directed graph to an equivalent but different, two-column form. In each instance in which element (i,j) is 1, indicating a link from i to j, the transformed matrix has a row consisting of (i,j). Here are the details:

- Your code will fill the blanks in the following:

```
LARGE BLANK

int main(int argc, char **argv)
{  int x[12] = {
       0,1,1,0,
       1,0,0,1,
       1,1,0,0};
   int nr=3,nc=4;
   LARGE BLANK
   for (i = 0; i < n1s; i++)
      printf("%d %d\n",newmat[2*i],newmat[2*i+1]);
}
```

Here **newmat** is a Thrust array you'll create.

- My script will run the commands

```
g++ -g -O2 Transgraph.cpp -fopenmp \
   -I/usr/local/cuda/include \
   -DTHRUST_DEVICE_BACKEND=THRUST_DEVICE_BACKEND_OMP
setenv OMP_NUM_THREADS 3
a.out
```

and will expect the output

```
0 1
0 2
1 0
1 3
2 0
2 1
```

- Your code must work for general **nr** and **nc**.
- Remember, you will submit an entire program, including the parts of **main()** given above.

Recommended approach: Keep in mind that both your input and output arrays are one-dimensional, even though we are storing matrices. First, use Thrust to determine the indices of the 1s in the input array. Then use Thrust to fill in the contents of the output array.

Solutions:

1.

```
#include <omp.h>

// translate from 2-D to 1-D indices
int onedim(int n,int i,int j) {  return n * i + j;  }

void transp(int *m, int n)
{
   #pragma omp parallel
   {  int i,j,tmp;
      // walk through all the above-diagonal elements, swapping them
      // with their below-diagonal counterparts
      #pragma omp for
      for (i = 0; i < n; i++) {
         for (j = i+1; j < n; j++) {
            tmp = m[onedim(n,i,j)];
            m[onedim(n,i,j)] = m[onedim(n,j,i)];
            m[onedim(n,j,i)] = tmp;
         }
      }
   }
}
```

2.

```
// transgraph problem, using Thrust

#include <stdio.h>

#include <thrust/host_vector.h>
#include <thrust/transform.h>
#include <thrust/sequence.h>
#include <thrust/remove.h>

// forms one row of the output matrix
struct makerow {
   const thrust::host_vector<int>::iterator outmat;
   const int nc;  // number of columns
   makerow(thrust::host_vector<int>::iterator _outmat,int _nc) :
      outmat(_outmat), nc(_nc) {}
   __host__ __device__
   // the j-th 1 is in position i of the orig matrix
   bool operator()(const int i, const int j)
   {  outmat[2*j] = i / nc;
      outmat[2*j+1] = i % nc;
   }
};


int main(int argc, char **argv)
{  int x[12] = {
      0,1,1,0,
      1,0,0,1,
      1,1,0,0};
   int nr=3,nc=4,nrc = nr*nc,i;
   thrust::host_vector<int> hx(x,x+nrc);
   thrust::host_vector<int> seq(nrc);
   thrust::sequence(seq.begin(),seq.end(),0);
   thrust::host_vector<int> ones(x,x+nrc);
   // get 1-D indices of the 1s
   thrust::host_vector<int>::iterator newend =
      thrust::copy_if(seq.begin(),seq.end(),hx.begin(),ones.begin(),
         thrust::identity<int>());
   int n1s = newend - ones.begin();
   thrust::host_vector<int> newmat(2*n1s);
   thrust::host_vector<int> out(n1s);
   thrust::host_vector<int> seq2(n1s);
   thrust::sequence(seq2.begin(),seq2.end(),0);
   thrust::transform(ones.begin(),newend,seq2.begin(),out.begin(),
      makerow(newmat.begin(),nc));
```

```
46      for (i = 0; i < n1s; i++)
47          printf("%d %d\n",newmat[2*i],newmat[2*i+1]);
48  }
```