```
/*
 ============================================================================
===
 Name        : matrix
 Author      : Nathan Christian
 Version     : TBD
 Copyright   : Not copyrighted
 Description : Matrix program to demonstrate recursive functions
 ============================================================================
===
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>


struct Point{
    int value, location;
}; //create a struct point to store the value and location of the highest
number in the matrix

struct Point recursive_call(void *arr, int m, int count, int *row, int
*col, int choice, struct Point p); //recursively reduce to four matrix and
find highest value

struct Point max(int *array, int count, int total, int biggest, int
location2); //recursively find highest value

int main(void) {
      clock_t begin = clock();

      int input, m, j, i, row, col;
      struct Point final; //store the final values here

      row=col=0; //store the row and column location of the highest value

      srand(time(NULL));
      setbuf(stdout, NULL);
      printf("This program generates a matrix of size 2^n. User selects
value of n.\n");
      printf("Matrix is divided into four quadrants:\n[1][2]\n[3][4]\n");
      printf("inside each quadrant there are columns and rows beginning
with 1 up to 2^n\n\n");
      printf("At each step of recursion, the quadrant with the highest
value is printed and identified.\n");
      printf("Please enter a value for n: ");
      clock_t end = clock();
      double time_spent1 = (double)(end - begin) / CLOCKS_PER_SEC;
      scanf("%d", &input); //explanation and read of matrix size
      begin = clock();
      m=pow(2,input);
      int matrix[m][m]; //create a matrix of appropriate size
```

```
	printf("Would you like to print the matrices on screen? (warning,
this may cause instability for n values greater than 5\n");
	printf("enter '1' for yes or '2' for no");
	end = clock();
	double time_spent2 = (double)(end - begin) / CLOCKS_PER_SEC;
	scanf("%d", &input);
	begin = clock();
	if (input==1)
			printf("\nThe original matrix before recursion is: \n");
	for(j=0;j<m;j++)
	{
			for(i=0;i<m;i++)
			{
					matrix[j][i]=0+rand()%((int)pow(m,3))+1;
//assign random values to entire matrix
					if (input==1)
						printf("%d ", matrix[j][i]);
			}
			printf("\n");
	} //randomly generate matrix values and print original matrix.

	final=recursive_call(matrix, m, 0, &row, &col, input, final); //call
recursive function to find max and location

	printf("The final product returned to main is %d and its location is
column %d row %d", final.value, col, row); //print final statement
	end = clock();
	double time_spent3 = (double)(end - begin) / CLOCKS_PER_SEC;
	printf("\nTotal execution time is approximately: %lf ms.",
time_spent1+time_spent2+time_spent3);
	return 0;
}

struct Point max(int *array, int count, int total, int biggest, int
location2)
{
	int temp[total*total], j;

	struct Point point;
	point.value=biggest;
	point.location=location2; //declare variables and set equal to
function arguments for recursion

	for(j=0;j<total*total;j++){
			temp[j]=*array++;
	}
	for(j=0;j<total*total;j++)
			array--; //temporarily store values in an array for
recursion. rewind array for recursion.

	if (temp[count]>point.value)
	{
			point.value=temp[count];
```

```
                point.location=count;
      } //if the current run of recursion is greater than value stored.
biggest value =current value


      if (count==total*total-1)
                return point; //return the point once all values have
been compared

      else
                max(array, (count+1), total, point.value,
point.location); //if not all values have been compared, recursivel call
the next value using count+1
}

struct Point recursive_call(void *arr, int m, int count, int *row, int
*col, int choice, struct Point p)
{
      int n=m/2, quad1[n][n], quad2[n][n], quad3[n][n], quad4[n][n], j, i,
*ptr, quadrant;

      struct Point p1,p2,p3,p4, biggest;
      biggest=p;  //declare variables and set equal to function argument
for recursion

      int (*arr2)[m][m] = (int (*)[m][m]) arr; //typef used because matrix
of undefined size cannot be passed as an argument


   for (i=0;i < n; i++)
            for (j=0; j <n; j++)
                     quad1[i][j]=(*arr2)[i][j];  //create the first
block (quadrant 1)
   for (i=0;i < n; i++)
            for (j=n; j < m; j++)
                     quad2[i][j-n]=(*arr2)[i][j];  //create the second
block (quadrant 2)
   for (i=n;i < m; i++)
            for (j=0; j < n; j++)
                     quad3[i-n][j]=(*arr2)[i][j];  //create the third
block (quadrant 3)
   for (i=n;i < m; i++)
            for (j=n; j < m; j++)
                     quad4[i-n][j-n]=(*arr2)[i][j];  //create the fourth
block (quadrant 4)

   ptr=&quad1[0][0];
   p1 = max(ptr, 0, n, 0, 0);

   ptr=&quad2[0][0];
   p2= max(ptr, 0, n, 0, 0);

   ptr=&quad3[0][0];
   p3= max(ptr, 0, n, 0, 0);
```

```
   ptr=&quad4[0][0];
   p4= max(ptr, 0, n, 0, 0); //temporarily store values in an array and
send to function max


     if(p1.value>p2.value && p1.value>p3.value && p1.value>p4.value){
            quadrant=1; //if largest value is in the 1st quadrant, set
quadrant equal to 1
            if (count==0){
            if (p1.location<4)
                        *col=p1.location+1;
             else
                         *col=(p1.location%n)+1;
             *row=(p1.location/n)+1;
             biggest=p1;
            } //if this is the first read, the largest value and its
location are stored so they are not lost in recursion

     }
     else if (p2.value > p3.value && p2.value >p4.value){
                  quadrant=2; //if largest value is in the 2nd quadrant
set quadrant equal to 3
            if (count==0){
            if(p2.location<4)
                         *col=(p2.location+n+1);
                    else
                         *col=((p2.location%n)+1+n);
                    row=(p2.location/n)+1;
                      biggest=p2;
            }//if this is the first read, the largest value and its
location are stored so they are not lost in recursion

     }
     else if (p3.value > p4.value){
            quadrant=3; //if largest value is in the 3rd quadrant set
quadrant equal to 3
            if (count==0){
            if(p3.location<4)
                        *col=p3.location+1;
            else
                         *col=(p3.location%n)+1;
            *row=((p3.location/n)+1+n);
                        biggest=p3;
            }
     }//if this is the first read, the largest value and its location are
stored so they are not lost in recursion
     else
     {
            quadrant=4; //if largest value is in the 4th quadrant set
quadrant equal to 4
            if (count==0){
            if(p4.location<4)
                        *col=(p4.location+1+n);
```

```
            else
                        *col=((p4.location%n)+n+1);
            *row=((p4.location/n)+n+1);
            quadrant=4;
                        biggest=p4;
            }
     }//if this is the first read, the largest value and its location are
stored so they are not lost in recursion

     printf("\nFor recursive call #%d the largest value was %d found in
the %d quadrant\n", count+1, biggest.value, quadrant);
            //print highest value found with each call and print the
quadrant it was found in
     if (choice==1){
                  for(i=0;i<n;i++){
                              for (j=0;j<n;j++){
                                    if(quadrant==1)
                                          printf("%d ", quad1[i][j]);
                                    else if (quadrant==2)
                                          printf("%d ", quad2[i][j]);
                                    else if (quadrant==3)
                                          printf("%d ", quad3[i][j]);
                                    else
                                          printf("%d ", quad4[i][j]);
                              }
                              printf("\n");
                  }
     }//print the quadrant with the highest value

     if (n==1)
            return biggest; //return the largest value found when the
matrix is 1X1

     if (quadrant==1)
            recursive_call(quad1, n, count+1, 0, 0,choice, biggest);
       else if (quadrant==2)
            recursive_call(quad2, n, count+1, 0, 0, choice, biggest);
       else if (quadrant==3)
            recursive_call(quad3, n, count+1, 0, 0, choice, biggest);
       else
            recursive_call(quad4, n, count+1, 0, 0, choice, biggest);
//until matrix is 1X1 recursively call and make new matrices of m/2Xm/2
zooming in on max value
}
```