Lab 8
Eight Number in a Cross

In the solution to this problem, use the backtracking scheme that we covered in class. **Write a program** which allocates the integers 1-8 to the squares in the figure above, subject to the restrictions that no two adjacent squares contain consecutive integers. By adjacent we mean vertically, horizontally, or diagonally.
Eight Number in a Cross

By-Hand Algorithm to Obtain Adjacency Table

1) Label cross boxes arbitrarily from 0 through 7.
2) for each box $i = 0, 1, ..., 7$:
   for each adjacent box $k$ (such that $k < i$):
     add $k$ to $i$’s neighbor list
     add -1 to $i$’s neighbor list
3) From the adjacency table obtained, create a look up in the form of a 2D array

Cross:

```
  1  4
  0  2  5  7
  3  6
```
<table>
<thead>
<tr>
<th>Box #</th>
<th>Neighbor List</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>0, -1</td>
</tr>
<tr>
<td>2</td>
<td>0, 1, -1</td>
</tr>
<tr>
<td>3</td>
<td>0, 2, -1</td>
</tr>
</tbody>
</table>

Diagram: [Graph of box connections]
<table>
<thead>
<tr>
<th>Box #</th>
<th>Neighbor List</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>0, -1</td>
</tr>
<tr>
<td>2</td>
<td>0, 1, -1</td>
</tr>
<tr>
<td>3</td>
<td>0, 2, -1</td>
</tr>
<tr>
<td>4</td>
<td>1, 2, -1</td>
</tr>
</tbody>
</table>

*Diagram of a 3x3 grid with boxes labeled 0 to 7.*
<table>
<thead>
<tr>
<th>Box #</th>
<th>Neighbor List</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>0, -1</td>
</tr>
<tr>
<td>2</td>
<td>0, 1, -1</td>
</tr>
<tr>
<td>3</td>
<td>0, 2, -1</td>
</tr>
<tr>
<td>4</td>
<td>1, 2, -1</td>
</tr>
<tr>
<td>5</td>
<td>1, 2, 3, 4, -1</td>
</tr>
<tr>
<td>6</td>
<td>2, 3, 5, -1</td>
</tr>
</tbody>
</table>

- 0: 2, 5, 7
- 1: 3, 6
- 2: 5
- 3: 3, 6
- 4: 1, 7
- 5: 2
- 6: 3
- 7: 1, 4
The Adjacency Table in C++ (Based on the Labeling Above)

//5 columns because the largest neighbor list contains 5 numbers in our case
int adj_table[8][5] = {
    {-1},  // box 0
    {0, -1},  // box 1
    {0, 1, -1},  // box 2
    {0, 2, -1},  // box 3
    {1, 2, -1},  // box 4
    {1, 2, 3, 4, -1},  // box 5
    {2, 3, 5, -1},  // box 6
    {4, 5, 6, -1}  // box 7
};
Psuedo-Code for ok function

```cpp
bool okay(int cross[], int box) {
    static int adj_table[8][5] = { ... };
    do eight-queens style row-check here; if the test fails, return false
    //consecutive number check, notice that we only care to check the
    //neighbors
    //of the current box, hence the need for the adjacency table
    i = 0;
    neighbor_box = adj_table[box][i]; //this is the index of an adjacent box!
    while (neighbor_box != -1) {
        if (value at neighbor_box and cross[box] are consecutive)
            return false;
        move on to next neighbor box
    }
    return true;
}
```