Executive Summary
In this project, you will recreate a version of the classic game Connect 4 said to have been played by the Royal Navy’s Captain Cook during his many extended at sea periods. In your version of the game, two human players can play the game, or, since we are after all a highly technical navy, one human player can play against the computer. For a way more souped up version of Connect 4 than you would ever need to code up, take a look here.

Due Dates and Honor
Steps 1 and 2 of this project will be due by the “close of business” on Wed April 21, 2010.  
- Meeting this milestone is worth 10 points of your final grade.  
- Code must be submitted electronically via Blackboard and in writing:  
  o Blackboard – submit your code under step 2.  
  o Written – Turn in a copy of your code and the screenshot described under Step 2 below  
  o Style/Comments are encouraged but not graded at this point.

The entire project will be due by the “close of business” on Fri April 30, 2010. See the course policy for details about due dates and late penalties.

Again, this is a Programming Project, and it is very important that you understand and abide by the Department's policy concerning programming projects. In particular, this means that you may give no assistance to and receive no assistance from anyone except your instructor. In addition, you may not use any Internet resources except for the course notes. For more details, please view: http://www.cs.usna.edu/academics/ProgrammingPolicy.pdf

Important Notes Up Front
1. You must submit your work on Blackboard as you go (after completing each step) -- not just the final product.  
2. You must write and use multiple different functions. Part of your grade depends upon logical use of functions for separate logical pieces of your program.  
3. No function (including main) may be more than two pages long (when printed out with a 10 pt font), and each function (other than main) must encapsulate a single, logical piece of functionality. Do NOT write does-everything-but-wipe-the-kitchen-sink type functions. If your function is too long, consider how it can itself be broken into distinct sub-sets, and think about making each of these sub-sets a function.  
4. You may NOT use “two-dimensional” arrays for this project. Instead, you will use a one-dimensional array of “bins” (and each bin will contain an array of integers). More details below.

Details
The purpose of this project is primarily to give you an opportunity to use structs, functions and arrays, and to hopefully have a little fun in the process. The project is divided up into multiple steps. You are strongly encouraged to solve each step completely - including thorough testing - before you move on to the next step. However, some steps can be solved out of order – it is permissible to move on to a later step if you get
Overview of the Connect 4 game. Connect 4 is a board game where two players take turns dropping disks into vertical columns (or bins). The objective of each player is to get 4 of its own disks in a row on a single horizontal, vertical, or diagonal line. Here is an example board of a game in progress. It is Player A’s turn to make a move and there are 6 disks in the bins: three from Player A and three from Player B.

<table>
<thead>
<tr>
<th>bin</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The board you will use has 7 columns (or bins) each with 6 rows. Each player chooses a bin, and drops its disk into that bin. The disk will “fall” to the bottom-most unoccupied row. Example: If player A chooses bin 6, the disk will fall to the bottom row and that cell will become a “A”. You do not have to show the disks actually falling into the bins, just where they rest once placed in a bin.

Step 0 (10 pts): Milestone

Meeting the milestone described above is worth 10 pts of your final grade.

Step 1 (20 pts): Create and display the board

a) Create a struct Bin that contains the following data members:

- a one dimensional array of chars, to simulate one column of the board,
- an integer to store the size of the column (number of rows),
- an integer to store the index of the bottom-most unoccupied row (i.e. the next open space) for that bin,
- a Boolean to store the answer to whether or not the bin is full of disks, and
- whatever other data members your program needs (but remember, your program may not use any two dimensional arrays!)

b) REQUIRED: Put a comment in the definition of struct “Bin” that explains how you are interpreting the array of integers. In particular, what index into the array corresponds to the “top” row and what index corresponds to the “bottom” row of the board? There are several different answers that will work here, but you must decide and then be consistent about it! The comment is required here to help you (and to help us understand your code).
c.) Using your class Bin, create an array of bins to simulate the entire board and design a function to initialize an empty board (all bin cells to spaces).

d.) Write a function that displays the game board with empty cells initially shown in each bin, as in the below figure. Be sure to include your name in the intro message of your program.

![Game Board Screenshot](image)

**Step 2 (25 pts): Basic game play**

Create a program to play the game. Allow for two human players that alternate taking turns. Whenever the first player (player A) moves by specifying a bin number, place an A in the last unoccupied row of that bin; place a B wherever player B moves. Each move must be to a bin with some empty space in it, so your program must enforce valid moves (i.e., a player cannot add their letter to a full bin). Display the board at the beginning of each game and after every move. Note that this step does **not** require you to detect whether or not someone has won the game (that is the subject of a later step).

See next page for a screenshot…
Step 3 (30 pts): Determining a winner

Modify your solution so that after each move, your program determines if the game has been won (4 of the same letter adjacent to each other anywhere in a single line along a row, column, or diagonal) or if the game is a draw (the board is full), providing appropriate screen messages in each case. If no winner or draw is detected, your program is to continue the play.

Note #1: you may find it simplest to separately consider each case – is there a win along a column? Along a row? Along a diagonal?

Note #2: do NOT write a huge amount of code for this step! It will require some complexity, but you should NOT have separate code for checking row 1, code for checking row 2, code for checking row 3, etc. Instead, make sensible use of loops.
Note #3: Be very careful not to access any array indices that do not exist! Accessing such values will cause you to lose points (and may cause your program to crash).

You should carefully test your program to make sure it works in every case. To help you get started, here are some test cases you could use (don’t type them! Paste into the console instead.):

- Winning diagonal sequence: 4 3 3 2 1 2 1 6 1
- Different diagonal sequence: 1 2 3 4 4 5 4 4 4 5 5 5
- A draw: 1 2 1 2 1 2 1 2 1 3 4 3 4 3 4 4 3 4 3 5 6 5 6 5 6 5 6 5 5 7 7 7

The test cases above are just a start – you should also write your own.

The below shows a sample of a program detecting a column win:

Step 4 (15 pts): Human vs. Computer

Modify your Step 3 solution so that at the beginning of a game, the user is asked if he or she wants to play against the computer (yes/no). If the answer is “no”, the basic game is played. If the answer is “yes”, then ask if the human user wants to go first (yes/no). Then play the game, with the computer automatically selecting moves for player A or player B as appropriate. The computer moves need not be “intelligent” moves (that is the subject of part of the extra credit), but moves by the computer must be valid moves (i.e. to non-full bins.). Make your program work like the sample shown below:
Step 5 (Extra Credit up to 15 points): Your program must meet Step 4 requirements in order to go for the extra credit opportunities described below. Include a discussion of any special commands or key strokes that your program uses (these should be briefly included in the start up menu of your program wherever possible).

a) Demo Version – 5 points: Modify your Step 4 solution so that, if the user chooses this option, the computer makes the moves for both players. You must pause for a second or two between moves so an observer can watch the game as it unfolds. This would be a good spot to include seeing the disks in the act of falling into their resting spots within the bins.

b) Intelligent Automatic Player – 5 points: Modify your Step 4 solution so that the computer player operates with at least a basic attempt to win (i.e. if there is a single move that can be made that leads to a win on that turn of play, the computer will take it.) More points are possible for a very sophisticated player.

c) Dynamic board – 5 points: Modify your Step 4 solution so that the user can select how big the game board is (number of rows and columns) AND how many contiguous disks are necessary for a win (e.g. so we can play Connect 6 on a bigger board).
Friendly Reminder:

- You MUST submit a working solution for EACH of the above steps as you complete them. Submit just the project3.cpp file. It’s okay if you make some changes to a step after you submit it, but we want to see the progression of your steps. The individual steps will not be graded for style or functionality – as long as it looks reasonably like you submitted the steps, what counts will be the style and functionality of the final result.

Important grading points:

- The default method of passing variables is pass by value.
- Appropriate use of functions to implement this program is critical to receiving the maximum possible score.
- If your program does not compile as submitted, you will receive a zero.
- Your program must read input and write output in the exact format specified in this project description.
- Your program’s source code must comply with the Required Style Guide in order to maximize the grade you receive on this project.
- Your grade will also depend on the reasonable use of C++ code. Don’t use 50 lines of code where 5 would do.

There will be both a paper and an electronic part to your submissions. The paper submission can be handed to your instructor or slid under their office door (but do not put your project in their mailbox). For the purposes of any late penalties, your project is not considered submitted until your instructor receives BOTH the electronic and paper portions of your submission. Any disagreement between your paper and electronic submissions will result in a grade reduction.

Electronic submission:

Unless otherwise specified by your instructor, your electronic submission should be made via Blackboard. As you complete each step you should submit your project3.cpp file. Submitting steps out of order is completely acceptable! Simply note the out of order submission in the comments at the top of your code with your name, i.e., “Submitting Step 4, step 3 has not yet been completed”. When completely finished, submit your project3.cpp file under the “Final submission” link on Blackboard.

If you desire to submit a revised solution to any completed step notify your instructor via email so they can clear your previous submission. However, unless changes are dramatic this is not required for intermediate steps (see “Friendly Reminder” above) – the final submission is what is graded.

Paper submission:

Staple in this order:

- Cover sheet, with signed honor statement
- Printout of your code. Make sure the indentation is correct on the printout! See notes on printing on the course homepage
- Screenshots for each step that works (see below for the required screenshots for each step). Be sure to LABEL each one (e.g., “STEP 3 – column win”).

Important note regarding screenshots:

- Each step will have specific requirements for the screenshot. Make sure to follow these requirements for full credit.

- For step 1: Ensure the output displays “The sum is: 17”.
- For step 2: Confirm the output shows “The largest number is: 31”.
- For step 3: Verify the output reads “The smallest number is: 17”.
- For step 4: The screenshot should demonstrate the correct reader OID.
- For step 5: Confirm the output correctly displays “Busy 20%”.

- For step 6: Ensure the output reflects the accurate display at the monitor locations.

- For step 7: Verify the output accurately shows the change in the number of employees.
Screenshots to turn in (include ALL that apply):

- **Step 1**
  - Screenshot showing the initial board being printed (only required if you don’t get Step 2 working)

- **Step 2**
  - Screenshot showing the user being prevented from placing a disk in a full bin
  - Screenshot showing the sequence described in the Step 2 instructions (only required if you don’t get Step 3 working)

- **Step 3** (you do NOT have to show the entire game in your screenshots – have your screenshot include just the last 1-2 moves)
  - Screenshot showing a column win
  - Screenshot showing a row win
  - Screenshot showing a positive slope diagonal win
  - Screenshot showing a negative slope diagonal win
  - Screenshot showing a draw

- **Step 4**
  - A screenshot of your program running showing at least 2 moves made by the computer

- **Step 5** (extra credit)
  - Include screenshots that show off what your extra credit can do. Also include a discussion of any special commands or keystrokes that your program uses (these should be included in the startup menu of your program wherever possible).

**TIPS:**

- Follow the Style Guide as you go! It’s easier to indent and comment as you write than at the end, and you’ll make it easier on yourself to write your program. And you’ll maximize your grade by following the guidelines.
- Compile often so you don’t end up with a snake’s nest of errors all at once
- Save backups of project3.cpp in case your sponsor’s dog eats your X drive.
- **Start early!** Allow time for you to get new ideas and to see the instructor for help.
- Remember that this project is not to be discussed with anyone besides your instructor!
  - Don’t defeat your learning AND jeopardize the honor of yourself or your friends by talking to them about it.