Int i,j;

    cout << "\n";

    cout << "\t\t"<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";

    coint ut << "\n\t\t"<<"|   |   |   |   |   |   |   |  ";

    for ( i=0; i <7; i++)

    {

        cout << "\n" <<"\t\t" <<"|";

        for (j=7; j > 0; j--)

            cout <<"\_\_\_|" ;

            cout <<"\n\t        |   |   |   |   |   |   |   |";

    }

        cout << "\n\t\t"<<"| 1 | 2 | 3 | 4 | 5 | 6 | 7 |";

    cout << "\n\n";

+---+---+---+---+---+---+---+

|   |   |   |   |   |   |   |

+---+---+---+---+---+---+---+

|   |   |   |   |   |   |   |

+---+---+---+---+---+---+---+

|   |   |   |   |   |   |   |

+---+---+---+---+---+---+---+

|   |   |   |   |   |   |   |

+---+---+---+---+---+---+---+

|   |   |   | O |   |   |   |

+---+---+---+---+---+---+---+

|   |   |   | X |   |   |   |

+---+---+---+---+---+---+---+

|   |   |   | X | O |   |   |

+---+---+---+---+---+---+---+

  1   2   3   4   5   6   7

+---+---+---+---+---+---+---+

|   |   |   |   |   |   |   |

+---+---+---+---+---+---+---+

|   |   |   |   |   |   |   |

+---+---+---+---+---+---+---+

|   |   |   |   |   |   |   |

+---+---+---+---+---+---+---+

|   |   |   |   |   |   |   |

+---+---+---+---+---+---+---+

|   |   |   | O |   |   |   |

+---+---+---+---+---+---+---+

|   |   |   | X |   |   |   |

+---+---+---+---+---+---+---+

|   |   |   | X | O |   |   |

+---+---+---+---+---+---+---+

  1   2   3   4   5   6   7

Copyright (c) British Informatics Olympiad, 2004

  BIO 2004 Question 2 "Four in a line"

  Question by Richard Forster

  Example solution by Antony Rix.

  This program may be freely copied by persons or organisations

  involved in the British Informatics Olympiad or the International

  Olympiad in Informatics, on condition that no changes are made and

  this notice is not altered or removed. Distribution for profit is

  forbidden unless permission is first obtained in writing from the BIO.

  This program is for educational purposes only and comes with no

  warranty, implied or otherwise, as to its fitness for any purpose.

<http://www.olympiad.org.uk/>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

/\* The player whose turn it is to play next \*/

int Player;

/\* The number of moves made \*/

int Moves;

/\* The board.  0 represents no piece, 1 player 1's piece, 2 player 2's piece. \*/

int Board[7][7];

/\* Note that as this is C, all array indices count from zero rather than 1. \*/

/\* Initialise the board to all 0's \*/

void InitBoard( void ) {

    int i, j;

    for( i = 0; i < 6; i++ ) for( j = 0; j < 7; j++ ) Board[i][j] = 0;

    Moves = 0;

}

/\* Show the board in the required format \*/

void ShowBoard( void ) {

    int i, j;

    for( i = 0; i < 6; i++ ) {

        for( j = 0; j < 7; j++ )

            if( Board[i][j] == 0 ) printf("-");

            else if( Board[i][j] == 1 ) printf("\*");

            else if( Board[i][j] == 2 ) printf("o");

        printf("\n");

    }

}

/\* Test whether the current move wins the game \*/

int WinsGame( int i0, int j0 ) {

    int i, j;

    /\* Test horizontal lines of 4 pieces including the current one \*/

    i = i0;

    j = j0-3; if( j < 0 ) j = 0;

    while( (j <= j0) && (j <= 3) ) {

        if( (Board[i][j] == Board[i0][j0]) &&

            (Board[i][j+1] == Board[i0][j0]) &&

            (Board[i][j+2] == Board[i0][j0]) &&

            (Board[i][j+3] == Board[i0][j0]) )

            return 1;

        /\* Not won - try the next starting position \*/

        j++;

    }

    /\* Test vertical lines of 4 pieces including the current one \*/

    j = j0;

    i = i0-3; if( i < 0 ) i = 0;

    while( (i <= i0) && (i <= 2) ) {

        if( (Board[i][j] == Board[i0][j0]) &&

            (Board[i+1][j] == Board[i0][j0]) &&

            (Board[i+2][j] == Board[i0][j0]) &&

            (Board[i+3][j] == Board[i0][j0]) )

            return 1;

        /\* Not won - try the next starting position \*/

        i++;

    }

    /\* Test \-diagonal lines of 4 pieces including the current one \*/

    j = j0-3; i = i0-3;

    while( (j < 0) || (i < 0) ) { j++; i++; };

    while( (i <= i0) && (i <= 2) && (j <= j0) && (j <= 3) ) {

        if( (Board[i][j] == Board[i0][j0]) &&

            (Board[i+1][j+1] == Board[i0][j0]) &&

            (Board[i+2][j+2] == Board[i0][j0]) &&

            (Board[i+3][j+3] == Board[i0][j0]) )

            return 1;

        /\* Not won - try the next starting position \*/

        j++; i++;

    }

    /\* Test /-diagonal lines of 4 pieces including the current one \*/

    j = j0+3; i = i0-3;

    while( (j > 6) || (i < 0) ) { j--; i++; };

    while( (i <= i0) && (i <= 2) && (j >= j0) && (j >= 3) ) {

        if( (Board[i][j] == Board[i0][j0]) &&

            (Board[i+1][j-1] == Board[i0][j0]) &&

            (Board[i+2][j-2] == Board[i0][j0]) &&

            (Board[i+3][j-3] == Board[i0][j0]) )

            return 1;

        /\* Not won - try the next starting position \*/

        j--; i++;

    }

    /\* No, this has not won the game \*/

    return 0;

}

/\* Apply the rules to make the next move, and exit if the game is over \*/

void DoMove( void ) {

    int i, j;

    /\* Rule 1 - if we can win immediately, do so. \*/

    for( j = 0; j < 7; j++ ) {

        if( Board[0][j] == 0 ) {

            /\* Find the first place that we can go \*/

            i = 0;

            while( (i < 6) && (Board[i][j] == 0) ) i++;

            i--;

            /\* See if going here wins \*/

            Board[i][j] = Player;

            if( WinsGame(i, j) ) {

                ShowBoard();

                printf("Player %d wins\n", Player);

                exit(0);

            } else

                /\* Erase this move and try the next \*/

                Board[i][j] = 0;

        }

    }

    /\* Rule 2 - if the other player can win immediately, go there. \*/

    for( j = 0; j < 7; j++ ) if( Board[0][j] == 0 ) {

        /\* Find the first place that we can go \*/

        i = 0;

        while( (i < 6) && (Board[i][j] == 0) ) i++;

        i--;

        /\* See if going here would win for the other player \*/

        Board[i][j] = 3-Player;

        if( WinsGame(i, j) ) {

            /\* We go there \*/

            Board[i][j] = Player;

            return;

        } else

            /\* Erase this move and try the next \*/

            Board[i][j] = 0;

    }

    /\* Rule 3 - go in first available space. \*/

    for( j = 0; j < 7; j++ ) if( Board[0][j] == 0 ) {

        /\* Find the first place that we can go \*/

        i = 0;

        while( (i < 6) && (Board[i][j] == 0) ) i++;

        i--;

        Board[i][j] = Player;

        return;

    }

}

/\* Function to solve Q2a \*/

void Q2a( void ) {

    char c;

    int n, count, i, j;

    Player = 1;

    InitBoard();

    /\* Play the initial sequence of moves \*/

    /\* Read the number of initial moves \*/

    scanf("%d", &n);

    /\* Do the initial moves \*/

    for( count = 0; count < n; count++ ) {

        /\* Get the column for the next move \*/

        scanf("%d", &j);

        /\* Reduce by 1 since we're counting from 0, but the question counts from 1 \*/

        j--;

        /\* Find the next position we can go \*/

        i = 0;

        while( (i < 6) && (Board[i][j] == 0) ) i++;

        i--;

        Board[i][j] = Player;

        Player = 3-Player;

        Moves++;

    }

    ShowBoard();

    /\* The main loop where we get an instruction and process it \*/

    while( Moves < 42 ) {

        scanf("%c", &c);

        if( (c == 'n') || (c == 'N') ) {

            /\* Play the next move \*/

            DoMove();

            ShowBoard();

            Moves++;

            Player = 3-Player;

        } else if( (c == 'r') || (c == 'R') ) {

            while( Moves < 42 ) {

                DoMove();

                Moves++;

                Player = 3-Player;

            }

        }

    }

    /\* Neither player has won and the board is now full. \*/

    ShowBoard();

    printf("Draw\n");

    scanf("%c", &c);

}

/\* Count of the number of combinations for Q2d \*/

long Npatterns;

/\* Recursive helper function to count the number of combinations for Q2d \*/

void TryPos( int Column, int Remaining ) {

    int This;

    if( Column == 8 ) {

        if( Remaining == 0 ) {

            /\* This is a valid pattern with all 21 pieces placed, so add to the count \*/

            Npatterns++;

        }

    } else {

        /\* Try to place between 0 and 6 pieces in Column, then recurse to try the next \*/

        This = 0;

        while( (This <= 6) && (This <= Remaining) ) {

            TryPos(Column+1, Remaining-This);

            This++;

        }

    }

}

/\* Function to solve Q2d \*/

void Q2d( void ) {

    Npatterns = 0;

    TryPos(1,21);

    printf("Number of distinct patterns for Q2d is %d\n", Npatterns);

}

/\* Main function called from the command-line \*/

void main( void ) {

    Q2a();      /\* Edit this line to Q2d() to solve 2d \*/

}

/\* End of program \*/