Programming assignment #2

ENG 101 Winter '05
Due Monday Feb 7th at 11:59pm.

For this assignment, you are asked to write three functions. One function involves arrays, another arrays of structures and the last one a multi-dimensional array. For two of the functions a sketch of an algorithm is provided. In some cases, code which implements similar algorithms have been discussed in class. This project is worth 3% of your course grade. The match() and stuSort() functions are worth 30% while the check_ttt() function is worth 40%.

You will turn it in by creating a directory named "P2" in your "eng101" (note the lower case) directory. You will turn in two files. One named "p2functions.cc" and the other named "p2main.cc". Your main can simply be a copy of the one from the website. Your p2functions.cc file must not have a main in it. Your p2functions.cc must compile with the p2main.cc found on the web, or you may receive zero points!

```
int match(int list1[], int list2[], int size)
```

Description:

This function takes two ***sorted*** lists, both of which have exactly "size" elements. The lists have the following properties:

- They are sorted with the smallest value found at index 0.
- Neither list will have repeated values. (The same value can't occur twice in the same list)

The function is to return the number of elements shared by the two lists.

Algorithm:

Not given.

Hint:

You can take advantage of the fact the lists are sorted. Try writing lists out by hand and seeing how to check for matches. Also, you don't have to check if the lists are sorted, you may assume that they are.

Sample results:

if A[0]=4, A[1]=6, A[2]=8, A[3]=10 and B[0]=0, B[1]=4, B[2]=10, B[3]=22 then
match(A,B,4) would return 2.
match(A,A,4) would return 4.

```
struct student
{
    int sid;
    double gpa;
};
```

`void stuSort(student stu[], int numStu)`

Description:

This function takes two arguments, an array of type student and an integer that specifies the number of elements in the array. The function sorts the array by sid (student ID number).

Algorithm

You can use either sort algorithm we've already discussed (bubble or selection) or you may use a different sorting algorithm (which we don't recommend, but its up to you). The only trick is that you are sorting on a field of a structure.

Comments:

First of all, you are allowed to use any of the code I've supplied as a starting point. Secondly, the changes needed are really very minor. Figuring out what changes need to be made may be challenging. We strongly suggest that you think a fair bit before you code.

`int check_ttt(int A[][3])`

Description:

This function takes a 3 by 3 array of integers, treats the integers as either "X", "O" or blanks on a tic-tac-toe board and checks to see who, if anyone, has won. If you aren't familiar with tic-tac-toe, you might want to look here: http://en.wikipedia.org/wiki/Tic-tac-toe.

The integers are to be treated as blanks if the value is a 0, an "X" if the value is a 1 and a "O" if the value is a 2. As an example, the values:
A[0][0]=1, A[0][1]=0, A[0][2]=2, A[1][0]=0, A[1][1]=1, A[1][2]=0, A[2][0]=0, A[2][1]=2, A[2][2]=0 would correspond to:

X		O
	X	
	O	

The function is to examine the board and check to see if either player has won. If "X" has won, return a 1, if "O" has won, return a 2, if neither has won, return a 0. If both players have won you may return either a 1 or a 2.

Hints:

There is some new C++ syntax that might be helpful here. The statement:

```
((i==0) && (j==1))
```

is only true if both statements are true (so i equals 0 *and* j equals 1). An example bit of code to check to see if X has taken one of the diagonals is:

```
if((A[0][0]==1)&&(A[1][1]==1)&&(A[2][2]==1))
```

Comments:

First of all, this one is a bit long (and tedious). There are 8 possible ways to win (3 horizontal, 3 vertical, 2 diagonals) and you need to check this for both players. There are ways to greatly reduce the amount of work, but you don't need to use them.

Secondly, a few comments about the syntax of the "if" statements used above. For one thing, the parenthesis around things like `(i==0)` are not actually needed. The order of operations C++ uses causes all == operators to be evaluated before the "&&" operator. But in general, using the parenthesis makes the code more readable. Also, if you want the statement to be evaluated as "true" if *either* condition is true you can use the "||" operator. So `((i==0) || (j==1))` would be true if i equals zero *or* j equals zero.

Sample results:

For the array described above (the one with the figure), the return result should be zero.