

This sheet must be the cover page for every programming assignment.

Name Joe Student

Assignment Title May the Greatest Integer Win

(do not write below this line; for grader use only)

Validity _____(up to 70%)

Matches assignment specification precisely
Uses appropriate data structures
Provides sufficient sample runs
Produces correct results
Algorithm is correct and as specified
Output matches code
No bugs

User-friendliness _____(up to 5%)

Data structure display is visually clear
Correct spelling and grammar
Neat, appropriately commented output
Clear error messages
Grammatical output
Robust
Demonstrates adequate testing

Documentation _____(up to 5%)

Introductory comments
Adequate individual routine comments (including pre/post conditions)
Clear comments
Neat, professional presentation
File names at top of each file
Header files before implementation files
User instructions provided

No-run conditions _____(up to 5%)

Programming style _____(up to 5%)

Good identifier names
Good indentation
Easy to follow
Main program precedes details
Efficient approach
Public before private in class
Good structure

Modularity _____(up to 10%)

Modularized to facilitate reuse
Sufficiently modular
Separate header and implementation files
Main program brief and clear
Header/implementation file pair for each class
Appropriate use of classes and templates
Appropriate use of functions

Extra credit _____

Total _____

Program Analysis Statement

Problems I encountered during the program: I didn't have any difficulty with this program. If I had, I would describe that difficulty here. For example, I might've had trouble opening the input file and/or reading its contents into an array of integers. If so, I would describe that problem here.

No Run Conditions: My program does not run under a few conditions:

- A file called "input.txt" must exist in the folder which contains the program executable. The input file (input.txt) must contain, on its first line, the number of spaces on the board. If the number of spaces on the board is given as, say, 10, then there must be ten additional lines in the file, each containing a positive integer.

Extra Credit Features: There are no extra credit features in my program, but if there had been I would describe them here.

STL Code/Code From Text: I didn't use an STL or any code from the textbook. If I had, I would document the use of that code here.

Program Design (Top-Down Design with Stubs)

```
main {
    int board[MAX_NUM_BOARD_SPACES];
    int p1Sum, p2Sum;
    int numBoardSpaces;

    GetInputFileContents ("input.txt", board, numBoardSpaces);
    PlayGame (board, numBoardSpaces, p1Sum, p2Sum);
    StoreResultsInFile ("output.txt", p1Sum, p2Sum);
}

/*purpose: To open the given input file and store the contents in the given array of ints
preconditions:      InFileName contains the path to an existing file. The file contains
                    one integer on each line; the first line contains the number of board
                    spaces and the rest of the lines each contain a board space value.
postconditions:     OutBoard will contain N positive integers where  $1 \leq N \leq 200$ ;
                    OutNumBoardSpaces will contain the number of board spaces (i.e., N); function
                    will return true if the file was opened correctly, false otherwise
*/
bool GetInputFileContents (string InFileName, int OutBoard[], int& OutNumBoardSpaces)
{
}

/*purpose: To play the game to completion as described in the rules: the two players move alternately.
When a player moves, he selects a board space. The selected number is deleted from the board. The game
is over when all numbers have been selected. The first player wins if the sum of the numbers he has
selected is at least as much as selected by the second player. Otherwise, the second player wins.
preconditions:      InBoard should contain N positive integers, where  $1 \leq N \leq 200$ 
postconditions:     OutP1Sum will contain the sum of the numbers chosen by the
                    player 1; OutP2Sum will contain the sum of the numbers chosen
                    by player 2
*/
void PlayGame (int InBoard[], int N, int& OutP1Sum, int& OutP2Sum)
{
    int player1Move, player2Move;
    player1Move = GetBestRemainingMove (InBoard, N);
    MakeMove (player1Move, InBoard, OutP1Sum);
    player2Move = GetBestRemainingMove (InBoard, N);
    MakeMove (player2Move, InBoard, OutP2Sum);
}

/*purpose: To store the results of the game in an output file.
preconditions:      InOutputFileName contains the path to a file in which output will be written.
                    InP1Sum contains the score of the first player (>0). InP2Sum is similar.
postconditions:     The file named by InOutputFileName will be made to contain "InP1Sum,
                    InP2Sum"; function will return true if the file was opened correctly, false
                    otherwise
*/
bool StoreResultsInFile (string InOutputFileName, int InP1Sum, int InP2Sum)
{
}
}
```

/*purpose: Finds the largest integer left on the board.
preconditions: InBoard should contain N positive integers, where $1 \leq N \leq 200$; N should be the number of spaces on the board
postconditions: returns the index of the largest integer remaining on the board (a number between 0 and N-1, inclusive)
*/

```
int GetBestRemainingMove (int InBoard[], int N)
{
}
}
```

/*purpose: Makes the move specified by InPlayerMove.
preconditions: OutBoard should contain N positive integers, where $1 \leq N \leq 200$; InPlayerMove should be between 0 and N-1; OutPlayerSum should contain the current player's score before the move is made
postconditions: OutBoard[InPlayerMove] will be set to -1 (indicating the space has been "played"); OutPlayerSum will be incremented by OutBoard[InPlayerMove]
*/

```
void MakeMove (int InPlayerMove, int OutBoard[], int& OutPlayerSum)
{
}
}
```

Program Output

The output file contains the score of player 1 and then the score of player 2, separated by a comma.

Output File 1

42, 24

End of Output File 1

If there had been extra credit, I would also list its output on this page.

Program Input

The input file contains the number of spaces on the board (must be between 2 and 200) on its first line. The rest of the file contains board-space values, each a positive integer and each on a separate line in the file.

Input File 1

6

12

8

2

9

13

22

End of Input File

If there had been extra credit, I would also list its input on this page.

Program Source Code

```

/*
Author:                               Szenher
Programming Language:                  C++
Compiler:                              Microsoft Visual C++ 6.0
Preferred Machine:                     Any with Windows 98 or later
Date Written:                           2/2/2002
*/

#include "stdafx.h"
#include <fstream>
#include <string>
using namespace std;

//function prototypes
bool GetInputFileContents (string InFileName, int OutBoard[], int& OutNumBoardSpaces);
void PlayGame (int InBoard[], int N, int& OutP1Sum, int& OutP2Sum);
bool StoreResultsInFile (string InOutputFileName, int InP1Sum, int InP2Sum);
int GetBestRemainingMove (int InBoard[], int N);
void MakeMove (int InPlayerMove, int OutBoard[], int& OutPlayerSum);

const int MAX_NUM_BOARD_SPACES = 200; //the maximum number of board spaces

int main () {
    int board[MAX_NUM_BOARD_SPACES];
    int p1Sum, p2Sum;
    int numBoardSpaces;

    //initialize player sums
    p1Sum = 0;
    p2Sum = 0;

    GetInputFileContents ("input.txt", board, numBoardSpaces);
    PlayGame (board, numBoardSpaces, p1Sum, p2Sum);
    StoreResultsInFile ("output.txt", p1Sum, p2Sum);

    return 0;
}

/*
purpose: To open the given input file and store the contents in the given array of ints
preconditions: InFileName contains the path to an existing file. The file contains
                one integer on each line; the first line contains the number of board
                spaces and the rest of the lines each contain a board space value.
postconditions: OutBoard will contain N positive integers where 1<=N<=200;
                OutNumBoardSpaces will contain the number of board spaces
                (i.e., N); function will return true if the file was opened
                correctly, false otherwise
*/
bool GetInputFileContents (string InFileName, int OutBoard[], int& OutNumBoardSpaces)
{

```

```

//open the input file
ifstream ifs;
ifs.open(&InFileName[0], ios::in);
if (!ifs)
    return false; //if file doesn't open then return false
else {
    //get the number of board spaces in the file
    ifs >> OutNumBoardSpaces;

    //get the board values
    for (int i = 0; i < OutNumBoardSpaces; i++)
        ifs >> OutBoard[i];

    ifs.close();
    return true; //if file did open then return true
}
}

/*
purpose: To play the game to completion as described in the rules: the two players move
alternately. When a player moves, he selects a board space. The selected number is deleted
from the board. The game is over when all numbers have been selected. The first player wins if
the sum of the numbers he has selected is at least as much as selected by the second player.
Otherwise, the second player wins.
preconditions:      InBoard should contain N positive integers, where 1<=N<=200
postconditions:     OutP1Sum will contain the sum of the numbers chosen by the
                    player 1; OutP2Sum will contain the sum of the numbers chosen
                    by player 2
*/
void PlayGame (int InBoard[], int N, int& OutP1Sum, int& OutP2Sum)
{
    int player1Move, player2Move;

    //loop until all board spaces have been chosen
    for (int i = 0; i < N/2; i++) {

        //player 1 chooses and makes a move
        player1Move = GetBestRemainingMove (InBoard, N);
        MakeMove (player1Move, InBoard, OutP1Sum);

        //player 2 chooses and makes a move
        player2Move = GetBestRemainingMove (InBoard, N);
        MakeMove (player2Move, InBoard, OutP2Sum);
    }
}

/*
purpose: To store the results of the game in an output file.
preconditions:  InOutputFileName contains the path to a file in which output will be written.
                InP1Sum contains the score of the first player (>0). InP2Sum is similar.
*/

```

postconditions: The file named by InOutputFileName will be made to contain "InP1Sum, InP2Sum"; function will return true if the file was opened correctly, false otherwise.

```
*/
bool StoreResultsInFile (string InOutputFileName, int InP1Sum, int InP2Sum)
{
    ofstream ofs;
    ofs.open(&InOutputFileName[0], ios::out);
    if (!ofs)
        return false; //if file didn't open then return false
    else {
        ofs << InP1Sum << ", " << InP2Sum << endl; //output scores to file
        return true; //if file opened, then return true
    }
}
```

/*

purpose: Finds the largest integer left on the board.

preconditions: InBoard should contain N positive integers, where $1 \leq N \leq 200$; N should be the number of spaces on the board

postconditions: returns the index of the largest integer remaining on the board (a number between 0 and N-1, inclusive)

*/

```
int GetBestRemainingMove (int InBoard[], int N)
{
    int bestRemainingMove = 0;

    //loop through all board spaces to find the largest value remaining
    for (int i = 1; i < N; i++)
    {
        if (InBoard[bestRemainingMove] < InBoard[i])
            bestRemainingMove = i;
    }

    return bestRemainingMove;
}
```

/*

purpose: Makes the move specified by InPlayerMove.

preconditions: OutBoard should contain N positive integers, where $1 \leq N \leq 200$; InPlayerMove should be between 0 and N-1; OutPlayerSum should contain the current player's score before the move is made

postconditions: OutBoard[InPlayerMove] will be set to -1 (indicating the space has been "played"); OutPlayerSum will be incremented by OutBoard[InPlayerMove]

*/

```
void MakeMove (int InPlayerMove, int OutBoard[], int& OutPlayerSum)
{
    OutPlayerSum += OutBoard[InPlayerMove]; //increase score by amount in board space
    OutBoard[InPlayerMove] = -1; //set chosen board space to empty
}
```