Kyle Vecere

**1/21/17 - Input and Storage Ideas**

The first part of any programming project that should always be dealt with is what the intended input is supposed to be, as well as how to handle it. Since this project asks that the input be "the contents of the 9 cells across a row for each of the 9 rows. Use a 0 for an empty cell.", I decided that some form of cin would be best since it would require some form of user interaction. This is where I ran into a few issues:

1. Initially I tried storing the cin into an int for each row (ie cin within a for loop) and then push that to a vector. The problem that I discovered during testing is if you try to input a number with a leading zero (ie. 004359621), the compiler treats this number as an octal (a number with base 8), which will store it incorrectly. Moreover, if that number had an 8 or 9 in it, the program will crash since 8 and 9 cannot exist in a base 8 number system.

2. I then decided to try and use cin.getline, since it would store each row as a c string and would therefore make the whole board-making process less time consuming. However, if the user puts in too many characters than the c string could hold (ie entering 12 chars into a 9 char array), it would cause a memory leak. Seeing how there was no way to prevent this sort of thing, I abandoned this idea as well.

This is where I decided that the best way to use cin was to store it as a string, check the length of the string to see if it was exactly 9, break each string down char by char, and then store it into a 2-dimensional vector for chars. In addition, this cin and string check would be inside a for loop so that it process it all row by row.

**1/25/17- Making the Board and Early Testing**

I decided to take the board from the example input shown in the sudoku handout and use that for testing, since the expected output is also shown in the handout. Also, I decided to leave the cin portion of the code out for now, since it’s something that can easily be done later and using cin would only slow the testing process down.

Started to make a void function called boardMaker that would break up the vector of strings and use push\_back into a global 2-dimensional vector of chars. I did this by having 2 nested for loops (one to iterate over each row, one to iterate over each character). In the outer for loop, I had the function keep creating a vector of characters called rowInfo which was then pushed back into the global 2-dimensional vector known as board. In the inner for loop, each character was pushed back into rowInfo.

Here are the tests conducted on boardMaker:

**Test 1: Normal Input**

Input: vector<string>a={"391500206","285090000","674010580","020400037","000906000","140002090","017020008","000060700","803007000"};

Output:

3 9 1 5 0 0 2 0 6 2 8 5 0 9 0 0 0 0 6 7 4 0 1 0 5 8 0 0 2 0 4 0 0 0 3 7 0 0 0 9 0 6 0 0 0 1 4 0 0 0 2 0 9 0 0 1 7 0 2 0 0 0 8 0 0 0 0 6 0 7 0 0 8 0 3 0 0 7 0 0 0

**Test 2:** **Incorrect Amount of Elements In Input Vector**

Input:

vector<string>a={“123456789”}

Output:

Error: Incorrect number of elements in vector

Since boardMaker function goes through the entire sudoku board in one swoop, I plan on using this for checking to see if there are duplicates in the same rows, columns, and 3x3 grids.

This is when I created the functions isDupCrossing (int row, int col) and isDupInGrid (int gridNum), and getGridNum (int row, int col). isDupCrossing works by passing the coordinates of the cell (aka the row and column number), then iterate through the row and column that it’s in to see if the 2 values being compared are identical, aren’t 0 (blank), and don't occupy the same cell. isDupInGrid is supposed to do the same thing but for grids, however I’m not sure how to implement it. Lastly, if isDupCrossing returns false, an error is printed to the console and the boardMaker function returns nothing.

Here are the tests conducted on isDupCrossing:

**Test 1: Normal Input**

Indirect Input (required for this function):

vector<string>a={"391500206","285090000","674010580","020400037","000906000","140002090","017020008","000060700","803007000"};

Direct Input:

4,4 (checks cell in row 4, column 4)

Output:

compared 0 in row 4, column 0. No true duplicate  
compared 0 in row 4, column 1. No true duplicate  
compared 0 in row 4, column 2. No true duplicate  
compared 9 in row 4, column 3. No true duplicate  
compared 0 in row 4, column 4. No true duplicate  
compared 6 in row 4, column 5. No true duplicate  
compared 0 in row 4, column 6. No true duplicate  
compared 0 in row 4, column 7. No true duplicate  
compared 0 in row 4, column 8. No true duplicate  
compared 0 in row 0, column 4. No true duplicate  
compared 9 in row 1, column 4. No true duplicate  
compared 1 in row 2, column 4. No true duplicate  
compared 0 in row 3, column 4. No true duplicate  
compared 0 in row 4, column 4. No true duplicate  
compared 0 in row 5, column 4. No true duplicate  
compared 2 in row 6, column 4. No true duplicate  
compared 6 in row 7, column 4. No true duplicate  
0

Here’s what the code currently looks like:

#include <iostream>

#include <vector>

#include <string>

using namespace std;

vector<vector<char>> board;//outer vector is rows, inner is each cell for the rows

int getGridNum(int row,int col){//note: grid numbers are from 0 to 8.

int bigRow=row/3;

int bigCol=col/3;

return bigCol+bigRow\*3;

}

bool isDupInGrid(int gridNum){}

bool isDupCrossing(int row,int col){

char number= board[row][col];

//checks if the 2 numbers are identical, arent 0 (blank), and don't occupy the same cell.

for(int r=0;r<board[row].size();r++){

if(board[row][r]==board[row][col] && board[row][r]!='0' && r!=col){return true;}

cout<<"compared "<<board[row][r]<<" in row "<<row<<", column "<<r<<". No true duplicate"<<endl;

}

for(int c=0;c<board.size()-1;c++){

//cout<<"dolla"<<endl;

if(board[c][col]==board[row][col] &&board[c][col]!='0' && c!=row){return true;}

cout<<"compared "<<board[c][col]<<" in row "<<c<<", column "<<col<<". No true duplicate"<<endl;

}

return false;

}

void boardMaker(vector<string> input){

if(input.size()!=9){

cerr<<"Error: Incorrect number of elements in vector";

return;

}

for(int row=0;row<9;row++){//for each row

vector<char> newRow;

board.push\_back(newRow);

for(int cell=0;cell<9;cell++){//for each cell

char c=input[row].at(cell);

if(input[row].length()!=9){

cerr<<"Error: Incorrect number of digits for row "<<row;

return;

}

//cout<<c<<"\t";

board[row].push\_back(c);

// if(isDupCrossing(row,cell)){

// cerr<<"Error: Number "<<c<<" in row "<<row<<" is in illgeal cell";

// return;

// }

}

}

//cout<<"WOW"<<endl;

}

int main(){

vector<string>a={"391500206","285090000","674010580","020400037","000906000","140002090","017020008","000060700","803007000"};

boardMaker(a);

for(int row=0;row<9;row++){//for each row

for(int cell=0;cell<9;cell++){//for each cell

//cout<<board[row][cell]<<"\t";

}

}

cout<<isDupCrossing(4,4);

}

**1/30/17 - Improving boardMaker, Interactive Input, and Checking All 3x3 Grids**

Added a check into boardMaker for non-numeric characters. Since all characters carry an assigned ASCII value, I was able to store the real value of the characters by subtracting the character by ‘0’ and storing that in a temporary variable called cToInt. This was then checked to see if the value was between 0 and 9, and if it wasn’t an error was printed to the console and the main function would return 0. In addition, isDupCrossing is now called in the main function instead of boardMaker, only because the main function would still continue regardless of there being an error in the board (boardMaker can’t return anything since it’s a void function). Will definitely change this in the future since the entire board is getting parsed twice only to do two things.

For the interactive input, I simply stored the input for each individual row from the user (gotten from cin) into a string that’s then stored into a vector of strings called fullInput, which is eventually passed into boardMaker. This all takes place in a for loop that iterates 9 times to represent the 9 rows. However, if one of the rows has a length that isn’t 9, then the input isn’t correct, in which case an error is printed to the console and the main function returns 0.

I couldn’t think of a way to check each individual 3x3 grid to see if there were duplicates, so I replaced the empty isDupInGrid function and replaced it with a boolean threeGridChecker (int row) function. This function checks the 3 3x3 grids on a 1 by 1 basis with 3 nested for loops (first one for each 3x3 grid along a 3x3 row, second one for each row in the current 3x3 grid, third one for each column in the current 3x3 grid). In the first loop, a string named fullGrid contains all the possible numbers within a 3x3 grid (ie. “123456789”). In the third loop, the function checks to see if the number in the current cell is nonzero and if it has been seen before in the 3x3 grid. If it has been seen before and the number is not zero, then an error is printed to the console and the threeGridChecker function returns false. Otherwise, the current cell character is removed from fullGrid. This function is executed in boardMaker after the whole board is made.

Here are the tests conducted on threeGridChecker:

**Test 1:** **Normal Input (Note: Function skips over zeros)**

Input:

vector<string>a={"391500206","285090000","674010580","020400037","000906000","140002090","017020008","000060700","803007000"};

Output:

seen num 3 in row 0 col 0, current nums not seen in grid: 12456789  
seen num 9 in row 0 col 1, current nums not seen in grid: 1245678  
seen num 1 in row 0 col 2, current nums not seen in grid: 245678  
seen num 2 in row 1 col 0, current nums not seen in grid: 45678  
seen num 8 in row 1 col 1, current nums not seen in grid: 4567  
seen num 5 in row 1 col 2, current nums not seen in grid: 467  
seen num 6 in row 2 col 0, current nums not seen in grid: 47  
seen num 7 in row 2 col 1, current nums not seen in grid: 4  
seen num 4 in row 2 col 2, current nums not seen in grid:   
seen num 5 in row 0 col 3, current nums not seen in grid: 12346789  
seen num 9 in row 1 col 4, current nums not seen in grid: 1234678  
seen num 1 in row 2 col 4, current nums not seen in grid: 234678  
seen num 2 in row 0 col 6, current nums not seen in grid: 13456789  
seen num 6 in row 0 col 8, current nums not seen in grid: 1345789  
seen num 5 in row 2 col 6, current nums not seen in grid: 134789  
seen num 8 in row 2 col 7, current nums not seen in grid: 13479  
seen num 2 in row 3 col 1, current nums not seen in grid: 13456789  
seen num 1 in row 5 col 0, current nums not seen in grid: 3456789  
seen num 4 in row 5 col 1, current nums not seen in grid: 356789  
seen num 4 in row 3 col 3, current nums not seen in grid: 12356789  
seen num 9 in row 4 col 3, current nums not seen in grid: 1235678  
seen num 6 in row 4 col 5, current nums not seen in grid: 123578  
seen num 2 in row 5 col 5, current nums not seen in grid: 13578  
seen num 3 in row 3 col 7, current nums not seen in grid: 12456789  
seen num 7 in row 3 col 8, current nums not seen in grid: 1245689  
seen num 9 in row 5 col 7, current nums not seen in grid: 124568  
seen num 1 in row 6 col 1, current nums not seen in grid: 23456789  
seen num 7 in row 6 col 2, current nums not seen in grid: 2345689  
seen num 8 in row 8 col 0, current nums not seen in grid: 234569  
seen num 3 in row 8 col 2, current nums not seen in grid: 24569  
seen num 2 in row 6 col 4, current nums not seen in grid: 13456789  
seen num 6 in row 7 col 4, current nums not seen in grid: 1345789  
seen num 7 in row 8 col 5, current nums not seen in grid: 134589  
seen num 8 in row 6 col 8, current nums not seen in grid: 12345679  
seen num 7 in row 7 col 6, current nums not seen in grid: 1234569

Here’s what the code currently looks like:

#include <iostream>

#include <vector>

#include <string>

using namespace std;

vector<vector<char>> board;//outer vector is rows, inner is each cell for the rows

bool threeGridChecker(int row){//the first row for the 3 3x3 grids

int gridRowStart=row;

for(int grid=0;grid<3;grid++){

int gridColStart=(grid\*3);

string fullGrid="123456789";

for(int r=gridRowStart;r<gridRowStart+3;r++){//for each row

for(int c=gridColStart;c<gridColStart+3;c++){//for each index along row(aka each column)

if(board[r][c]!='0'){

int posOfNumInStr=fullGrid.find(board[r][c]);//gets position of char in string, -1 otherwise

if(posOfNumInStr==-1){//is a duplicate and isn't empty cell

cerr<<"Error: "<<board[r][c]<<" is a duplicate in row "<<r<<" and col "<<c<<endl;

return false;

}

else{

fullGrid.erase(posOfNumInStr,1);

cout<<"seen num "<<board[r][c]<<" in row "<<r<<" col "<<c<<", current nums not seen in grid: "<<fullGrid<<endl;

}

}

}

}

}

return true;

}

bool isDupCrossing(int row,int col){

char number= board[row][col];

//checks if the 2 numbers are identical, aren’t 0 (blank), and don't occupy the same cell.

for(int r=0;r<board[row].size();r++){

if(board[row][r]==board[row][col] && board[row][r]!='0' && r!=col){return true;}

//cout<<"compared "<<board[row][r]<<" in row "<<row<<", column "<<r<<". No true duplicate"<<endl;

}

for(int c=0;c<board.size()-1;c++){

if(board[c][col]==board[row][col] &&board[c][col]!='0' && c!=row){return true;}

//cout<<"compared "<<board[c][col]<<" in row "<<c<<", column "<<col<<". No true duplicate"<<endl;

}

return false;

}

void boardMaker(vector<string> input){

for(int row=0;row<9;row++){//for each row

vector<char> newRow;

board.push\_back(newRow);

for(int cell=0;cell<9;cell++){//for each cell

char c=input[row].at(cell);

int cToNum=c-'0';//makes numbers in character form equal to its int form (ie '0'= 0,'2'=2, etc.)

if(cToNum>9||cToNum<0){

cerr<<"Error: Character "<<c<<" in row "<<row<<" does not represent a known digit";

}

board[row].push\_back(c);

if(isDupCrossing(row,cell)){

cerr<<"Error: Number "<<c<<" in row "<<row<<" is in illgeal cell"<<endl;

return;

}

}

}

for(int bigRow=0;bigRow<9;bigRow+=3){//checks the grids

threeGridChecker(bigRow);

}

}

void boardFiller(vector<string> board){}

int main(){ vector<string>a={"391500206","285090000","674010580","020400037","000906000","140002090","017020008","000060700","803007000"};

boardMaker(a);

// vector<string> fullInput;

// cout<<"Welcome to...\nKyle Vecere's Grandiose Sudoku Puzzle Helper!"<<endl;

// string row;

// for(int i=0;i<9;i++){

// cout<<"Enter the values for row "<<i+1<<endl;

// cin>>row;

// if(row.length()!=9){cerr<<"Error: Row 1 has an incorrect number of digits"<<endl;return 0;}

// fullInput.push\_back(row);

// }

// boardMaker(fullInput);

}

**2/8/17 - Improving Efficiency and Solving the Actual Problem**

Now that the board information can easily be interpreted and read, I realized that the problem still needed to be solved, and that it couldn’t be done with the efficiency nor tidiness that I wanted (ie. calling several other functions), so I thought that the best way to do this was by taking the underlying concepts of isDupCrossing and threeGridChecker and put them into a new function called getPossibleCombos (int row, int col). In the function a string named possibilities contains all the possible numbers in a given cell (ie. “123456789”) is changed using a series of for loops. In these for loops, the function checks for two things. If a number is found along a row/column/3x3 grid that isn’t a duplicate and the cell is empty, then the same character in possibilities is replaced with a dash . However, if an identical number is found along the row/column/3x3 grid and the cell isn’t empty, then getPossibleCombos returns “!”, signifying that there’s an error in the board. If the function goes through all the for loops with no faults, then it checks to see if the cell in question is empty, and if it isn’t then every dash within possibilities will be replaced with a space. This function will be placed inside a nested for loop inside the main function, and will be executed after boardMaker. In addition, all the results of getPossibleCombos will be pushed to a 2 dimensional vector of strings called potentialAnswers.

Finally, isDupCrossing, getGridNum, threeGridChecker, and isDupCrossing will soon be deleted, as they now serve no purpose other than for reference.

Here’s the testing conducted on getPossibleCombos

**Test 1:** **Normal Input (Scaled down to just check 1 cell)**

Input: 0,6

Output:

Parsing row...  
seen num 3 in row 0 col 0, current nums not seen: 123456789  
seen num 9 in row 0 col 1, current nums not seen: 123456789  
seen num 1 in row 0 col 2, current nums not seen: 123456789  
seen num 5 in row 0 col 3, current nums not seen: 123456789  
seen num 0 in row 0 col 4, current nums not seen: 123456789  
seen num 0 in row 0 col 5, current nums not seen: 123456789  
seen num 2 in row 0 col 6, current nums not seen: 123456789  
seen num 0 in row 0 col 7, current nums not seen: 123456789  
seen num 6 in row 0 col 8, current nums not seen: 123456789  
Parsing column...  
seen num 2 in row 0 col 6, current nums not seen: 123456789  
seen num 0 in row 1 col 6, current nums not seen: 123456789  
seen num 5 in row 2 col 6, current nums not seen: 123456789  
seen num 0 in row 3 col 6, current nums not seen: 123456789  
seen num 0 in row 4 col 6, current nums not seen: 123456789  
seen num 0 in row 5 col 6, current nums not seen: 123456789  
seen num 0 in row 6 col 6, current nums not seen: 123456789  
seen num 7 in row 7 col 6, current nums not seen: 123456789  
seen num 0 in row 8 col 6, current nums not seen: 123456789  
Parsing 3x3 grid...  
seen num 2 in row 0 col 6, current nums not seen: 123456789  
seen num 0 in row 0 col 7, current nums not seen: 123456789  
seen num 6 in row 0 col 8, current nums not seen: 123456789  
seen num 0 in row 1 col 6, current nums not seen: 123456789  
seen num 0 in row 1 col 7, current nums not seen: 123456789  
seen num 0 in row 1 col 8, current nums not seen: 123456789  
seen num 5 in row 2 col 6, current nums not seen: 123456789  
seen num 8 in row 2 col 7, current nums not seen: 123456789  
seen num 0 in row 2 col 8, current nums not seen: 123456789  
 2

Here’s what the code currently looks like:

#include <iostream>

#include <vector>

#include <string>

using namespace std;

vector<vector<char>> board;//outer vector is rows, inner is each cell for the rows

vector<vector<string>>potentialAnswers;//^^

int getGridNum(int row,int col){

int bigRow=row/3;

int bigCol=col/3;

return bigCol+bigRow\*3;

}

//below function to become deprecated

bool threeGridChecker(int row){//the first row for the 3 3x3 grids

//checks the 3 3x3 grids on a 1 by 1 basis, since trying to implement this function

//just for checking 1 3x3 grid is too much of a hassle.

int gridRowStart=row;

for(int grid=0;grid<3;grid++){

int gridColStart=(grid\*3);

string fullGrid="123456789";

for(int r=gridRowStart;r<gridRowStart+3;r++){//for each row

for(int c=gridColStart;c<gridColStart+3;c++){//for each index along row(aka each column)

if(board[r][c]!='0'){

int posOfNumInStr=fullGrid.find(board[r][c]);//gets position of char in string, -1 otherwise

if(posOfNumInStr==-1){//is a duplicate and isn't empty cell

cerr<<"Error: "<<board[r][c]<<" is a duplicate in row "<<r<<" and col "<<c<<endl;

return false;

}

//cout<<"row "<<r<<"\tcol "<<c<<"\t"<<board[r][c]<<endl;

else{

fullGrid.erase(posOfNumInStr,0);

}

}

}

}

}

return true;

}

bool isDupCrossing(int row,int col){

char number= board[row][col];

//checks if the 2 numbers are identical, arent 0 (blank), and don't occupy the same cell.

for(int r=0;r<board[row].size();r++){

if(board[row][r]==number && board[row][r]!='0' && r!=col){return true;}

}

for(int c=0;c<board.size();c++){

if(board[c][col]==number && board[c][col]!='0' && c!=row){return true;}

}

return false;

}

string getPossibleCombos(int row, int col){

//returns "!" if error with board is found, actual results otherwise

string possibilities="123456789";

char cellVal=board[row][col];

cout<<"Parsing row..."<<endl;

for(int r=0;r<9;r++){//go through the row

int posIndex=possibilities.find(board[row][r]);//holds index in string possibilities of found char, -1 otherwise

//cout<<posIndex<<endl;

if(posIndex>=0 && r!=col && cellVal=='0'){//if a number is found along row that hasn't been seen

possibilities.replace(posIndex,1,"-");

}

if(posIndex==-1 && r!=col && board[row][r]==cellVal && cellVal!='0'){

//number not found in string, nonzero identical number found along row (error in board)

return "!";

}

cout<<"seen num "<<board[row][r]<<" in row "<<row<<" col "<<r<<", current nums not seen: "<<possibilities<<endl;

}

//cout<<possibilities<<endl;

cout<<"Parsing column..."<<endl;

for(int c=0;c<9;c++){//go through the column

int posIndex=possibilities.find(board[c][col]);

if(posIndex>=0 && c!=row && cellVal=='0'){//if a number is found along column that hasn't been seen

possibilities.replace(posIndex,1,"-");

}

if(posIndex==-1 && c!=row && board[c][col]==cellVal && cellVal!='0'){

//number not found in string, nonzero identical number found along col (error in board)

return "!";

}

cout<<"seen num "<<board[c][col]<<" in row "<<c<<" col "<<col<<", current nums not seen: "<<possibilities<<endl;

}

//cout<<possibilities<<endl;

//TIME FOR THE GRID

cout<<"Parsing 3x3 grid..."<<endl;

for(int smallRow=(row/3)\*3;smallRow<((row/3)\*3)+3;smallRow++){

for(int smallCol=(col/3)\*3;smallCol<((col/3)\*3)+3;smallCol++){

int posIndex=possibilities.find(board[smallRow][smallCol]);

if(posIndex>=0 && board[smallRow][smallCol]!=cellVal && cellVal=='0'){

//if a number is found in 3x3 grid that hasn't been seen

possibilities.replace(posIndex,1,"-");

}

if(posIndex>=0 && board[smallRow][smallCol]==cellVal && cellVal!='0' && (smallCol!=col || smallRow!=row)){

//duplicate of cellVal found in 3x3 grid (error in board)

return "!";

}

cout<<"seen num "<<board[smallRow][smallCol]<<" in row "<<smallRow<<" col "<<smallCol<<", current nums not seen: "<<possibilities<<endl;

}

}

//cout<<possibilities<<endl;

if(cellVal!='0'){

for(int i=0;i<9;i++){

if(possibilities.at(i)!=cellVal){possibilities.replace(i,1," ");}

}

}

return possibilities;

}

void boardMaker(vector<string> input){

for(int row=0;row<9;row++){//for each row

vector<char> newRow;

board.push\_back(newRow);

for(int cell=0;cell<9;cell++){//for each cell

char c=input[row].at(cell);

int cToNum=c-'0';//makes numbers in character form equal to its int form (ie '0'= 0,'2'=2, etc.)

if(cToNum>9||cToNum<0){

cerr<<"Error: Character "<<c<<" in row "<<row<<" does not represent a known digit";

}

board[row].push\_back(c);

}

}

for(int bigRow=0;bigRow<9;bigRow+=3){//checks the grids

threeGridChecker(bigRow);

}

}

void boardFiller(vector<string> board){}

int main(){ vector<string>a={"391500206","285090000","674010580","020400037","000906000","140002090","017020008","000060700","803007000"};

boardMaker(a);

cout<<getPossibleCombos(0,6);

for(int row=0;row<9;row++){//calculates the grid number

vector<string> rowCombos;

for(int col=0;col<9;col++){

//string combo= getPossibleCombos(row,col);

//rowCombos.push\_back(combo);

}

//potentialAnswers.push\_back(rowCombos);

}

// vector<string> fullInput;

// cout<<"Welcome to...\nKyle Vecere's Grandiose Sudoku Puzzle Helper!"<<endl;

// string row;

// for(int i=0;i<9;i++){

// cout<<"Enter the values for row "<<i+1<<endl;

// cin>>row;

// if(row.length()!=9){cerr<<"Error: Row 1 has an incorrect number of digits"<<endl;return 0;}

// fullInput.push\_back(row);

// }

// boardMaker(fullInput);

}

**2/17/17 - Printing the Potential Answers**

Now that the program will provide the correct potential answers for each cell, it still has to show them in a displayable manner. I referred back to the outline for the project, and noticed that for the sample output, each row and column had 3 smaller rows and columns within them that represented the potential combinations. This is where I came up with the method for printing out the results in the newly declared boardPrinter (vector<vector<string>> printMe) function. In this function are 3 for loops nested into one for loop. The first for loop iterates over each row, the second over each smaller row within a cell, the third over each cell, and the fourth over each smaller cell in a cell. The fourth for loop handles the spacing between characters, where each character gets separated by a space unless the character in question is at the end of the board, in which case an empty string is printed. The third for loop handles the vertical borders of the board, where each cell has a “| “ border and each 3x3 grid has a “|| “ border. This is the case unless it’s at the end of the board, in which case endl is printed. The second loop simply ensures that the previous actions are done 3 times per row. Finally, the first loop handles the horizontal borders of the board accordingly. Note that this function is only printing to the console at this time for easier testing.

Here’s the testing conducted for boardPrinter

**Test 1: Normal Input**

Input: all of the results from getPossibleCombos, where the output from that is based off of this vector:

vector<string>a={"391500206","285090000","674010580","020400037","000906000","140002090","017020008","000060700","803007000"};

Output:

3 | | 1 || | - - - | - - - || 2 | - - - |   
 | | || 5 | 4 - - | 4 - - || | 4 - - | 6  
 | 9 | || | 7 8 - | - 8 - || | 7 - - |   
------+-------+-------||-------+-------+-------||-------+-------+------  
 2 | | || - - 3 | | - - 3 || 1 - 3 | 1 - - | 1 - 3  
 | | 5 || - - 6 | | 4 - - || 4 - - | 4 - - | 4 - -  
 | 8 | || 7 - - | 9 | - - - || - - - | 7 - - | - - -  
------+-------+-------||-------+-------+-------||-------+-------+------  
 | | || - 2 3 | 1 | - - 3 || | | - - 3  
 6 | | 4 || - - - | | - - - || 5 | | - - -  
 | 7 | || - - - | | - - - || | 8 | - - 9  
======================++=======================++======================  
- - - | 2 | - - - || | - - - | 1 - - || 1 - - | 3 |   
- 5 - | | - - 6 || 4 | - 5 - | - 5 - || - - 6 | |   
- - 9 | | - 8 9 || | - 8 - | - 8 - || - 8 - | | 7   
------+-------+-------||-------+-------+-------||-------+-------+------  
- - - | - - 3 | - - - || | - - 3 | || 1 - - | 1 2 - | 1 2 -  
- 5 - | - 5 - | - - - || | - 5 - | 6 || 4 - - | 4 5 - | 4 5 -  
7 - - | - - - | - 8 - || 9 | 7 8 - | || - 8 - | - - - | - - -  
------+-------+-------||-------+-------+-------||-------+-------+------  
1 | | - - - || - - 3 | - - 3 | 2 || - - - | | - - -  
 | 4 | - - 6 || - - - | - 5 - | || - - 6 | | - 5 -  
 | | - 8 - || 7 8 - | 7 8 - | || - 8 - | 9 | - - -  
======================++=======================++======================  
- - - | 1 | || - - 3 | 2 | - - 3 || - - 3 | - - - |   
4 5 - | | || - - - | | 4 5 - || 4 - 6 | 4 5 6 |   
- - 9 | | 7 || - - - | | - - 9 || - - 9 | - - - | 8   
------+-------+-------||-------+-------+-------||-------+-------+------  
- - - | - - - | - 2 - || 1 - 3 | | 1 - 3 || | 1 2 - | 1 2 3  
4 5 - | - 5 - | - - - || - - - | 6 | 4 5 - || | 4 5 - | 4 5 -  
- - 9 | - - - | - - 9 || - 8 - | | - 8 9 || 7 | - - - | - - 9  
------+-------+-------||-------+-------+-------||-------+-------+------  
 | - - - | 3 || 1 - - | - - - | || 1 - - | 1 2 - | 1 2 -  
 | - 5 6 | || - - - | 4 5 - | || 4 - 6 | 4 5 6 | 4 5 -  
 8 | - - - | || - - - | - - - | 7 || - - 9 | - - - | - - 9

**Test 2: Empty Input**

Input: a 2 dimensional vector of string vectors, which has 81 strings of “000000000”

Output:

1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3  
4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6  
7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9  
------+-------+-------||-------+-------+-------||-------+-------+------  
1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3  
4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6  
7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9  
------+-------+-------||-------+-------+-------||-------+-------+------  
1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3  
4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6  
7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9  
======================++=======================++======================  
1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3  
4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6  
7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9  
------+-------+-------||-------+-------+-------||-------+-------+------  
1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3  
4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6  
7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9  
------+-------+-------||-------+-------+-------||-------+-------+------  
1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3  
4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6  
7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9  
======================++=======================++======================  
1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3  
4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6  
7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9  
------+-------+-------||-------+-------+-------||-------+-------+------  
1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3  
4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6  
7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9  
------+-------+-------||-------+-------+-------||-------+-------+------  
1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3 || 1 2 3 | 1 2 3 | 1 2 3  
4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6 || 4 5 6 | 4 5 6 | 4 5 6  
7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9 || 7 8 9 | 7 8 9 | 7 8 9

**Test 3: Full Input**

Input: a 2 dimensional vector of string vectors where every string contains no zeros (ie. “123456879”, “456789123”, “789123456”, etc.)

Output:

1 | 2 | 3 || | | || | |   
 | | || 4 | 5 | 6 || | |   
 | | || | | || 7 | 8 | 9  
------+-------+-------||-------+-------+-------||-------+-------+------  
 | | || | | || 1 | 2 | 3  
4 | 5 | 6 || | | || | |   
 | | || 7 | 8 | 9 || | |   
------+-------+-------||-------+-------+-------||-------+-------+------  
 | | || 1 | 2 | 3 || | |   
 | | || | | || 4 | 5 | 6  
7 | 8 | 9 || | | || | |   
======================++=======================++======================  
 2 | 3 | || | | || | | 1   
 | | 4 || 5 | 6 | || | |   
 | | || | | 7 || 8 | 9 |   
------+-------+-------||-------+-------+-------||-------+-------+------  
 | | || | | 1 || 2 | 3 |   
 5 | 6 | || | | || | | 4   
 | | 7 || 8 | 9 | || | |   
------+-------+-------||-------+-------+-------||-------+-------+------  
 | | 1 || 2 | 3 | || | |   
 | | || | | 4 || 5 | 6 |   
 8 | 9 | || | | || | | 7   
======================++=======================++======================  
 3 | | || | | || | 1 | 2   
 | 4 | 5 || 6 | | || | |   
 | | || | 7 | 8 || 9 | |   
------+-------+-------||-------+-------+-------||-------+-------+------  
 | | || | 1 | 2 || 3 | |   
 6 | | || | | || | 4 | 5   
 | 7 | 8 || 9 | | || | |   
------+-------+-------||-------+-------+-------||-------+-------+------  
 | 1 | 2 || 3 | | || | |   
 | | || | 4 | 5 || 6 | |   
 9 | | || | | || | 7 | 8

Here’s what the code looks like so far:

#include <iostream>

#include <vector>

#include <string>

using namespace std;

vector<vector<char>> board;//outer vector is rows, inner is each cell for the rows

vector<vector<string>>potentialAnswers;//^^

string getPossibleCombos(int row, int col){

//returns "!" if error with board is found, actual results otherwise

string possibilities="123456789";

char cellVal=board[row][col];

for(int r=0;r<9;r++){//go through the row

int posIndex=possibilities.find(board[row][r]);//holds index in string possibilities of found char, -1 otherwise

//cout<<posIndex<<endl;

if(posIndex>=0 && r!=col && cellVal=='0'){//if a number is found along row that hasn't been seen

possibilities.replace(posIndex,1,"-");

}

if(posIndex==-1 && r!=col && board[row][r]==cellVal && cellVal!='0'){

//number not found in string, nonzero identical number found along row (error in board)

return "!";

}

}

//cout<<possibilities<<endl;

for(int c=0;c<9;c++){//go through the column

int posIndex=possibilities.find(board[c][col]);

if(posIndex>=0 && c!=row && cellVal=='0'){//if a number is found along column that hasn't been seen

possibilities.replace(posIndex,1,"-");

}

if(posIndex==-1 && c!=row && board[c][col]==cellVal && cellVal!='0'){

//number not found in string, nonzero identical number found along col (error in board)

return "!";

}

}

//cout<<possibilities<<endl;

//TIME FOR THE GRID

for(int smallRow=(row/3)\*3;smallRow<((row/3)\*3)+3;smallRow++){

for(int smallCol=(col/3)\*3;smallCol<((col/3)\*3)+3;smallCol++){

//cout<<row<<col<<"= "<<cellVal<<"\t"<<smallRow<<smallCol<<"= "<<board[smallRow][smallCol]<<endl;

int posIndex=possibilities.find(board[smallRow][smallCol]);

if(posIndex>=0 && board[smallRow][smallCol]!=cellVal && cellVal=='0'){

//if a number is found in 3x3 grid that hasn't been seen

possibilities.replace(posIndex,1,"-");

}

if(posIndex>=0 && board[smallRow][smallCol]==cellVal && cellVal!='0' && (smallCol!=col || smallRow!=row)){

//duplicate of cellVal found in 3x3 grid (error in board)

return "!";

}

}

}

//cout<<possibilities<<endl;

if(cellVal!='0'){

for(int i=0;i<9;i++){

if(possibilities.at(i)!=cellVal){possibilities.replace(i,1," ");}

}

}

return possibilities;

}

void boardMaker(vector<string> input){

for(int row=0;row<9;row++){//for each row

vector<char> newRow;

for(int cell=0;cell<9;cell++){//for each cell

char c=input[row].at(cell);

int cToNum=c-'0';//makes numbers in character form equal to its int form (ie '0'= 0,'2'=2, etc.)

if(cToNum>9||cToNum<0){

cerr<<"Error: Character "<<c<<" in row "<<row<<" does not represent a known digit";

}

newRow.push\_back(c);

}

board.push\_back(newRow);

}

cout<<" WOOOOOOOOOOOWWWWWWWWWWWWWWWWWWW"<<endl;

}

void boardPrinter(vector<vector<string>> printMe){

for(int row=0;row<9;row++){

for(int rowInCell=0;rowInCell<3;rowInCell++){

for(int cell=0;cell<9;cell++){

for(int cellInCell=rowInCell\*3;cellInCell<(rowInCell\*3)+3;cellInCell++){

cout<<printMe[row][cell].at(cellInCell);

if(cell==8 && (cellInCell+1)%3==0){

cout<<"";

}

else{

cout<<" ";

}

}

if(cell!=8 && (cell+1)%3==0){cout<<"|| ";}//border of 3x3 grid

if(cell!=8 && (cell+1)%3!=0){cout<<"| ";}//border of cell/column

if(cell==8){cout<<endl;}

}

}

if((row+1)%3==0 && row!=8){//border of 3x3 grid

cout<<"======================++=======================++======================"<<endl;

}

if((row+1)%3!=0 && row!=8){//border of row

cout<<"------+-------+-------||-------+-------+-------||-------+-------+------"<<endl;

}

}

}

int main(){

//vector<string>a={"391500206","285090000","674010580","020400037","000906000","140002090","017020008","000060700","803007000"};

//boardMaker(a);

vector<string> fullInput;

cout<<"Welcome to...\nKyle Vecere's Grandiose Sudoku Puzzle Helper (Supreme C++ Edition)!"<<endl;

string row;

for(int i=0;i<9;i++){

cout<<"Enter the values for row "<<i<<endl;

cin>>row;

if(row.length()!=9){cerr<<"Error: Row "<<i<<" has an incorrect number of digits"<<endl;return 0;}

fullInput.push\_back(row);

}

cout<<"wow"<<endl;

boardMaker(fullInput);

cout<<"blank"<<endl;

for(int r=0;r<9;r++){

vector<string> rowCombos;

for(int c=0;c<9;c++){

rowCombos.push\_back(getPossibleCombos(r,c));

}

potentialAnswers.push\_back(rowCombos);

}

cout<<"Lastly, please enter the filename you want"<<endl;

string filename;

cin>>filename;

boardPrinter(potentialAnswers);

return 1;

}

**3/1/16 - Valid Filenames**

One of the last things the project requires is the to let the user name the file they’re outputting towards. In order to get the user input, I created a string called filename, used string::getline(cin, filename) as a means of storing something as a string that includes whitespaces, then passed filename to a new function called isFilenameValid (string name). I referred to Microsoft’s documentation regarding filenames and paths (<https://msdn.microsoft.com/en-us/library/aa365247>), and applied the specific details regarding naming conventions toward isFilenameValid. In addition, the input method for getting the filename is different from the one for the board in that it doesn’t ignore whitespaces.

Here’s the code so far:

#include <fstream>

#include <iostream>

#include <vector>

#include <string>

using namespace std;

vector<vector<char>> board;//outer vector is rows, inner is each cell for the rows

vector<vector<string>>potentialAnswers;//^^

string getPossibleCombos(int row, int col){

//returns "!" if error with board is found, actual results otherwise

string possibilities="123456789";

char cellVal=board[row][col];

for(int r=0;r<9;r++){//go through the row

int posIndex=possibilities.find(board[row][r]);//holds index in string possibilities of found char, -1 otherwise

//cout<<posIndex<<endl;

if(posIndex>=0 && r!=col && cellVal=='0'){//if a number is found along row that hasn't been seen

possibilities.replace(posIndex,1,"-");

}

if(posIndex==-1 && r!=col && board[row][r]==cellVal && cellVal!='0'){

//number not found in string, nonzero identical number found along row (error in board)

return "!";

}

}

//cout<<possibilities<<endl;

for(int c=0;c<9;c++){//go through the column

int posIndex=possibilities.find(board[c][col]);

if(posIndex>=0 && c!=row && cellVal=='0'){//if a number is found along column that hasn't been seen

possibilities.replace(posIndex,1,"-");

}

if(posIndex==-1 && c!=row && board[c][col]==cellVal && cellVal!='0'){

//number not found in string, nonzero identical number found along col (error in board)

return "!";

}

}

//cout<<possibilities<<endl;

//TIME FOR THE GRID

for(int smallRow=(row/3)\*3;smallRow<((row/3)\*3)+3;smallRow++){

for(int smallCol=(col/3)\*3;smallCol<((col/3)\*3)+3;smallCol++){

//cout<<row<<col<<"= "<<cellVal<<"\t"<<smallRow<<smallCol<<"= "<<board[smallRow][smallCol]<<endl;

int posIndex=possibilities.find(board[smallRow][smallCol]);

if(posIndex>=0 && board[smallRow][smallCol]!=cellVal && cellVal=='0'){

//if a number is found in 3x3 grid that hasn't been seen

possibilities.replace(posIndex,1,"-");

}

if(posIndex>=0 && board[smallRow][smallCol]==cellVal && cellVal!='0' && (smallCol!=col || smallRow!=row)){

//duplicate of cellVal found in 3x3 grid (error in board)

return "!";

}

}

}

//cout<<possibilities<<endl;

if(cellVal!='0'){

for(int i=0;i<9;i++){

if(possibilities.at(i)!=cellVal){possibilities.replace(i,1," ");}

}

}

return possibilities;

}

void boardMaker(vector<string> input){

for(int row=0;row<9;row++){//for each row

vector<char> newRow;

for(int cell=0;cell<9;cell++){//for each cell

char c=input[row].at(cell);

int cToNum=c-'0';//makes numbers in character form equal to its int form (ie '0'= 0,'2'=2, etc.)

if(cToNum>9||cToNum<0){

cout<<"Error: Character "<<c<<" in row "<<row<<" does not represent a known digit";return;

}

newRow.push\_back(c);

}

board.push\_back(newRow);

}

cout<<" WOOOOOOOOOOOWWWWWWWWWWWWWWWWWWW"<<endl;

}

bool isFilenameValid(string name){// https://msdn.microsoft.com/en-us/library/aa365247

if(name.length()>256){cout<<"Error: filename is too long"<<endl;}

cout<<name<<endl;

char illegalCharacters[11]={'<', '>', ':', '\"', '\\', '/', '|', '?', '\*', '\0'};

if(name.length()>=1){

for(int i=0;i<11;i++){

if(name.find(illegalCharacters[i])!=-1){

cout<<"Error: filename contains illegal character "<<illegalCharacters[i]<<endl;

return false;

}

}

//Note: spaces and periods cant be at the end of a filename

}

if(name.length()==3||name.length()==4){

string reservedNames[22]={"CON","PRN","AUX","NUL","COM1","COM2","COM3","COM4","COM5","COM6","COM7","COM8","COM9","LPT1","LPT2","LPT3","LPT4","LPT5","LPT6","LPT7","LPT8","LPT9"};

for(int i=0;i<22;i++){

if(name==reservedNames[i]){

cout<<"Error: filename is reserved"<<endl;

return false;

}

}

}

if(name.find\_last\_of(' ')==name.length()-1){

cout<<"Error: space at end of filename"<<endl;

return false;

}

if(name.find\_last\_of('.')==name.length()-1){

cout<<"Error: period at end of filename"<<endl;

return false;

}

return true;

}

void boardPrinter(vector<vector<string>> printMe){

for(int row=0;row<9;row++){

for(int rowInCell=0;rowInCell<3;rowInCell++){

for(int cell=0;cell<9;cell++){

for(int cellInCell=rowInCell\*3;cellInCell<(rowInCell\*3)+3;cellInCell++){

cout<<printMe[row][cell].at(cellInCell);

if(cell==8 && (cellInCell+1)%3==0){

cout<<"";

}

else{

cout<<" ";

}

}

if(cell!=8 && (cell+1)%3==0){cout<<"|| ";}//border of 3x3 grid

if(cell!=8 && (cell+1)%3!=0){cout<<"| ";}//border of cell/column

if(cell==8){cout<<endl;}

}

}

if((row+1)%3==0 && row!=8){//border of 3x3 grid

cout<<"======================++=======================++======================"<<endl;

}

if((row+1)%3!=0 && row!=8){//border of row

cout<<"------+-------+-------||-------+-------+-------||-------+-------+------"<<endl;

}

}

}

int main(){

//vector<string>a={"391500206","285090000","674010580","020400037","000906000","140002090","017020008","000060700","803007000"};

//boardMaker(a);

vector<string> fullInput;

cout<<"Welcome to...\nKyle Vecere's Grandiose Sudoku Puzzle Helper (Supreme C++ Edition)!"<<endl;

string row;

for(int i=0;i<9;i++){

cout<<"Enter the values for row "<<i<<endl;

cin>>row;

if(row.length()!=9){cerr<<"Error: Row "<<i<<" has an incorrect number of digits"<<endl;return 0;}

fullInput.push\_back(row);

}

cout<<"wow"<<endl;

boardMaker(fullInput);

cout<<"blank"<<endl;

for(int r=0;r<9;r++){

vector<string> rowCombos;

for(int c=0;c<9;c++){

string answer=getPossibleCombos(r,c);

if(answer=="!"){cout<<"Error: Duplicate found in row "<<r<<", column "<<c<<endl;return 0;}

rowCombos.push\_back(answer);

}

potentialAnswers.push\_back(rowCombos);

}

cout<<"Lastly, please enter the filename you want"<<endl;

string filename;

cin.ignore();//ignores leading whitespaces. required since getline will see it.

getline(cin,filename);

//cout<<"FILENAME IS:"<<filename<<endl;

isFilenameValid(filename);

boardPrinter(potentialAnswers);

return 1;

}

**3/29/17 - Back To Basics**

Since this project requires that no special modules (such as strings and vectors) be used, nearly all of the code needed to be modified in order to use cstrings. This is where I ran into a few problems:

1. In boardMaker, the ability to determine the size of the input was a challenge. Since using vectors (as well as their properties) was out of the question, it meant that I needed to somehow get the length of the input via other means. I didn’t want it to parse through the c-string to find the length because I wanted to keep everything as simple as possible. After some research I found and used cin.gcount() as a means of obtaining the length. This returns the size of the stream most recently read.

2. Originally in getPossibleCombos, I was returning a full string, which wasn’t a problem since it was being returned by value and it was essentially one item. However, this function now returns a char \*, which is problematic since the entire c string would become garbage upon exiting the function. After doing some research, I found that one of the solutions is to declare a global c string within the function that is of type char \*, then set the values for it accordingly.

3. (Still a problem) For some strange reason, certain cells within row 1 will show up as being blank regardless of what the input is.

I also haven’t added/changed isFilenameValid and boardPrinter yet, as I wanted to get the core aspects of the program working first.

Here’s the code so far:

#include<fstream>

#include<iostream>

using namespace std;

char \* getPossibleCombos(char (\*board)[9][10],int row, int col){

//first char in possibilities is \0 if error with board is found, actual results otherwise

char\* possibilities= new char[10];

possibilities[0]='1';

possibilities[1]='2';

possibilities[2]='3';

possibilities[3]='4';

possibilities[4]='5';

possibilities[5]='6';

possibilities[6]='7';

possibilities[7]='8';

possibilities[8]='9';

possibilities[9]='\0';

char cellVal=\*board[row][col];

//cout<<"cellVal is "<<cellVal<<endl;

for(int r=0;r<9;r++){//go through the row

int posIndex=0;//holds index in string possibilities of found char, 9 otherwise

while(posIndex<9 && possibilities[posIndex]!=\*board[row][r]){

posIndex++;

}

//cout<<posIndex<<endl;

if(posIndex<9 && r!=col && cellVal=='0'){//if a number is found along row that hasn't been seen

possibilities[posIndex]='-';

}

if(posIndex<9 && r!=col && \*board[row][r]==cellVal && cellVal!='0'){

possibilities[0]='!';

return possibilities;

}

}

//cout<<possibilities<<endl;

for(int c=0;c<9;c++){//go through the column

int posIndex=0;

while(posIndex<9 && possibilities[posIndex]!=\*board[c][col]){

posIndex++;

}

if(posIndex<9 && c!=row && cellVal=='0'){//if a number is found along column that hasn't been seen

possibilities[posIndex]='-';

}

if(posIndex<9 && c!=row && \*board[c][col]==cellVal && cellVal!='0'){

possibilities[0]='!';

return possibilities;

}

}

//TIME FOR THE GRID

for(int smallRow=(row/3)\*3;smallRow<((row/3)\*3)+3;smallRow++){

for(int smallCol=(col/3)\*3;smallCol<((col/3)\*3)+3;smallCol++){

//cout<<row<<col<<"= "<<cellVal<<"\t"<<smallRow<<smallCol<<"= "<<board[smallRow][smallCol]<<endl;

int posIndex=0;

while(posIndex<9 && possibilities[posIndex]!=\*board[smallRow][smallCol]){

posIndex++;

}

if(posIndex<9 && \*board[smallRow][smallCol]!=cellVal && cellVal=='0'){

//if a number is found in 3x3 grid that hasn't been seen

possibilities[posIndex]='-';

}

if(posIndex<9 && \*board[smallRow][smallCol]==cellVal && cellVal!='0' && (smallCol!=col || smallRow!=row)){

//duplicate of cellVal found in 3x3 grid (error in board)

possibilities[0]='!';

return possibilities;

}

}

}

if(cellVal!='0'){

for(int i=0;i<9;i++){

if(possibilities[i]!=cellVal){possibilities[i]=' ';}

}

}

return possibilities;

}

void boardMaker(char (\*board)[9][10]){

int currentRow=0;

while(currentRow<9){

char row[10];

int cellInRow=0;

cout<<"Enter info for row "<<currentRow<<","<<cin.gcount()<<endl;

cin.getline(row,10);

if(cin.gcount()!=10){//row is incorrect size

cin.clear();

cin.ignore(256,'\n');

cout<<"Error: row "<<currentRow<< " is of incorrect size."<<endl;

currentRow--;

}

while(cellInRow<=9 && cin.gcount()==10){

int charNumToRealNum=row[cellInRow]-'0';

if((charNumToRealNum>9 || charNumToRealNum<0) && charNumToRealNum!=-48){

//checks if its a real number or terminating null char

cin.clear();

cin.ignore(256,'\n');

cout<<"Error: illegal character in row "<<currentRow<<"."<<endl;

currentRow--;

}

\*board[currentRow][cellInRow]=row[cellInRow];

//cout<<row[cellInRow]-'0'<<endl;

cellInRow++;

}

currentRow++;

}

}

int main(){

char board[9][10];//9 rows of 9 character arrays (not including null character)

char potentialAnswers[9][9][10];//9 rows, 9 columns, 9 smaller cells in a cell (not including null character)

// char board[9][10]={

// {'3','9','1','5','0','0','2','0','6','\0'},{'2','8','5','0','9','0','0','0','0','\0'},{'6','7','4','0','1','0','5','8','0','\0'},{'0','2','0','4','0','0','0','3','7','\0'},{'0','0','0','9','0','6','0','0','0','\0'},{'1','4','0','0','0','2','0','9','0','\0'},{'0','1','7','0','2','0','0','0','8','\0'},{'0','0','0','0','6','0','7','0','0','\0'},{'8','0','3','0','0','7','0','0','0','\0'},

// 391500206

// 285090000

// 674010580

// 020400037

// 000906000

// 140002090

// 017020008

// 000060700

// 803007000

// };

boardMaker(&board);

cout<<"moms"<<endl;

for(int r=0;r<9;r++){

for(int c=0;c<9;c++){

char \*cellPossibleNums= getPossibleCombos(&board,r,c);

for(int z=0;z<9;z++){cout<<cellPossibleNums[z];}

cout<<",";

}

cout<<endl;

}

// char \*cellPossibleNums= getPossibleCombos(&board,0,3);

// for(int z=0;z<9;z++){cout<<cellPossibleNums[z];}

cout<<"homs"<<endl;

//cout<<cellPossibleNums<<endl;

cout<<"doms"<<endl;

return 1;

}

**4/7/17 - Addons & Improvements**

After an extensive look at my code and lots of testing, I found/fixed some issues with it. The most notable fix was the strange 2 cell bug that occurred, which was fixed by changing the parameter in getPossibleCombos so that it’s passed via pointer.

Here’s the test to show that it does work:

Input for row 0: 123456789

Input for row 1: 456789123

Input for row 2: 789123456

Input for row 3: 234567891

Input for row 4: 567891234

Input for row 5: 891234567

Input for row 6: 345678912

Input for row 7: 678912345

Input for row 8: 912345678

Output:

after boardmaker  
123456789,000000000,000000000,000000000,000000000,000000000,000000000,000000000,000000000,  
  
starting gpc for 0, 0:   
1 ,  
starting gpc for 0, 1:   
 2 ,  
starting gpc for 0, 2:   
 3 ,  
starting gpc for 0, 3:   
 4 ,  
starting gpc for 0, 4:   
 5 ,  
starting gpc for 0, 5:   
 6 ,  
starting gpc for 0, 6:   
 7 ,  
starting gpc for 0, 7:   
 8 ,  
starting gpc for 0, 8:   
 9,  
  
starting gpc for 1, 0:   
---456789,  
starting gpc for 1, 1:   
---456789,  
starting gpc for 1, 2:   
---456789,  
starting gpc for 1, 3:   
123---789,  
starting gpc for 1, 4:   
123---789,  
starting gpc for 1, 5:   
123---789,  
starting gpc for 1, 6:   
123456---,  
starting gpc for 1, 7:   
123456---,  
starting gpc for 1, 8:   
123456---,  
  
starting gpc for 2, 0:   
---456789,  
starting gpc for 2, 1:   
---456789,  
starting gpc for 2, 2:   
---456789,  
starting gpc for 2, 3:   
123---789,  
starting gpc for 2, 4:   
123---789,  
starting gpc for 2, 5:   
123---789,  
starting gpc for 2, 6:   
123456---,  
starting gpc for 2, 7:   
123456---,  
starting gpc for 2, 8:   
123456---,  
  
starting gpc for 3, 0:   
-23456789,  
starting gpc for 3, 1:   
1-3456789,  
starting gpc for 3, 2:   
12-456789,  
starting gpc for 3, 3:   
123-56789,  
starting gpc for 3, 4:   
1234-6789,  
starting gpc for 3, 5:   
12345-789,  
starting gpc for 3, 6:   
123456-89,  
starting gpc for 3, 7:   
1234567-9,  
starting gpc for 3, 8:   
12345678-,  
  
starting gpc for 4, 0:   
-23456789,  
starting gpc for 4, 1:   
1-3456789,  
starting gpc for 4, 2:   
12-456789,  
starting gpc for 4, 3:   
123-56789,  
starting gpc for 4, 4:   
1234-6789,  
starting gpc for 4, 5:   
12345-789,  
starting gpc for 4, 6:   
123456-89,  
starting gpc for 4, 7:   
1234567-9,  
starting gpc for 4, 8:   
12345678-,  
  
starting gpc for 5, 0:   
-23456789,  
starting gpc for 5, 1:   
1-3456789,  
starting gpc for 5, 2:   
12-456789,  
starting gpc for 5, 3:   
123-56789,  
starting gpc for 5, 4:   
1234-6789,  
starting gpc for 5, 5:   
12345-789,  
starting gpc for 5, 6:   
123456-89,  
starting gpc for 5, 7:   
1234567-9,  
starting gpc for 5, 8:   
12345678-,  
  
starting gpc for 6, 0:   
-23456789,  
starting gpc for 6, 1:   
1-3456789,  
starting gpc for 6, 2:   
12-456789,  
starting gpc for 6, 3:   
123-56789,  
starting gpc for 6, 4:   
1234-6789,  
starting gpc for 6, 5:   
12345-789,  
starting gpc for 6, 6:   
123456-89,  
starting gpc for 6, 7:   
1234567-9,  
starting gpc for 6, 8:   
12345678-,  
  
starting gpc for 7, 0:   
-23456789,  
starting gpc for 7, 1:   
1-3456789,  
starting gpc for 7, 2:   
12-456789,  
starting gpc for 7, 3:   
123-56789,  
starting gpc for 7, 4:   
1234-6789,  
starting gpc for 7, 5:   
12345-789,  
starting gpc for 7, 6:   
123456-89,  
starting gpc for 7, 7:   
1234567-9,  
starting gpc for 7, 8:   
12345678-,  
  
starting gpc for 8, 0:   
-23456789,  
starting gpc for 8, 1:   
1-3456789,  
starting gpc for 8, 2:   
12-456789,  
starting gpc for 8, 3:   
123-56789,  
starting gpc for 8, 4:   
1234-6789,  
starting gpc for 8, 5:   
12345-789,  
starting gpc for 8, 6:   
123456-89,  
starting gpc for 8, 7:   
1234567-9,  
starting gpc for 8, 8:   
12345678-,

In addition, I have added the boardPrinter function as well as a new function called setFilename (replaces isFilenameValid). The reason for the latter is because of the fact that the process of obtaining the filename and validating it is much more complex with the use of c-strings, so it’s much easier to just incorporate it all into one function. Another reason is because I want to eventually implement the program so that you can re-enter all of the information if needed.

Note: The main function is purely in a testing state right now, and doesn’t even remotely represent what it’s going to look like in the future.

Documentation regarding filename restrictions can be found in the link below. <https://msdn.microsoft.com/en-us/library/aa365247>

Here’s the code so far:

#include<fstream>

#include<iostream>

using namespace std;

char \* getPossibleCombos(char board[9][10],int row, int col){

//first char in possibilities is \0 if error with board is found, actual results otherwise

char\* possibilities= new char[10];

possibilities[0]='1';possibilities[1]='2';

possibilities[2]='3';possibilities[3]='4';

possibilities[4]='5';possibilities[5]='6';

possibilities[6]='7';possibilities[7]='8';

possibilities[8]='9';possibilities[9]='\0';

char cellVal=board[row][col];

//cout<<"cellVal is "<<cellVal<<endl;

for(int r=0;r<9;r++){//go through the row

int posIndex=0;//holds index in c-string possibilities of found char, 9 otherwise

while(posIndex<9 && possibilities[posIndex]!=board[row][r]){

posIndex++;

}

//cout<<"parsing row, posIndex is "<<posIndex<<", board char is "<<board[row][r]<<endl;

if(posIndex<9 && r!=col && cellVal=='0'){//if a number is found along row that hasn't been seen

possibilities[posIndex]='-';

}

if(posIndex<9 && r!=col && board[row][r]==cellVal && cellVal!='0'){

possibilities[0]='!';

return possibilities;

}

}

//cout<<possibilities<<endl;

for(int c=0;c<9;c++){//go through the column

int posIndex=0;

while(posIndex<9 && possibilities[posIndex]!=board[c][col]){

posIndex++;

}

if(posIndex<9 && c!=row && cellVal=='0'){//if a number is found along column that hasn't been seen

possibilities[posIndex]='-';

}

if(posIndex<9 && c!=row && board[c][col]==cellVal && cellVal!='0'){

possibilities[0]='!';

return possibilities;

}

}

//TIME FOR THE GRID

for(int smallRow=(row/3)\*3;smallRow<((row/3)\*3)+3;smallRow++){

for(int smallCol=(col/3)\*3;smallCol<((col/3)\*3)+3;smallCol++){

//cout<<row<<col<<"= "<<cellVal<<"\t"<<smallRow<<smallCol<<"= "<<board[smallRow][smallCol]<<endl;

int posIndex=0;

while(posIndex<9 && possibilities[posIndex]!=board[smallRow][smallCol]){

posIndex++;

}

if(posIndex<9 && board[smallRow][smallCol]!=cellVal && cellVal=='0'){

//if a number is found in 3x3 grid that hasn't been seen

possibilities[posIndex]='-';

}

if(posIndex<9 && board[smallRow][smallCol]==cellVal && cellVal!='0' && (smallCol!=col || smallRow!=row)){

//duplicate of cellVal found in 3x3 grid (error in board)

possibilities[0]='!';

return possibilities;

}

}

}

if(cellVal!='0'){

for(int i=0;i<9;i++){

if(possibilities[i]!=cellVal){possibilities[i]=' ';}

}

}

return possibilities;

}

// !!!! END OF GETPOSSIBLECOMBOS !!!!

char \* setFilename(){// https://msdn.microsoft.com/en-us/library/aa365247

bool isFilenameValid=false;

char\* filename= new char[257];//256 char limit + null char

while(!isFilenameValid){

bool errorFound=false;

int cellInRow=0;

cout<<"Results ready for output. Please enter your filename "<<endl;

cin.getline(filename,257);

int cStringSize=cin.gcount();//last stream read (includes null, excludes carriage return)

if(cStringSize>257){//row is incorrect size

cin.clear();

cin.ignore(1024,'\n');

cout<<"Error: filename is too long."<<endl;

errorFound=true;

}

if(filename[0]=='\0' && !errorFound){//if filename length >=1

char illegalCharacters[11]={'<', '>', ':', '\"', '\\', '/', '|', '?', '\*', '\0'};

for(int illChar=0;illChar<11 && !errorFound;illChar++){

int nameIndex=0;

while(illegalCharacters[illChar]!=filename[nameIndex] && nameIndex<cStringSize-1){

nameIndex++;

}

if(nameIndex<cStringSize){

cout<<"Error: filename contains illegal character "<<illegalCharacters[illChar]<<endl;

errorFound=true;

}

}

}

if(filename[3]=='\0' && !errorFound){

char reservedNames[4][4]={{'C','O','N','\0'},{'P','R','N','\0'},{'A','U','X','\0'},{'N','U','L','\0'}};

for(int i=0;i<4 && !errorFound;i++){

int charCount=0;

for(int j=0;j<3;j++){

if(reservedNames[i][j]==filename[j]){charCount++;}

}

if(charCount==3){

cout<<"Error: filename is reserved"<<endl;

errorFound=true;

}

}

}

if(filename[4]=='\0' && !errorFound){

char reservedNames[2][4]={{'C','O','M','\0'},{'L','P','T','\0'}};

for(int i=0;i<2 && !errorFound;i++){

int charCount=0;

for(int j=0;j<3;j++){

if(reservedNames[i][j]==filename[j]){charCount++;}

}

if(charCount==3 && (filename[3]-0>=49 && filename[3]-0<=57)){

//reserved names in this group have some number at the end from 1 to 9.

cout<<"Error: filename is reserved"<<endl;

errorFound=true;

}

}

}

// {"COM1","COM2","COM3","COM4","COM5","COM6","COM7","COM8","COM9","LPT1","LPT2","LPT3","LPT4","LPT5","LPT6","LPT7","LPT8","LPT9"};

//cout<<"supposed end char is"<<filename[cStringSize-2]<<", cStringSize is "<<cStringSize<<endl;

if(filename[cStringSize-2]==' ' && !errorFound){

cout<<"Error: space at end of filename"<<endl;

errorFound=true;

}

if(filename[cStringSize-2]=='.' && !errorFound){

cout<<"Error: period at end of filename"<<endl;

errorFound=true;

}

if(!errorFound){

isFilenameValid=true;

}

}

return filename;

}

void boardMaker(char board[9][10]){

// cout<<"start boardmaker"<<endl;

// for(int i=0;i<9;i++){

// for(int j=0;j<9;j++){

// cout<<board[i][j];

// }

// }

// cout<<endl;

int currentRow=0;

while(currentRow<9){

char row[10];

int cellInRow=0;

cout<<"Enter info for row "<<currentRow<<endl;

cin.getline(row,10);

if(cin.gcount()!=10){//row is incorrect size

cin.clear();

cin.ignore(256,'\n');

cout<<"Error: row "<<currentRow<< " is of incorrect size."<<endl;

currentRow--;

}

while(cellInRow<=9 && cin.gcount()==10){

int charNumToRealNum=row[cellInRow]-'0';

if((charNumToRealNum>9 || charNumToRealNum<0) && charNumToRealNum!=-48){

//checks if its a real number or terminating null char

cin.clear();

cin.ignore(256,'\n');

cout<<"Error: illegal character in row "<<currentRow<<"."<<endl;

currentRow--;

}

board[currentRow][cellInRow]=row[cellInRow];

//cout<<row[cellInRow];

cellInRow++;

} //while

currentRow++;

}

// cout<<"end boardmaker"<<endl;

// for(int k=0;k<9;k++){

// for(int m=0;m<9;m++){

// cout<<board[k][m];

// }

// }

// cout<<endl;

}

void boardPrinter(char potentialAnswers[9][9][10],char filename[]){

for(int row=0;row<9;row++){

for(int rowInCell=0;rowInCell<3;rowInCell++){

for(int cell=0;cell<9;cell++){

for(int cellInCell=rowInCell\*3;cellInCell<(rowInCell\*3)+3;cellInCell++){

cout<<potentialAnswers[row][cell][cellInCell];

if(cell==8 && (cellInCell+1)%3==0){

cout<<"";

}

else{

cout<<" ";

}

}

if(cell!=8 && (cell+1)%3==0){cout<<"|| ";}//border of 3x3 grid

if(cell!=8 && (cell+1)%3!=0){cout<<"| ";}//border of cell/column

if(cell==8){cout<<endl;}

}

}

if((row+1)%3==0 && row!=8){//border of 3x3 grid

cout<<"======================++=======================++======================"<<endl;

}

if((row+1)%3!=0 && row!=8){//border of row

cout<<"------+-------+-------||-------+-------+-------||-------+-------+------"<<endl;

}

}

}

int main(){

char board[9][10];//9 rows of 9 character arrays (not including null character)

char potentialAnswers[9][9][10];//9 rows, 9 columns, 9 smaller cells in a cell (not including null character)

//char board[9][10]={

// {'3','9','1','5','0','0','2','0','6','\0'},{'2','8','5','0','9','0','0','0','0','\0'},{'6','7','4','0','1','0','5','8','0','\0'},{'0','2','0','4','0','0','0','3','7','\0'},{'0','0','0','9','0','6','0','0','0','\0'},{'1','4','0','0','0','2','0','9','0','\0'},{'0','1','7','0','2','0','0','0','8','\0'},{'0','0','0','0','6','0','7','0','0','\0'},{'8','0','3','0','0','7','0','0','0','\0'},

//};

// 391500206

// 285090000

// 674010580

// 020400037

// 000906000

// 140002090

// 017020008

// 000060700

// 803007000

// cout<<"input"<<endl;

// for(int i=0;i<9;i++){

// for(int j=0;j<9;j++){

// cout<<board[i][j];

// }

// }

// cout<<endl;

boardMaker(board);

// cout<<"after boardmaker"<<endl;

// for(int i=0;i<9;i++){

// for(int j=0;j<9;j++){

// cout<<board[i][j];

// }

// cout<<",";

// }

// cout<<endl<<endl;

// cout<<"moms"<<endl;

for(int row=0;row<9;row++){

for(int col=0;col<9;col++){

//cout<<"starting gpc for "<<row<<", "<<col<<endl;

char \*cellPossibleNums=getPossibleCombos(board,row,col);

for(int cellInCell=0;cellInCell<9;cellInCell++){

potentialAnswers[row][col][cellInCell]=cellPossibleNums[cellInCell];

//cout<<cellPossibleNums[cellInCell];

}

//cout<<",";

}

// cout<<endl;

}

// cout<<endl<<endl<<"how results are stored"<<endl;

// for(int i=0;i<9;i++){

// for(int j=0;j<9;j++){

// for(int c=0;c<9;c++){

// cout<<potentialAnswers[i][j][c];

// //cout<<cellPossibleNums[cellInCell];

// }

// cout<<",";

// }

// cout<<endl;

// }

char \*name=setFilename();

//for(int i=0;name[i]!='\0';i++){cout<<name[i];}

boardPrinter(potentialAnswers,name);

return 1;

}

**4/20/17 - Error Handling / The Final Product**

The final version of the program contains some vital changes to the boardPrinter function as well as a complete rework of what the main function is supposed to be. For boardPrinter, I added a filename parameter since we need the name of the file to output to. Ofstream has been added and given the name file. In addition, filename had to be converted to a string, as the .open and .close properties of ofstream will not take c-strings as an argument. Cout keywords have been replaced with file accordingly. Also, I made a small change to how it handles spacing, where it is now done with one if statement rather than one if and and else.

For main, I wanted to implement it with user error in mind. For example, if the user enters a board where a row or column has two identical numbers in it, then the program should allow the user to reenter the board correctly. In addition, if the program finds an error, then it should not be able to continue as a means of both saving time and the prevention of outputting the wrong answer. Also, if a user wants to enter in multiple boards, then the user should not have to reopen the program after each successful output.

The vast majority of the main function is encapsulated in a while loop, where a bool named makeAnotherBoard (true by default) represents the decision of whether or not the user wants to enter in another board. In this first while loop, boardMaker is called and another bool, named isValidCombo (true by default), is declared. isValidCombo is used in a way such that if there’s an error in the board (found by getPossibleCombos), then isValidCombo becomes false. If it’s false, then the whole board-making process starts all over again. If it’s true, then the user is prompted to input a filename and the results are output to that file.

Once this takes place, char response (given a value of \0) and bool notValidResponse (true by default) are declared, and the program enterers another while loop. In this second while loop, the user is prompted as to whether or not he/she wants to enter another board, and must enter either y or n (only implemented lowercase letters, as it’s simply easier to read). Their input replaces the character in response, and the program then checks to see if it’s a valid response by checking its ascii value. If it’s valid, then notValidResponse is set to false, and the program leaves the second while loop. Finally, the program checks to see if response is n, and if it is then makeAnotherBoard is set to false, which makes the program leave the first while loop and subsequently exit the program. If response isn’t no (meaning it’s y), then the entire process starts over again.

Here’s the final product:

/\* 4/20/17

Sudoku Puzzle Assistant

Made by Kyle Vecere III Esq. pHD MD HIJKLMNOP

Info: This program allows you to input a sudoku puzzle, then ouputs to a file the potential

numbers you can put for each cell. Note that for empty cells, a zero must be entered in its place,

and any non-empty cell will NOT contain dashes in its ouput.

\*/

#include<fstream>

#include<iostream>

using namespace std;

char \* getPossibleCombos(char board[9][10],int row, int col){

//first char in possibilities is \0 if error with board is found, actual results otherwise

char\* possibilities= new char[10];

possibilities[0]='1';possibilities[1]='2';

possibilities[2]='3';possibilities[3]='4';

possibilities[4]='5';possibilities[5]='6';

possibilities[6]='7';possibilities[7]='8';

possibilities[8]='9';possibilities[9]='\0';

char cellVal=board[row][col];

for(int r=0;r<9;r++){//go through the row

int posIndex=0;//holds index in c-string possibilities of found char, 9 otherwise

while(posIndex<9 && possibilities[posIndex]!=board[row][r]){

posIndex++;

}

if(posIndex<9 && r!=col && cellVal=='0'){//if a number is found along row that hasn't been seen

possibilities[posIndex]='-';

}

if(posIndex<9 && r!=col && board[row][r]==cellVal && cellVal!='0'){

possibilities[0]='!';

return possibilities;

}

}

for(int c=0;c<9;c++){//go through the column

int posIndex=0;

while(posIndex<9 && possibilities[posIndex]!=board[c][col]){

posIndex++;

}

if(posIndex<9 && c!=row && cellVal=='0'){//if a number is found along column that hasn't been seen

possibilities[posIndex]='-';

}

if(posIndex<9 && c!=row && board[c][col]==cellVal && cellVal!='0'){

possibilities[0]='!';

return possibilities;

}

}

//TIME FOR THE GRID

for(int smallRow=(row/3)\*3;smallRow<((row/3)\*3)+3;smallRow++){

for(int smallCol=(col/3)\*3;smallCol<((col/3)\*3)+3;smallCol++){

//cout<<row<<col<<"= "<<cellVal<<"\t"<<smallRow<<smallCol<<"= "<<board[smallRow][smallCol]<<endl;

int posIndex=0;

while(posIndex<9 && possibilities[posIndex]!=board[smallRow][smallCol]){

posIndex++;

}

if(posIndex<9 && board[smallRow][smallCol]!=cellVal && cellVal=='0'){

//if a number is found in 3x3 grid that hasn't been seen

possibilities[posIndex]='-';

}

if(posIndex<9 && board[smallRow][smallCol]==cellVal && cellVal!='0' && (smallCol!=col || smallRow!=row)){

//duplicate of cellVal found in 3x3 grid (error in board)

possibilities[0]='!';

return possibilities;

}

}

}

if(cellVal!='0'){

for(int i=0;i<9;i++){

if(possibilities[i]!=cellVal){possibilities[i]=' ';}

}

}

return possibilities;

}

// !!!! END OF GETPOSSIBLECOMBOS !!!!

char \* setFilename(){// https://msdn.microsoft.com/en-us/library/aa365247

bool isFilenameValid=false;

char\* filename= new char[257];//256 char limit + null char

while(!isFilenameValid){

bool errorFound=false;

cout<<"Results ready for output. Please enter your filename "<<endl;

cin.getline(filename,257);

int cStringSize=cin.gcount();//last stream read (includes null, excludes carriage return)

if(cStringSize>257){//row is incorrect size

cin.clear();

cin.ignore(1024,'\n');

cout<<"Error: filename is too long."<<endl;

errorFound=true;

}

if(filename[0]=='\0' && !errorFound){//if filename length >=1

char illegalCharacters[11]={'<', '>', ':', '\"', '\\', '/', '|', '?', '\*', '\0'};

for(int illChar=0;illChar<11 && !errorFound;illChar++){

int nameIndex=0;

while(illegalCharacters[illChar]!=filename[nameIndex] && nameIndex<cStringSize-1){

nameIndex++;

}

if(nameIndex<cStringSize){

cout<<"Error: filename contains illegal character "<<illegalCharacters[illChar]<<endl;

errorFound=true;

}

}

}

if(filename[3]=='\0' && !errorFound){

char reservedNames[4][4]={{'C','O','N','\0'},{'P','R','N','\0'},{'A','U','X','\0'},{'N','U','L','\0'}};

for(int i=0;i<4 && !errorFound;i++){

int charCount=0;

for(int j=0;j<3;j++){

if(reservedNames[i][j]==filename[j]){charCount++;}

}

if(charCount==3){

cout<<"Error: filename is reserved"<<endl;

errorFound=true;

}

}

}

if(filename[4]=='\0' && !errorFound){

char reservedNames[2][4]={{'C','O','M','\0'},{'L','P','T','\0'}};

for(int i=0;i<2 && !errorFound;i++){

int charCount=0;

for(int j=0;j<3;j++){

if(reservedNames[i][j]==filename[j]){charCount++;}

}

if(charCount==3 && (filename[3]-0>=49 && filename[3]-0<=57)){

//reserved names in this group have some number at the end from 1 to 9.

cout<<"Error: filename is reserved"<<endl;

errorFound=true;

}

}

}

if(filename[cStringSize-2]==' ' && !errorFound){

cout<<"Error: space at end of filename"<<endl;

errorFound=true;

}

if(filename[cStringSize-2]=='.' && !errorFound){

cout<<"Error: period at end of filename"<<endl;

errorFound=true;

}

if(!errorFound){

isFilenameValid=true;

}

}

return filename;

}

// !!!! END OF SETFILENAME !!!!

void boardMaker(char board[9][10]){

int currentRow=0;

while(currentRow<9){

char row[10];

int cellInRow=0;

cout<<"Enter info for row "<<currentRow<<endl;

cin.getline(row,10);

if(cin.gcount()!=10){//row is incorrect size

cin.clear();

cin.ignore(256,'\n');

cout<<"Error: row "<<currentRow<< " is of incorrect size."<<endl;

currentRow--;

}

while(cellInRow<=9 && cin.gcount()==10){

int charNumToRealNum=row[cellInRow]-'0';

if((charNumToRealNum>9 || charNumToRealNum<0) && charNumToRealNum!=-48){

//checks if its a real number or terminating null char

cin.clear();

cin.ignore(256,'\n');

cout<<"Error: illegal character in row "<<currentRow<<"."<<endl;

currentRow--;

}

board[currentRow][cellInRow]=row[cellInRow];

//cout<<row[cellInRow];

cellInRow++;

} //while

currentRow++;

}

}

// !!!! END OF BOARDMAKER !!!!

void boardPrinter(char potentialAnswers[9][9][10],char filename[]){

ofstream file;

string actualName(filename);//has to be converted to string in order to pass to below line

file.open(actualName);

for(int row=0;row<9;row++){

for(int rowInCell=0;rowInCell<3;rowInCell++){

for(int cell=0;cell<9;cell++){

for(int cellInCell=rowInCell\*3;cellInCell<(rowInCell\*3)+3;cellInCell++){

file<<potentialAnswers[row][cell][cellInCell];

if(cell!=8||(cellInCell+1)%3!=0){

file<<" ";

}

}

if(cell!=8 && (cell+1)%3==0){file<<"|| ";}//border of 3x3 grid

if(cell!=8 && (cell+1)%3!=0){file<<"| ";}//border of cell/column

if(cell==8){file<<endl;}

}

}

if((row+1)%3==0 && row!=8){//border of 3x3 grid

file<<"======================++=======================++======================"<<endl;

}

if((row+1)%3!=0 && row!=8){//border of row

file<<"------+-------+-------||-------+-------+-------||-------+-------+------"<<endl;

}

}

file.close();

}

// !!!! END OF BOARDPRINTER

int main(){

char board[9][10];//9 rows of 9 character arrays (not including null character)

char potentialAnswers[9][9][10];//9 rows, 9 columns, 9 smaller cells in a cell (not including null character)

bool makeAnotherBoard=true;

while(makeAnotherBoard){

boardMaker(board);

bool isValidCombo=true;

for(int row=0;row<9;row++){

for(int col=0;col<9 && isValidCombo;col++){//no need to continue through rest of board if theres an error

char \*cellPossibleNums=getPossibleCombos(board,row,col);

if(cellPossibleNums[0]=='!'){

cout<<"Error: duplicate in row "<<row<<", column "<<col<<"\nYou must enter your board information correctly"<<endl;

isValidCombo=false;

}

for(int cellInCell=0;cellInCell<9 && isValidCombo;cellInCell++){

potentialAnswers[row][col][cellInCell]=cellPossibleNums[cellInCell];

}

}

if(!isValidCombo){//start from scratch if there's an error in the board

isValidCombo=true;

boardMaker(board);

row-=(row+1);

}

}

char \*name=setFilename();

boardPrinter(potentialAnswers,name);

cout<<"Board outputted to file."<<endl<<"Check the directory this program is located in"<<endl<<endl;

bool notValidResponse=true;

char response='\0';

while(notValidResponse){

cout<<"Would you like to continue entering in sudoku boards? y or n"<<endl;

cin>>response;

if(response==121 || response==110){//y or n (lowercase only)

notValidResponse=false;

}

else{

cout<<"Invalid response. Please try again"<<endl;

cin.clear();

cin.ignore(1024,'\n');

}

}

if(response=='n'){makeAnotherBoard=false;}

}

cout<<"This program will now close. Goodbye!"<<endl;

return 1;

}