Project I

Battleship

CIS-17A-48206

November 05, 2022

Danielle Fernandez

# Table of Contents

[Introduction](#_Introduction).…………………………………………..... 2

[Development Summary](#_Development_Summary)….……………………………... 2

[Sample Inputs/Outputs](#_Sample_Outputs:)….…………………………….... 3

[Flowchart](#_Flowchart)….……………………………………………. 5

[Pseudo Code](#_Pseudo_Code) ………………………………………….. 11

[Major Variables](#_892pz4i5n333) ………………………………………. 13

[Checklist](#_Checklist) …….……………………………………….. 14

[Reference](#_Reference) ……….……………………………………..15

[Program](#_Program)……………………………………………….. 15

## 

## **Introduction**

This game simulates a slimmed-down version of the classic board game, Battleship.

**Objective:**

Guess the location of your opponent's ship.

**Rules**:

* Each player has 1 ship per game.
* The board has 20 different locations that are represented by an integer between 1 and 20 (For ease of use, the application automatically generates the ship’s location with a random number).
* When a player successfully guesses their opponent's ship location, then a “HIT” message is printed. If their guess is wrong, then a “MISS” message is printed, and it is the other player’s turn. If both player’s guess wrong, then a “You both missed. Try again” message is printed.
* Players continue taking one guess at a time until someone gets a “HIT”.
  + It can sometimes take over 10 guesses before a player is correct, so for ease of use, the computer automatically generates each player’s guess with a random number.
* Once someone gets a win, a scoreboard will show how many wins each player has earned. As well as how many games are left.
* The game asks the user to press “Enter” before proceeding to the next game, so they can review the scoreboard.
* For ease of use, the max number of games is preset to five and the game automatically generates each player’s guess and ship’s location after the user presses “Enter”.

## **Development Summary**

|  |  |
| --- | --- |
| Lines of code  (including h files) | 521 |
| Comment lines | 89 |
| Blank lines | 116 |
| Total lines of source file | 316 |

This game covers chapters nine through twelve in the [textbook](#_Reference) by illustrating pointers, dynamic memory, c-strings, strings, structures, writing structures to binary files, and random-access files.

My initial challenge was figuring out how to convert my project 2 Battleship game from CIS-5 to utilize structures. I thought it would be easy to convert an already functioning game, but I was partially wrong. My project 2 game is an advanced version using concepts I have used for years and feel very confident using. Whereas my experience with structures and pairing them with binary files is minimal. I understand how they work theoretically and my experience with using them has been limited to the last few weeks. I wasn’t even sure where to start converting my old code. I was passing two-dimensional arrays to numerous functions. My first attempt to change a two-dimensional array to a pointer failed. My second attempt consisted of trying to convert a static one-dimensional array went better, but then I changed something and couldn’t get the code to work again. I resolved my problem by going back to the final version of my Project 1 Battleship game from CIS-5. This version had a working game and did not use functions or two-dimensional arrays yet. This version matched my abilities to work with structures.

My second dilemma involved writing and reading my Score structure to a binary file. I tried to write a structure that had a structure inside of it to binary but could not get it work. My solution was to create a new and very basic structure, and then write that to binary. I was successful in writing a simple structure to binary as well as locating a random record within it.

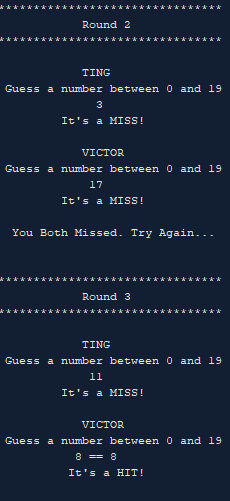
My third dilemma was trying to figure out why I couldn’t write a Score structure read the correct data from binary. Eventually I asked Dr. Lehr for help, and he told me that I was not able to pass a pointer of a pointer of a structure to binary the same way I was handling my simple structure. I thought I could write the entire contents of structure to binary by sending its reference. I imagined it was like passing an array to a function, but I completely skipped the part where you still need to write each member of the structure to binary. I ended up giving up on this because I did not need it to meet the project’s requirements.

My fourth dilemma came when one of my pushes to Github got messed up and my latest version was overwritten by a previous version. I am not sure how I managed to do that, but after that I started keeping a copy of my latest version in a text file that was not connected to my Github.

My final hurdle came when I got the smart idea to clean my code within fillScore( ) function by getting rid of repetitive code and replacing it with functions. I moved variable declarations from one function to another, and things got hairy. Variables were being reinitialized to zero each the function was called. I was never able to get the number of total games variable to accumulate properly in play( ) function. I resolved the issue by not setting the variables to zero and setting structures member to zero in the beginning of the function that sets a Score structure.

#### **Sample Inputs:** Enter, Enter, Enter, Enter, Enter

#### **Sample Outputs:**



Text

Description automatically generated

#### **Version 1**

* Changed P1 & P2 strings to pointers.
* Created Player structure and set number of wins as a structure member.
* Added name, total rounds to structure.
* Added char array for choices to randomly fills game board with a ship or blank and writes it to a text file.
* Created game banner function that passes a pointer to a structure.

#### **Version 2**

* Added number of players to Score structure.
* Added player's guess to Player structure.
* Changed names array to all lowercase so I could utilize toupper().
* Created a pointer for number of rounds, passed it to a function and added 1 to it each time it was called.
* Changed names array to all lowercase so I could utilize toupper()
* Created string \*toUpper(string \*) with a parameter that accepts a pointer to an array to the array index that I'm looking to convert to capitals.

#### **Version 3**

* Changed Player.name to a char[] instead of string for binary files.
* Created fillScore() to dynamically create and fill Score and Player structures.
* The function randomly selects a player's name from a static array and initializes the name member.
* Added enumeration and referenced it inside of for loops.

#### **Version 4**

* Changed choices from 2D to a new structure Choices.
* Wrote a Choices structure to binary and text files.
* Then I looked for a specific rand() record, read it from the binary file.
* Confirmed the correct record was located by comparing it to the text file.

#### **Version 5**

* Wrote to Choices & Scores structures to separate binary files because I can't get scores to read from binary correctly.
* Had choices binary pick a random letter and then switched it back to reference
* chcePtr that points to array of Ships or blanks.
* Fixed scores->board[ ][ ] bug by randomly picking a row and col inside of its brackets.
* Got rid of inFile and set number of players inside fillScore().
* Cleaned up play() by printing end of game results in the printScore();
* PrintScore() displays their opponent’s corresponding row, so you can compare for correct guesses.

## **Flowchart**

I have included the main function’s flowchart below. Please see the Documents folder for the complete flowchart that illustrates each function’s flowchart.

Diagram, text

Description automatically generated

Diagram

Description automatically generatedDiagram

Description automatically generatedDiagram

Description automatically generated

## **Pseudo Code for main( )**

* Include libraries: iostream, iomanip, cmath, cstdlib, fstream, cstring, string,cctype,ctime.
* Include user libraries. 3 structures: Choices.h, Player.h, Score.h
* Declare enumeration numbers to reference [0,10]
* Declare function protypes
* Start main function.
* Set random number seed.
* Declare fstream variables for text and binary files.
* Declare and set data type variables.
* Declare string array, names with 7 different first names in lowercase letters.
* Fill a 2D game board with ships or blanks
  + Declare a static char array. Set with letters to represent ships or blank spaces on a game board.
  + Declare char pointer, \*bordPtr. Set to nullptr.
  + Assigns address of static char array to bordPtr
  + Declare a char 2D pointer, \*\*board2D. Set to nullptr.
  + Set board2D with random letters from bordPtr by calling fill2DPtr( ). Pass bordPtr and its array size as arguments.
* Fill 1 instances of a Score structure
  + Create a pointer to the Score structure, \*score.
  + Initialize score to Score by calling fillScore( ). Pass names array and reference \*\*pointer.
  + Set game board that’s a member of score by calling setBoard( ). Pass reference to score and \*\*pointer reference as arguments.
* Open binary file for reading, writing, or binary.
  + Conditional to check if binary file does not exist, then print message.
  + Open text file for reading or writing.
  + Create 20 records inside of binary and text files with for loop
    - Create a pointer to the Choices structure, \*choice, and set with a function that returns a pointer to Choices that has been set with values. Pass bordPtr and its size as arguments.
    - Write each record to binary and text files
    - Delete choice pointer
  + Read a random record from binary file. Use text file to confirm correct record was read.
    - Get random number [0,20]
    - Create a pointer, \*choice, to Choices structure and set with a function call to read binary file and returns the record looking for. Pass binary file and record number as arguments.
    - For loop on the size of choice->size member.
      * Print record contents
    - Delete choice
    - Close binary and text files
* De-allocate any dynamic memory that was created.
* Reset pointers to nullptr.

**Pseudo for functions**

**void setBoard(Score \*score, char \*\*board2D)**

* Create and set dynamic 2D array, \*\*board, for each player.
  + Loop based on number of players inside of score
  + Set number of rows and columns the \*\*board will have
  + Create a new array of rows inside of players->board for 2D array
  + Allocate each row with 10 columns because \*\*board should be a 2D array with a for loop
  + Set \*\*board [ ] [ ] with random elements from \*\*board2D
  + Print to confirm board was set inside of score

**char \*\*fill2DPtr (char \*bordPtr, const int SIZE17)**

* Function accepts a pointer, and its size. Return reference to a 2D pointer
  + Declare \*\*pointer. Set to nullptr.
  + Allocate pointer with a char array of pointer
  + For loop to allocate 10 columns in each row
  + For loop to set each element in \*\*pointer by randomly filling it with ships or blank from bordPtr.
  + Return pointer to main

**Choices \*readBin1(fstream &binFile, int recInd)**

* Read binary file with 20 records and return one record from it
  + Create new instance of Choice structure
  + Calculate cursors size
    - Set cursor to beginning of file
    - Call .read( ) to read in the size of choice->size
    - Calculate the sum # of bytes by multiplying recInd & size of each data type in Choice structure
  + Call seekg( ) to find record we’re looking for. Pass the size of the actual record and to read from the beginning of that record.
  + Read binary record’s size.
  + Allocate memory for char array
  + Read binary record’s char [ ] and save to choice’s array member
  + Allocate choice’s indx member with integer array
  + Read binary record and save it choice’s indx member
  + Return choice to main

**Score \*fillScore (string names[], char \*\*bordPtr)**

* Fill instance of Score structure by playing game
* While loop
  + Initialize each player's ship to random number between 1-20 to represent its location on game board
  + Set p1\_correct and p2\_correct to default starting values == false
  + Display “BATTLESHIP” banner
  + Loops until a player correctly guesses opponents ship location.
    - Display round variable
    - Call bool function to check if player 1’s guess is correct or not.
    - If player 1 is wrong, then call bool function to check if player 2’s guess is correct or not.
    - If both players are wrong, then print message
  + Decrement number of games left
  + Set score’s total Games member to the number of games left.
  + Call function to print score structure
  + If number of games is not zero, then call function to pause the game to view score
  + Reset variables for the next game
  + Return score to main

**bool play(Score \*score, int a, int b, int &round)**

* Returns if player's guess is correct or not. Changes value of round
  + Automatically generate player’s guess to a rand num between 1-20
  + if conditional to check if p1Guess equals the location of opponent’s ship
    - Increment player’s win by 1
    - Decrement nGmsLft by 1
    - set score’s isRight member to true
    - Calculate total number of games won & number rounds played
    - Call function to print player was correct
    - Return true to fillScore( )
  + if player1 guess is wrong display MISS message
  + set score’s isRight member to false.
  + Call function to print that player was wrong
  + Return false to fillScore( )

## **Major Variables**

fstream binFile;

fstream outFile;

char \*bordPtr;

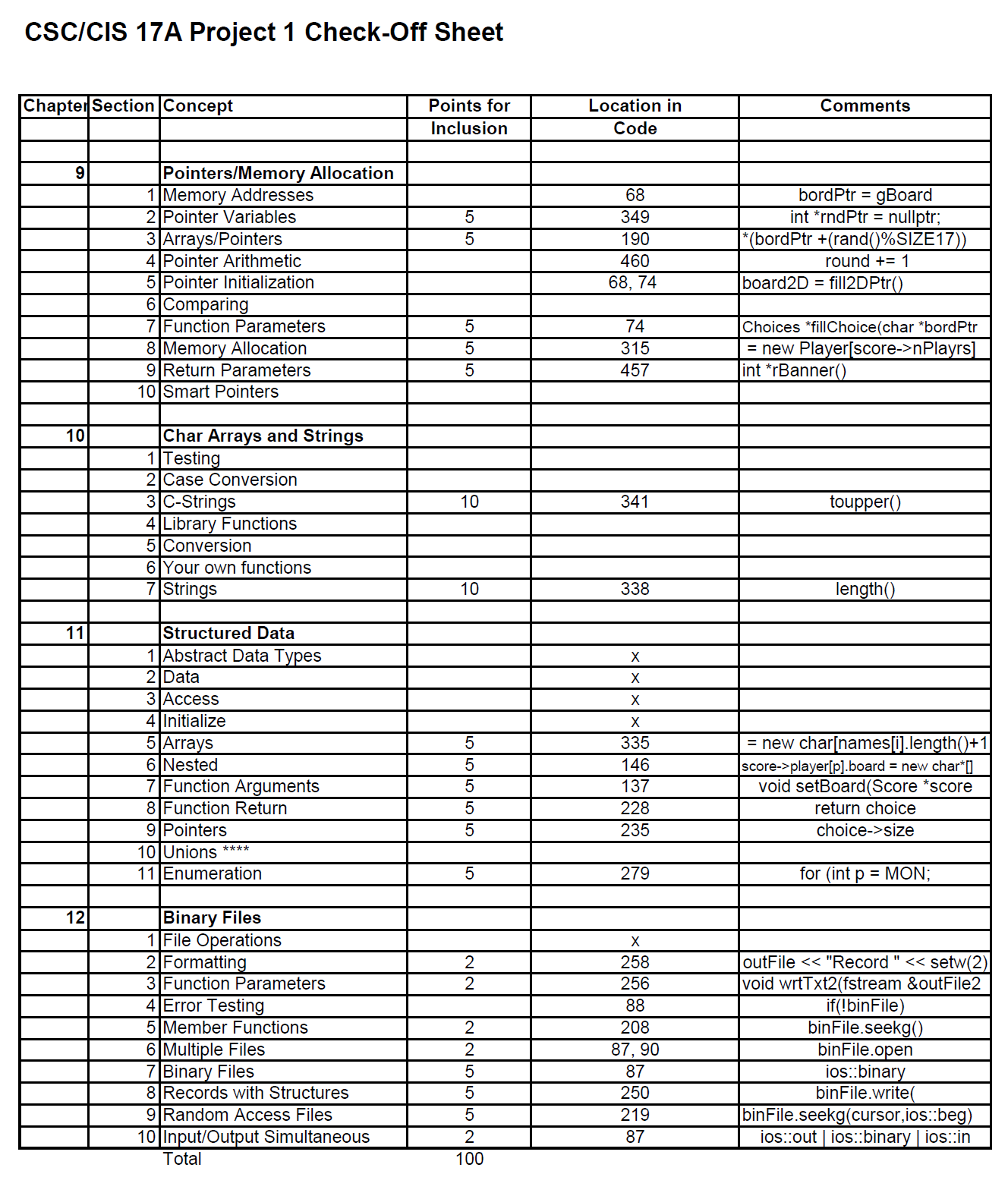
char \*\*board2D;

Score \*score;

Choices \*choice;

int cursor;

#### **Checklist**



## **Reference**

1. Gaddis, Tony. *Starting out with C++: From Control Structures through Objects*. 9th ed., Pearson, 2018.
2. Lehr, Mark. “2022\_Spring\_CSC\_CIS\_5/Projects at Master · ml1150258/2022\_spring\_csc\_cis\_5.” *GitHub*, 2022, https://github.com/ml1150258/2022\_Fall\_CSC\_CIS\_17a.

## **Program**

/\* File: main.cpp

\* Author: Danielle Fernandez

\* Created: 11-04-2022 @ 4 PM

\* Purpose: CIS 17A Project 1. Covers chapters 9-12 in Tony Gaddis. Battleship v1

Version 6f:

\*/

// System Libraries:

#include <iostream> // cin, cout

#include <iomanip> // fixed, setprecision()

#include <cmath> // round()

#include <cstdlib> // rand()

#include <fstream> // fstream

#include <cstring> // char [] library

#include <string> // length() library

#include <ctime> // time library for rand()

#include <cctype> // toupper()

using namespace std;

// User libraries

#include "Choices.h"

#include "Player.h"

#include "Score.h"

// Global Constants

// Physics/Chemistry/Math/Conversions

enum nums {

ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN

};

// Function prototypes

Choices \*fillChoice(char \*,const int); // fills Choice structure

void wrtTxt1(fstream&,Choices \*,int); //write Choices structure to text file

void wrtBin1(fstream &,Choices \*); // write Choices structure to binary file

Choices \*readBin1(fstream &,int); // read binary file

Score \*fillScore(string [],char \*\*); // fill Score structure by playing game

bool play(Score \*,int,int,int &); // run's each player's guess

char \*\*fill2DPtr(char \*,const int); // fills a dynamic 2D pointer

void setBoard(Score \*score,char \*\*ptr2D); // sets 2D pointer with Score structure

void prntScore(Score \*); // prints Score structure

void pause(); // pauses game to allow user to view results before proceeding

void banner(Score \*,string); // print banner

int \*rBanner(int &); // print round banner

void hitMiss(Score \*,int,int,bool); // print message after each player guesses

int main(int argc, char\*\* argv) {// Program execution begins here

// set random number seed

srand(static\_cast<unsigned int> (time(0)));

// declare variables

fstream binFile; // write Choices structure to binary file

fstream outFile; // write Choices structure to text file

const int SIZE17 = 17; // # of characters in choices array

int nRecords = 20,

recInd; // index for random record

string names[SEVEN] = {"mom", "bart", "homer", "jillian", "ting", "victor", "danielle"};

// Fill game board with ships or blanks

//char gBoard[SIZE17 + 1] = "SbSsaSsehSjSwpSsx"; // S=ship B=blank

char gBoard[SIZE17 + 1] = "SbSsBSsbBSBSbbSsb"; // S=ship B=blank

char \*bordPtr = nullptr; // declare pointer to a char

bordPtr = gBoard; // assigns address of gBoard array to bordPtr

// 2D array to represent a player's game board

char \*\*board2D = nullptr;

// Call fill2Darr func to fill \*\*board2D with random letters from \*bordPtr

board2D = fill2DPtr(bordPtr,SIZE17);

// creating 1 new Score structure

Score \*score = new Score;

// Initialize pointer to Score by calling fillScore function

score = fillScore(names,board2D);

// fill \*\*board within Score structure

setBoard(score,board2D);

/\*\*\*\*\*\*\*\*\*\*\* 1 Write Choices structure to binary file \*\*\*\*\*\*\*\*\*\*\*\*\*\*/

binFile.open("2choices.bin", ios::in|ios::out|ios::binary);

if(!binFile) cout <<"Error opening choices.bin\n";

outFile.open("choices.txt", ios::in|ios::out); // Create text file to write to

// Create 20 records of Choice structures in binary & text files

for (int record = 0; record < nRecords; record++) {

// Declare pointer & set with a Choice structure

Choices \*choice = fillChoice(bordPtr,SIZE17);

// write each instance of Choice to binary & text files

wrtTxt1(outFile,choice,record);

wrtBin1(binFile,choice);

delete choice;

}

/\*\*\*\*\*\*\*\*\*\*\* 1 Read Binary Files \*\*\*\*\*\*\*\*\*\*\*\*\*\*/

recInd = rand() % nRecords;

Choices \*choice = readBin1(binFile,recInd);

cout << "Reading choices.bin...\nLocating record #"<<recInd<<"\nFound Choices record # " << recInd << "\n";

for(int i=ZERO;i<choice->size;i++){

cout << choice->arr[choice->indx[i]];

if(i%TEN==NINE) cout<<endl;

}

delete choice;

// close file

binFile.close();

outFile.close();

//de-allocate dynamic memory

for (int p = 0; p < score->nPlayrs; p++) {

for (int row = 0; row < score->player[p].rows; row++) {

delete []score->player[p].board[row]; //Deletes the Data row by row

}

}

//delete []choice->a;

delete []score->player;

delete score;

score = nullptr;

bordPtr = nullptr;

// exit code

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\* FUNCTION DEFINITIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void setBoard(Score \*score,char \*\*board2D){

// Create & Set dynamic 2D array inside of players for \*\*board

for (int p = 0; p < score->nPlayrs; p++) {

score->player[p].rows = TWO; // set rows to 2

score->player[p].cols = TEN; // set columns to 10

// create new array of rows inside of players->board for 2D array

score->player[p].board = new char\*[score->player[p].rows];

// allocate each row with 10 columns because \*\*board should be a 2D array

for (int r = 0; r < score->player[p].rows; r++) {

score->player[p].board[r] = new char[score->player[p].cols];

}

//set \*\*board[][] with

cout<<score->player[p].name << " board in Score structure: \n";

int r,c;

for (int row = 0; row < score->player[p].rows; row++) {

for (int col = 0; col < score->player[p].cols; col++) {

r=rand()%TWO;

c=rand()%TEN;

// Setting structure to a rand() value from 2D char[][]

score->player[p].board[row][col] = board2D[r][c];

cout<< score->player[p].board[row][col];

if(col%10==9) cout<<endl;

}

}

cout<<endl;

}

cout<<endl;

}

// fill \*\*pointer

char \*\*fill2DPtr(char \*bordPtr,const int SIZE17){

char \*\*ptr2D = nullptr;

ptr2D = new char\*[TWO];

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

ptr2D[row] = new char[TEN];

}

// fill 2D array with game board with a random char from gBoard[]

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

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

// setting 2D array to a random value from gBoard[]

//board[row][col] = \*(bordPtr + (rand()%SIZE17));

ptr2D[row][col] = \*(bordPtr + (rand()%SIZE17));

}

}

return ptr2D;

}

// read binary file and return one record

Choices \*readBin1(fstream &binFile,int recInd){

int count=0;

long cursor = 0L;

Choices \*choice = new Choices;

// Find the record by finding the size of one Choice structure &

// multiplying it by the index we're looking for

while(++count<=recInd){

// set cursor to beginning of file

binFile.seekg(cursor,ios::beg);

// reading size of char arr[]

binFile.read(reinterpret\_cast<char \*>(&choice->size),sizeof(int));

// calculates # of bytes by multiplying index & size of each data types in Choice structure

cursor += ( sizeof(int) + choice->size\*sizeof(char) + choice->size\*sizeof(int));

}

//cout << "cursor " << cursor << " bits\n";

// set cursor to beginning of file so it can read that record

binFile.seekg(cursor, ios::beg);

// read 1 record from binary file & save to new pointer

binFile.read(reinterpret\_cast<char \*> (&choice->size), sizeof(int));

choice->arr = new char[choice->size]; // allocate memory for char[]

binFile.read(reinterpret\_cast<char \*>(choice->arr),choice->size\*sizeof(char));

choice->indx = new int[choice->size];

binFile.read(reinterpret\_cast<char \*>(choice->indx),choice->size\*sizeof(int));

return choice;

}

// Declare & fill 1 instances of Choice structure

Choices \*fillChoice(char \*bordPtr,const int SIZE17){

Choices \*choice = new Choices;

choice->size = TWO\*TEN;

choice->arr = new char[choice->size];

choice->indx = new int[choice->size];

//Fill array inside of Choice structure

for(int i=0;i<choice->size;i++){

choice->arr[i]=\*(bordPtr+(rand()%SIZE17)); // fills with S=ship or B=blank

//choice->arr[i]=rand() % 26 + 65; // rand() letters from alphabet

choice->indx[i]=i;

}

return choice;

}

// write Choices structure to binary file

void wrtBin1(fstream &binFile, Choices \*choice) {

binFile.write(reinterpret\_cast<char \*> (&choice->size), sizeof (int));

binFile.write(reinterpret\_cast<char \*> (choice->arr),choice->size\*sizeof(char));

binFile.write(reinterpret\_cast<char \*> (choice->indx),choice->size\*sizeof(int));

}

// write Choices structure to text file

void wrtTxt1(fstream &outFile, Choices \*choice, int recInd) {

int perLine=TEN;

outFile << "Record " << setw(2) << right << recInd << " \n";

for(int i=0;i<choice->size;i++){

outFile << setw(2) << choice->arr[choice->indx[i]];

if(i%perLine==(perLine-1))outFile<<endl;

}

outFile<<endl;

}

// print Score structure member's

void prntScore(Score \*score) {

// Display scoreboard banner

banner(score, "SCOREBOARD");

cout <<"\n "<< score->ttlGams << " games left.\n";

cout << " Total rounds in this game: " << score->ttlRnds << endl << endl;

cout << " Player 1's Ship Location: " << score->player[0].shipLoc << endl

<< " Player 2's Ship Location: " << score->player[1].shipLoc << endl << endl;

for (int p=ZERO; p < score->player[p].rows; p++) {

cout << " Player "<<p+1<<"'s Game board \n";

for (int row = 0; row < score->player[p].rows; row++) {

for (int col = 0; col < score->player[p].cols; col++) {

cout<< "&" << score->player[p].board[row][col];

if(col%TEN==NINE) cout<<endl;

}

}

}

cout << endl << endl;

}

// play game

Score \*fillScore(string names[],char \*\*bordPtr) {

int MAX = TEN\*2, // maximum number for rand()

SIZE17 = 17; //size of choices array

bool p1\_crrt, // player 1 correct

p2\_crrt; // player 2 correct

int indx1, // index for player 1's name

indx2, // index for player 2's name

maxGmes = FIVE, // number of games

nGmsLft, // number of games left

round = ZERO, // round

numPlay = TWO;

// create new pointer to Score structure

Score \*score = new Score;

score->nPlayrs = numPlay;

score->maxGmes = maxGmes;

score->ttlGams = ZERO;

// create a pointer to Player's structure and create array size of 2

score->player = new Player[score->nPlayrs];

// loop through number of players

for (int i = ZERO; i < score->nPlayrs; i++) {

// generate new random name for each player

if (i == ZERO) {

indx1 = rand()%SEVEN;

} else { //makes sure the same 2 names are NOT picked from the names array

do {

indx2 = rand() % SEVEN;

} while (indx2 == indx1);

indx1 = indx2;

}

// creating new char[] the size of string +1

score->player[i].name = new char[names[indx1].length() + 1];

// loop through the length of name string

for (int c = ZERO; c < names[indx1].length() + 1; c++) {

// convert string to c-string and to uppercase. static\_cast as char

score->player[i].name[c] = (char) toupper(names[indx1].c\_str()[c]);

}

// set each player's number of wins to zero

score->player[i].numWins = 0;

}

// pointer for rounds

int \*rndPtr = nullptr;

// read in maximum number of games that can be played from file

nGmsLft = score->maxGmes; // set numberOfGamesLeft to equal maxGames

while (nGmsLft>ZERO) {

// initial variables to represent the location of each player's ship

score->player[0].shipLoc = rand() % (ZERO+MAX); // random number between [1,14]

score->player[1].shipLoc = rand() % (ZERO+MAX);

// sets variables to default starting values

p1\_crrt = p2\_crrt = false;

// display game's introduction message

banner(score,"BATTLESHIP");

// loops until a player correctly guesses opponents ship location

while ((!p1\_crrt) && (!p2\_crrt)) {

// display round number banner

rndPtr = rBanner(round);

// Player 1's Guess

p1\_crrt = play(score,0,1,round);

// conditional only runs if player 1 misses player 2's ship

// Player 2's Guess

if(!p1\_crrt) {

p2\_crrt = play(score,1,0,round);

}

// if both players guess wrong, then display message

if ((!p1\_crrt) && (!p2\_crrt)) cout << endl << " You Both Missed. Try Again...\n\n";

} // ends while((!p1\_crrt) && (!p2\_crrt))

nGmsLft--;

score->ttlGams = nGmsLft;

prntScore(score);

// checks maximum number of games has NOT been played

if(nGmsLft>ZERO) pause();

// reset variables for next game

\*rndPtr = 0;

p1\_crrt = p2\_crrt = false;

} // ends while(nGmsLft> ZERO)

return score;

}

// returns if player's guess is correct or not

bool play(Score \*score,int a, int b, int &round){

int MAX = 20, // maximum number for rand()

SIZE17 = 17; //size of choices array

int maxGmes = 3, // number of games

pGuess, // player 1 guess

numWins, // counter for player 1 wins

ttlGmes, // sum of both players number of wins

ttlRnds = ZERO; // sum of total rounds played

// display instructions

cout << endl << setw(12) << " " << score->player[a].name << "\n Guess a number between "<<ZERO<<" and " << MAX-1 << endl;

// Generates random number guess between [1,14]

pGuess = rand()%(ZERO+MAX);

// checks if player guess is correct

if (pGuess == score->player[b].shipLoc) {

numWins = score->player[a].numWins+1; // increment player 1 number of wins

// set structure's member value for player

score->player[a].numWins = numWins;

score->player[a].isRight = true;

// calculate total number of games won & number rounds played

ttlGmes = score->player[a].numWins + score->player[b].numWins;

score->ttlGams = ttlGmes;

score->ttlRnds = ttlRnds + round; // sums the total number of rounds from all games

// display HIT message for player's correct guess

hitMiss(score, pGuess, b, true); // display MISS message for player 2's wrong guess

return true;

}

// if player1 guess is wrong display MISS message

score->player[a].isRight = false;

hitMiss(score, pGuess,b, false);

return false;

}

// display hit message when player guesses correctly

void hitMiss(Score \*score, int pGuess, int b, bool isHit) {

if (isHit) {

cout << setw(12) << pGuess << " == " << score->player[b].shipLoc << endl;

cout << setw(22) << "It\'s a HIT!\n" << endl;

} else {

cout << setw(15) << pGuess << endl << setw(22) << "It\'s a MISS!\n";

}

}

// banner displays round number

int \*rBanner(int &round) {

int \*rndPtr = nullptr;

round += 1;

rndPtr = &round;

cout << endl << setw(26) << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << setw(18) << "Round " << \*rndPtr << endl;

cout << setw(26) << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

return rndPtr;

}

// displays game's name and instructions in a banner

void banner(Score \*score, string title) {

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n" << setw(21) << title << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << setw(10) << setfill(' ') << score->player[0].name

<< setw(6) << setfill(' ') << right << "vs"

<< setw(12) << score->player[1].name << endl;

if (title == "SCOREBOARD") {

cout << setw(8) << score->player[0].numWins

<< setw(14) << score->player[1].numWins << endl;

} else {

cout << setw(2) << " " << "\n Try to guess the location of \n"

<< setw(6) << " " << "your opponent\'s ship." << endl;

}

}

// pause screen before game starts

void pause() {

cout << "\nPress enter to continue. \n\n";

cin.get();

}