Project I
STL
Battleship

Danielle F
CIS-17C-48784
November 19, 2023

Table of Contents

Introduction

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

Objective:

Guess the location of your opponent's ship.

Rules:

- Best out of three hits wins.
- Each player has 3 ships 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 hit, the scoreboard will show how many hits each player currently 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 three and the game automatically generates each player's guess and ship's location once the user presses "Enter".

GitHub Repository

https://github.com/koa2019/cis17c_fall2023_2/tree/main/projects

Development Summary

Battleship, MySTL, Board, User Classes	
Lines of code	1965
Comment lines	444
Blank lines	269
Total lines of source file	**1252**

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

My initial strategy for this project was to convert my CIS-5 Project 2 Battleship using STL (Standard Library Template) containers and various algorithms. This version showcased the game very well, it had functions, and the game was played in main.cpp, so I thought it would be a good starting point to begin converting my arrays to STL containers.

My first challenge appeared quickly. My main.cpp was extremely crowded which made things a little hectic. My solution was to create a new program called convertArrays_to_STL_v1. A clean slate allowed me to copy blocks of working code from Battleship to my new main function, so I could convert the various arrays to containers one at a time and without breaking the game.

My second dilemma came when I realized I needed my STL concepts in classes and not functions which meant my game needed to be in classes as well. Converting my STL concepts into a single class wasn't the difficult part. The problem was I needed to move the Battleship code from main into functions. I wanted to move it to functions and call them in main before I moved them to classes. This was tedious because I had a two-player game which meant I had four two-dimensional arrays and various variables used as flags to control the flow of the game. I had to choose between passing them as reference variables or making the functions return an array. I chose to pass them by reference and later that ended up making more work for me when I moved it to classes. This ended up taking more time and effort than I had expected.

My solution was to change my strategy. The project required STL concepts to be applied to a game and have a working game. I decided to combine my CIS-17B Group Project 2 Yahtzee game and my independent STL class. Yahtzee is a refined and polished program that aggregates Admin, User, Yahtzee, Scorecard, and Dice classes. I was unsure if I should use a group project, but my partner from that project only wrote the code for Scorecard and Dice class and I was planning on replacing it with Battleship. This code works with menus, and you can run it. Please see cis17c_project_1_Battleship_combineYahtzeeMySTLClass_v6.

My third problem came when my local branch wouldn't push to my Github anymore. This was not my first run in with Git, so I decided to save myself time and stress by creating a new repository and copying the folders.

I was able to move my Battleship into classes and working. I added Battleship and Board classes to a new version and got it to work with my Admin, and MySTL classes. This code works with menus, and you can run it. Please see cis17c_project_1_Battleship_addBattleship_v7.

My current dilemma is that I have two games (Yahtzee and Battleship) in classes running independently of MySTL class, and I'm finally ready to convert one of using STL containers. Yahtzee uses two 1-dimensional arrays and Battleship uses four 2-dimensional arrays.

Versions Timelines

1. **convertArrayToSTL_v1 thru v2b**
    a. Converted string arrays (names and p2names) to unordered set containers
    b. Changed map top Player pair to (set<string>,list<float>), so I could find a name
    c. Top Player(pair("name", hi Score)
2. **battleship_v1_BROKEN**
    a. I unsuccessfully tried combining my game and my STL functions together.
        i. The code is based off my cis5_project_2_battleship_v6 code.
        ii. Added convertArrayToSTL_v2b and it broke it.
3. **battleship_v2**
    a. I copied cis5_project_2_battleship _v6 code again.
    b. I ran the original code to make sure the game worked and then I commented everything out and added convertArrayToSTL_v2b.
    c. It didn't break the code this time.
    d. Used std::transform() when a user inputs their name.
    e. Converted scores integer array to a std::set.

   f. Created a partially filled std::list<float>, and then copied the set into the list using std::copy() and a std::insert iterator.

4. **battleship_v4**
   a. Moved all of the STL code that was in the main function to void concepts(), so I could confirm it worked before I moved it to a class.
5. **battleship_v5**
   a. I tried initializing a stack with a list and tried accessing the link's pointer member. It didn't work. Apparently, you can't.
6. **convertArrayToSTL_v5**
   a. This project needs to be in classes, but the problem with cis5_project_2_battleship is that the game runs from the main function and it doesn't have any classes.
   b. Removed all the STL functions from battleship_v4 to this program because I need to add these cis17c concepts to one of my projects that is already in classes and working.
   c. I tried reverse stack with reverse() and recursive insert_at_bottom() but couldn't get it to work.
7. **convertArrays_to_STL_v6**
   a. Problem: Couldn't get ostream to work in with a list or a vector or with sstream.
8. **convertArrays_to_STL_v7**
   a. Added MySTL class
9. **battleship_v6_addYahtzeeMySTL**
   a. I need the game to be in classes, so I combined my cis17b_yahtzee AND convertArrays_to_STL_v8.
   b. Made User destructor virtual
   c. Added menu2 function that asks user if they want to view CIS 17C concepts applied to project or if they want to play the game.
10. **cis5_project_2_battleship_v7**
   a. I was finally able to get this version into classes.
   b. Nightmare trying move all of the code in main to functions and then into a class.
   c. Board class creates and prints 2 static char 2D arrays for each player: board1, board2.
   d. Moved everything in main to Battleship class. I commented everything out and will uncomment one section at a time.
   e. My CIS-5 game had great parts to it, but they were lost during CIS-17A when I had to introduce classes. I had no prior experience with classes, so the quality of my game deteriorated.
11. **cis17a_project_2_v7 thru v9**
   a. New strategy implemented because I'm running out of time. I'm taking this version of Battleship and adding
   b. Replaced Scores class with Battleship class works.
   c. Add MySTL class to run independtly.
   d. Converted MySTL functions to Battleship game in:
      i. Board.h
      ii. Player.h
      iii. TopPlyrsBrd.h
12. **battleship_v7_addBattleship_BROKEN**

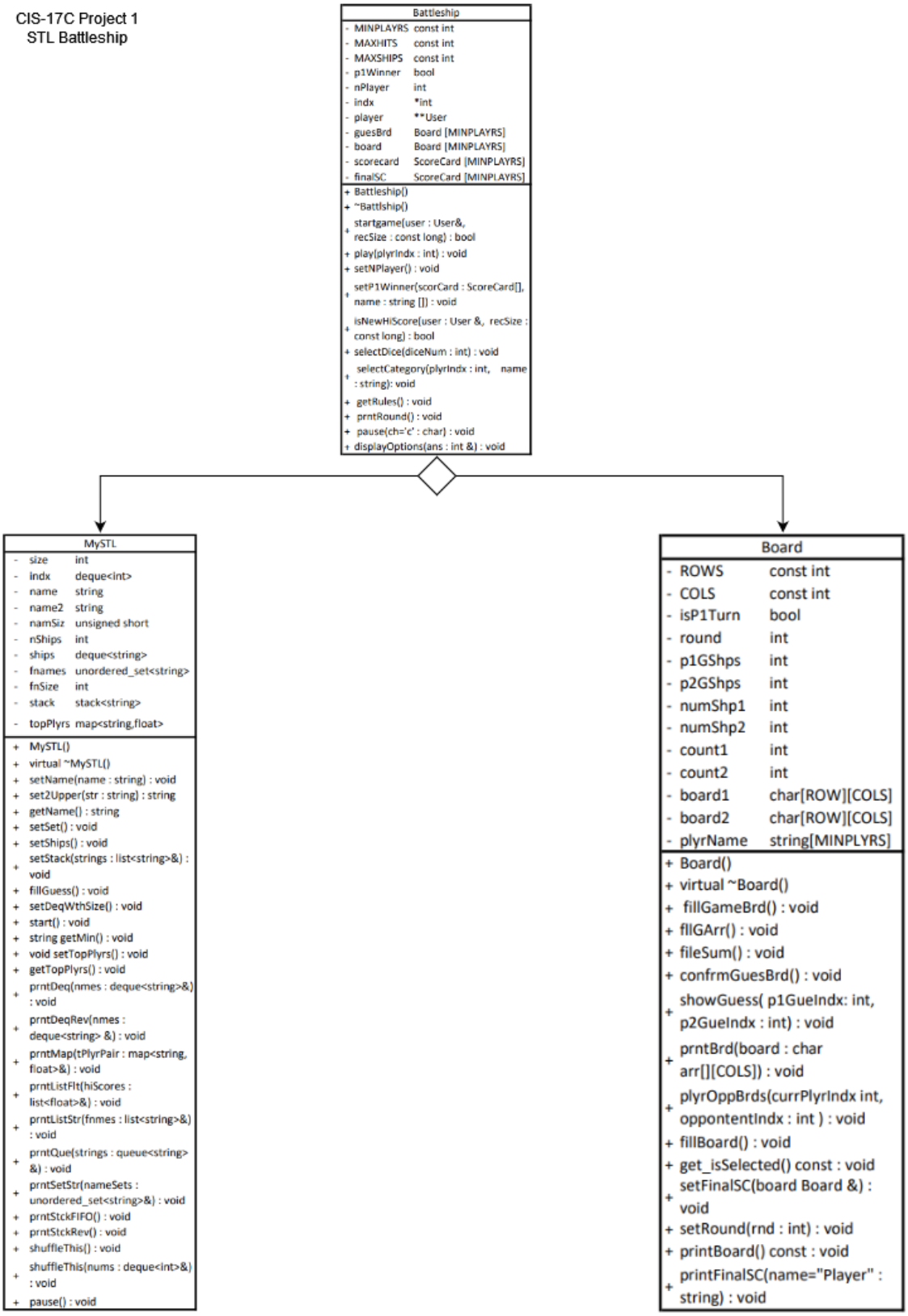
a. I used my CIS-17B Yahtzee class as a template for Battleship
b. The Battleship class has access to User and Admin classes.

13. battleship_v8

a. Copied cis17a_project_2_v9.
b. I added a menu to allow the user to run the STL concepts or the game.
c. The game works with the STL concepts implemented in them.
d. MySTL class runs independently of the game to show they work in one spot.

UML

CIS-17C Project 1
STL Battleship



Sample STL Outputs:

Sample STL Outputs

```
Player 1: Enter your name
Danielle  -> DANIELLE.
Used transform(std::toupper)
```

```
Push List into Stack.
List of names
1. MARTY
2. BART
3. ALEX
4. CHOPPER
5. MAUL
6. ANAKIN
7. KENOBI

Stack (Last In First Out)
1. KENOBI
2. ANAKIN
3. MAUL
4. CHOPPER
5. ALEX
6. BART
7. MARTY

Reverse Order Stack.
1. MARTY
2. BART
3. ALEX
4. CHOPPER
5. MAUL
6. ANAKIN
7. KENOBI
```

```
Locating your opponent...
min(KENOBI,MARTY) = MARTY

DANIELLE vs MARTY!
```

```
      Queue Container
Each player has 3 ships that can be hit.
ship1
ship2
ship3
```

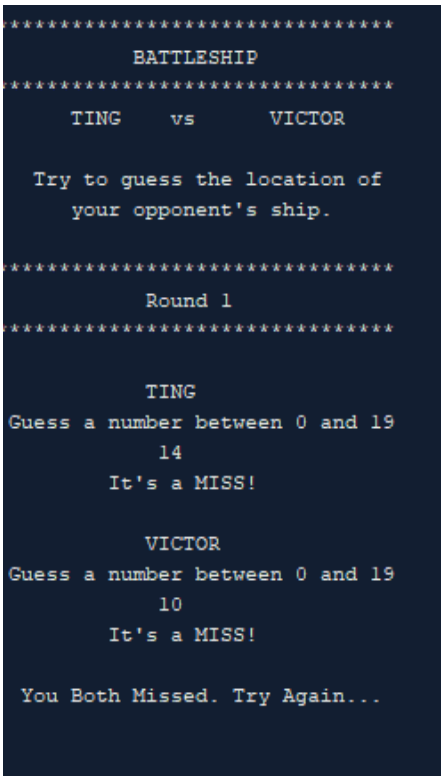
```
Set:
97.43
98.81
98.98
99.69

Partial List:
1. 101.9
2. 99.71
3. 99.24
4. 99.18

Set copied into List:
1. 101.9
2. 97.43
3. 98.81
4. 98.98
5. 99.69
6. 99.71
7. 99.24
8. 99.18

Map Sorted Alphabetically:
Map pair(set<string>, list<float>)
Map Top Player's Scores
1. (ALEX    ,99.71)
2. (ANAKIN  ,98.81)
3. (BART    ,99.24)
4. (CHOPPER ,99.69)
5. (DANIELLE,101.9)
6. (KENOBI  ,97.43)
7. (MARTY   ,99.18)
8. (MAUL    ,98.98)


Congratulate DANIELLE for being
this week's winner
with 101.9 points!
```

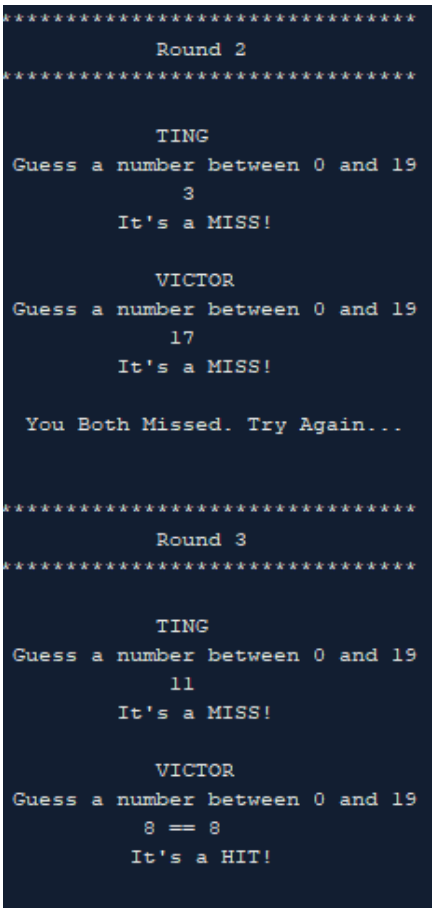
```
Map pair(set<string>name, list<float>scores).
Map Sorted Alphabetically:
Map Top Player's Scores
1. (ALEX    ,99.71)
2. (BART    ,98.98)
3. (CHRIS   ,99.69)
4. (GABE    ,99.24)
5. (JILLIAN ,99.18)
6. (MARTY   ,97.43)
7. (SANTA   ,101.9)
8. (VICTOR  ,98.81)


Enter a player's name to return
what place they're in this week.
bart

BART is in the 2 spot for this week's top player
with 98.98 points.
```

```
Enter a player's name to return
what place they're in this week.
kenobi

KENOBI is in the 6 spot
for this week's top player
with 97.43 points.
```

Sample Inputs: Enter, Enter, Enter, Enter, Enter

Sample Outputs:



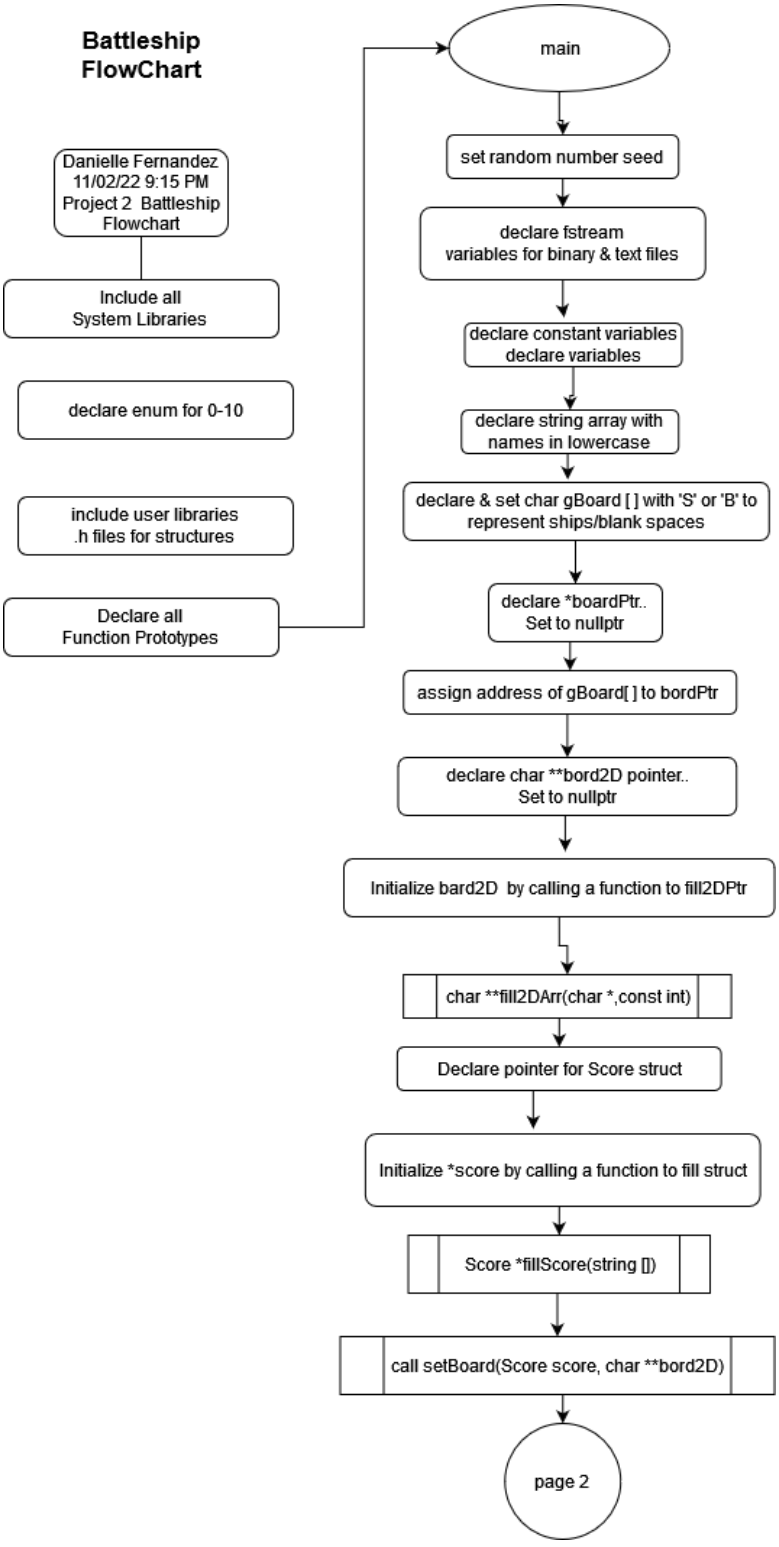


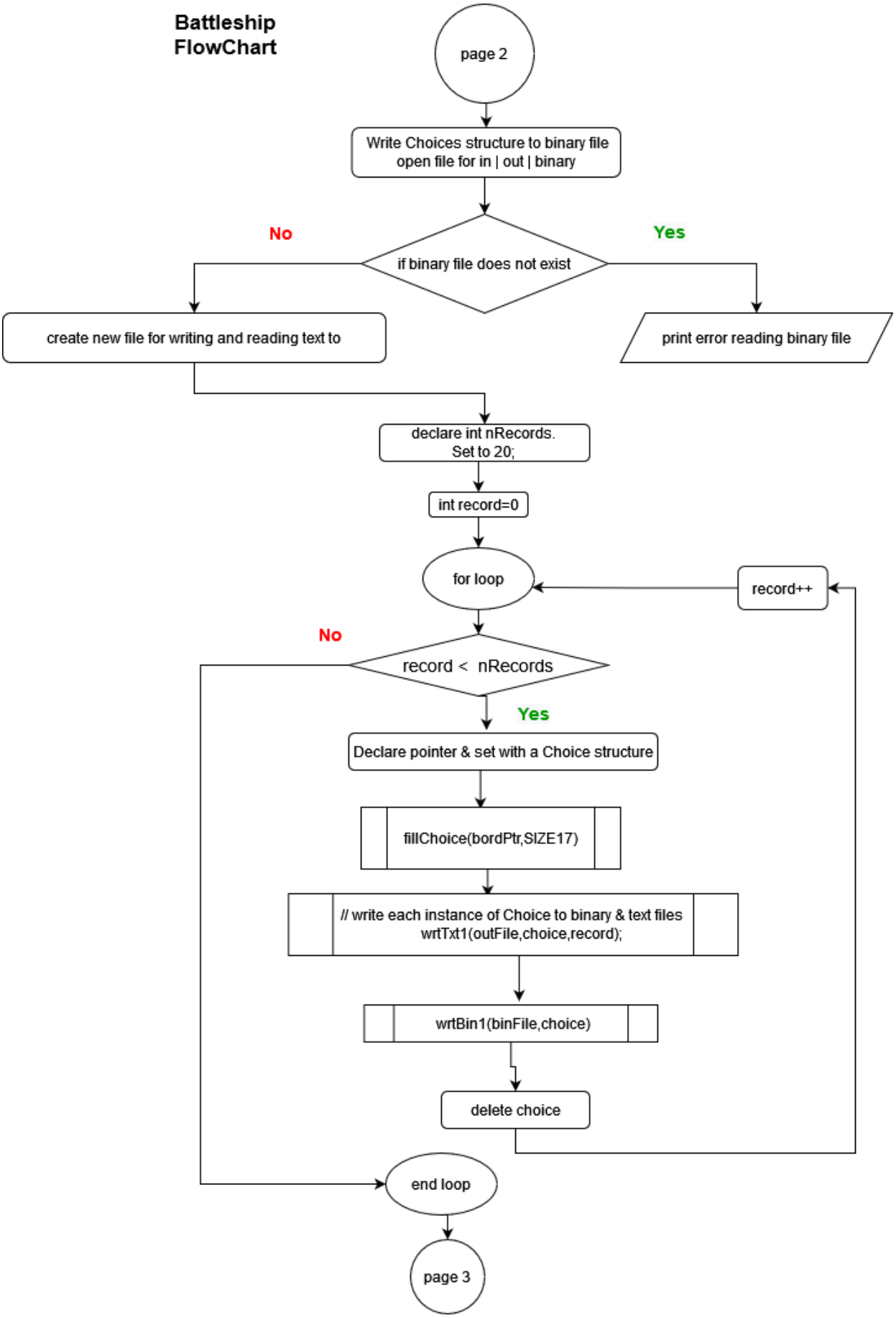
Checklist

CIS 17C Fall 2023 Project 1 Checklist			
Instructions:			
	Note: no vectors. Show as many concepts as possible, meaning all the below. Especially Algorithm/Iterators/Containers in the STL as possible for the game.		
Github Repo:		https://github.com/koa2019/cis17c_fall2023_2	
	Checklist		**Location in code/comments**
	X	Minimum 750 lines of code	
1. Container classes			
	1. Sequence (At least 1)		**See MySTL.cpp**
	X	list	168, 243
	☐	forward_list	
	☐	bit_vector	
	2. Associative Containers (At least 2)		**See MySTL.cpp**
	X	set	163,
	X	map	186
	☐	hash	
	3. Container adaptors (At least 2)		**See MySTL.cpp**
	X	stack	290
	X	queue	265, 463
	☐	priority_queue	
2. Iterators			
	1. Concepts (Describe the iterators utilized for each Container)		**See MySTL.cpp**
	X	Trivial Iterator	320. int*, const int*, vector
	X	Input Iterator (get)	213, 286, 320, 344. Refinement of Trivial Itr
	X	Output Iterator (set)	300, 311, 174. list, set, map,
	X	Forward Iterator	177, 178, list. Refinement of Trivial Itr
	X	Bidirectional Iterator	172, 244. Default itr for list. Refinement of Fwrd Itr
	X	Random Access Iterator	313, 263. Default itr for vectors, deques. Refinement of Bidirectional
3. Algorithms (Choose at least 1 from each category)			
	1. Non-mutating algorithms		**See MySTL.cpp**
	☐	for_each	
	X	find	139
	☐	count	
	☐	equal	
	☐	search	
	2. Mutating algorithms		**See MySTL.cpp**
	X	copy	173, 312
	☐	swap	
	X	transform	427
	☐	replace	
	☐	fill	
	☐	remove	
	X	shuffle	477, 452
	3. Organization		**See MySTL.cpp**
	☐	sort	
	☐	binary search	
	☐	merge	
	☐	inplace_merge	
	X	minimum and maximum	221 min, 121 max

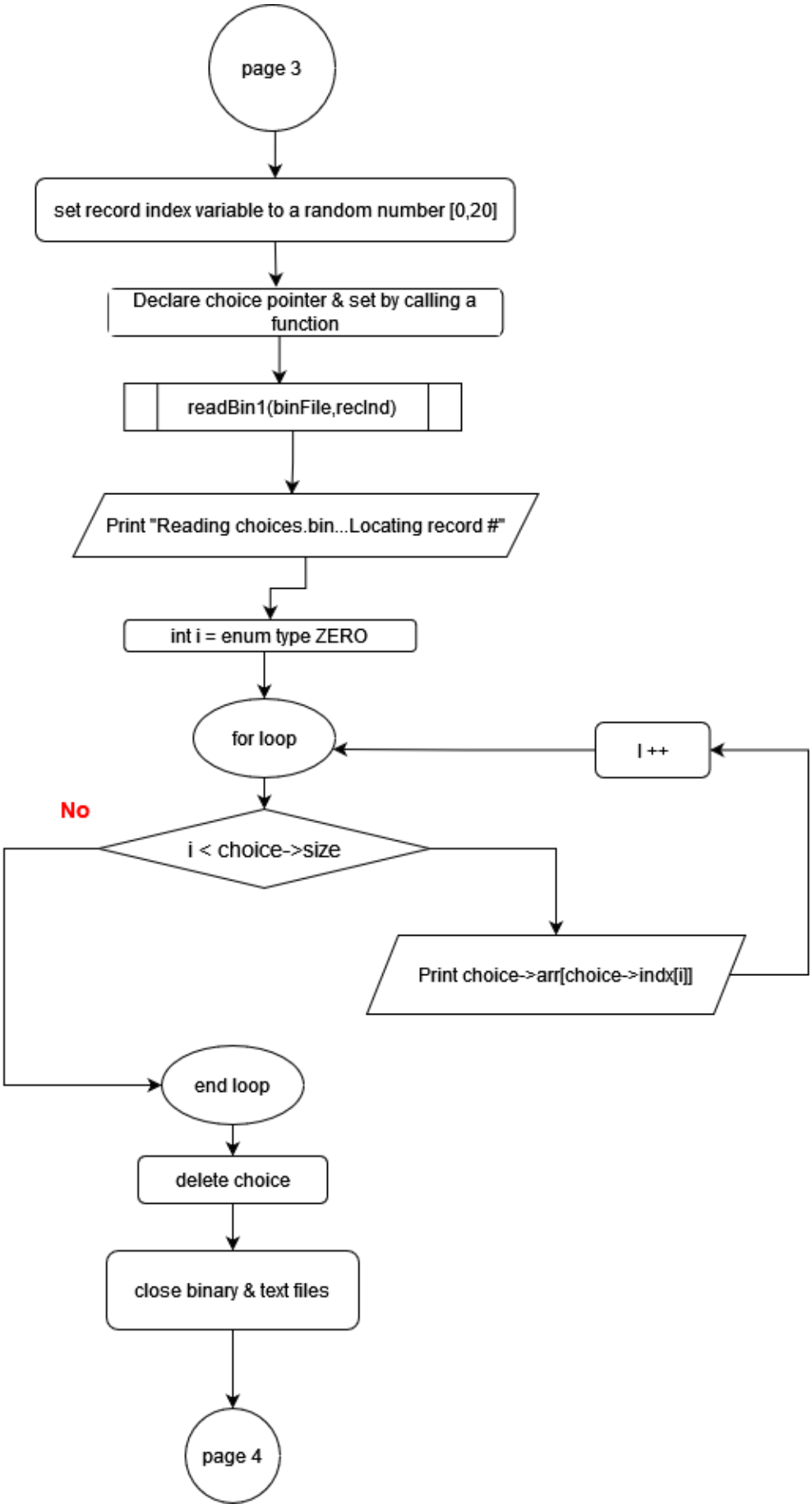
Flowchart

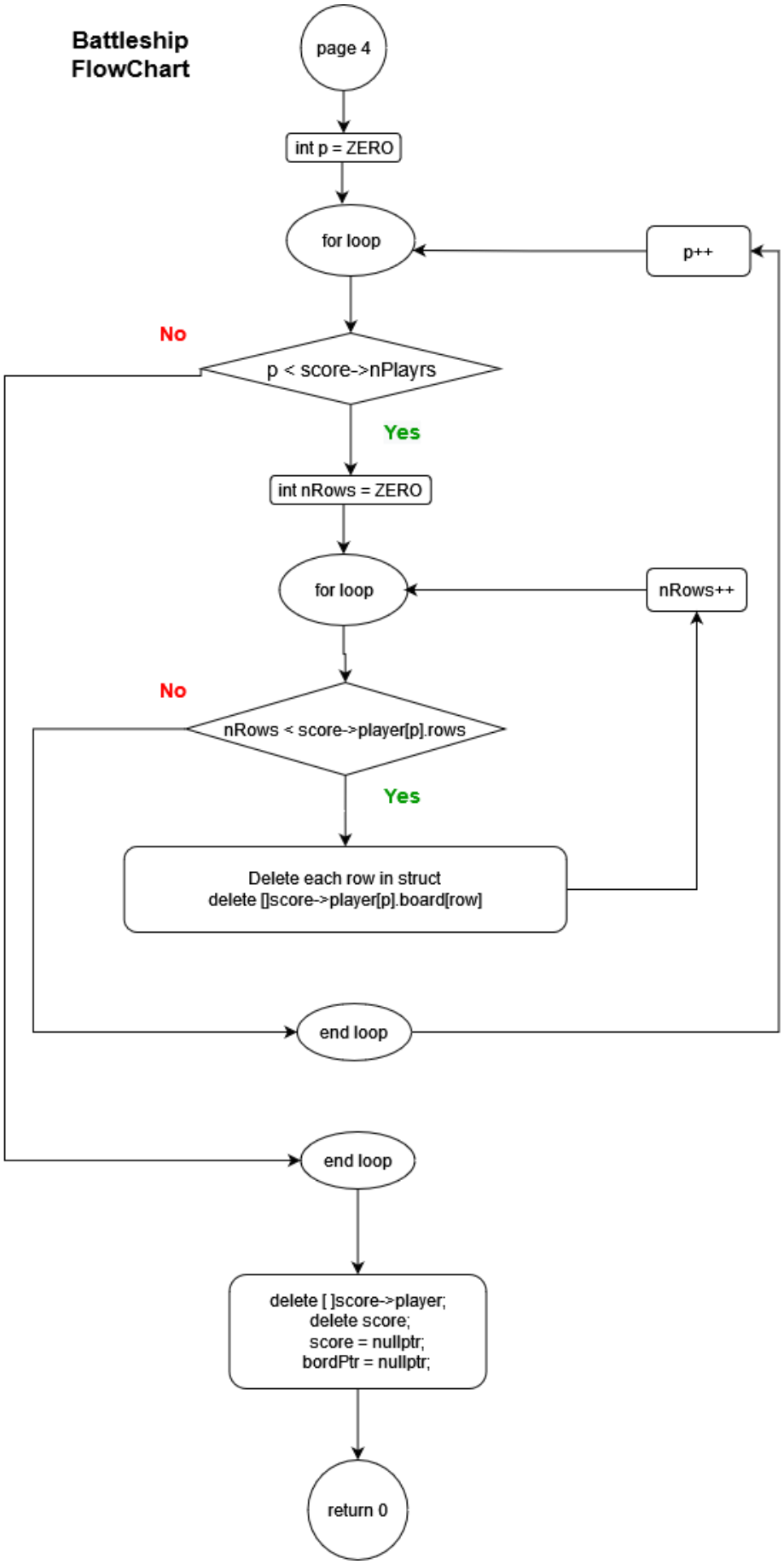
I have included the main function's flowchart below. Please see the Documents folder for the complete flowchart that illustrates each function's flowchart.


Battleship
FlowChart
Danielle Fernandez
11/02/22 9:15 PM
Project 2 Battleship
Flowchart
Include all
System Libraries
declare enum for 0-10
include user libraries
.h files for structures
Declare all
Function Prototypes
main
set random number seed
declare fstream
variables for binary & text files
declare constant variables
declare variables
declare string array with
names in lowercase
declare & set char gBoard [ ] with 'S' or 'B' to
represent ships/blank spaces
declare *boardPtr..
Set to nullptr
assign address of gBoard[ ] to bordPtr
declare char **bord2D pointer..
Set to nullptr
Initialize bard2D by calling a function to fill2DPtr
char **fill2DArr(char *,const int)
Declare pointer for Score struct
Initialize *score by calling a function to fill struct
Score *fillScore(string [])
call setBoard(Score score, char **bord2D)
page 2


Battleship
FlowChart
page 2
Write Choices structure to binary file
open file for in | out | binary
No
if binary file does not exist
Yes
create new file for writing and reading text to
print error reading binary file
declare int nRecords.
Set to 20;
int record=0
for loop
record++
No
record < nRecords
Yes
Declare pointer & set with a Choice structure
fillChoice(bordPtr,SIZE17)
// write each instance of Choice to binary & text files
wrtTxt1(outFile,choice,record);
wrtBin1(binFile,choice)
delete choice
end loop
page 3

Battleship FlowChart




Battleship
FlowChart
page 4
int p = ZERO
for loop
p++
No
p < score->nPlayrs
Yes
int nRows = ZERO
for loop
nRows++
No
nRows < score->player[p].rows
Yes
Delete each row in struct
delete []score->player[p].board[row]
end loop
end loop
delete [ ]score->player;
delete score;
score = nullptr;
bordPtr = nullptr;
return 0

Pseudo Code

- 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( )

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();
}
```