Jason Handen

Dr. Lehr

CSC-17A (43396)

5 June 2021

**Project 2: Baccarat (w/ Object Orientation) Documentation**

Introduction:The casino game Baccarat is a card game in which the player has little agency, aside from his initial bet type and amount. At its most basic, the goal is to bet on one hand scoring a higher sum of drawn cards than the other hand. Aces are worth one point, pip cards 2 through 9 are worth face-value, and 10 and face cards are worth zero. If a sum exceeds 9, the value in the tens place is discarded, such that a sum of 11 is equal to 1. If the player scores less than six for a sum of his first two cards, he will draw a third. The banker will draw according to a more complex set of conditions, which depend on the banker’s sum, the draw condition of the player, and the value of the player’s drawn card. If either the player or banker have a sum of eight or nine in the first two cards, this is called a “natural” and all play for that round stops there, with the recipient of the natural as the automatic winner.

Summary:

*Project Size:* 1081 lines, including classes, given whitespace formatting.

*Number of Variables:* about 40, including member variables.

*Number of Functions:* about 70, including class implementations.

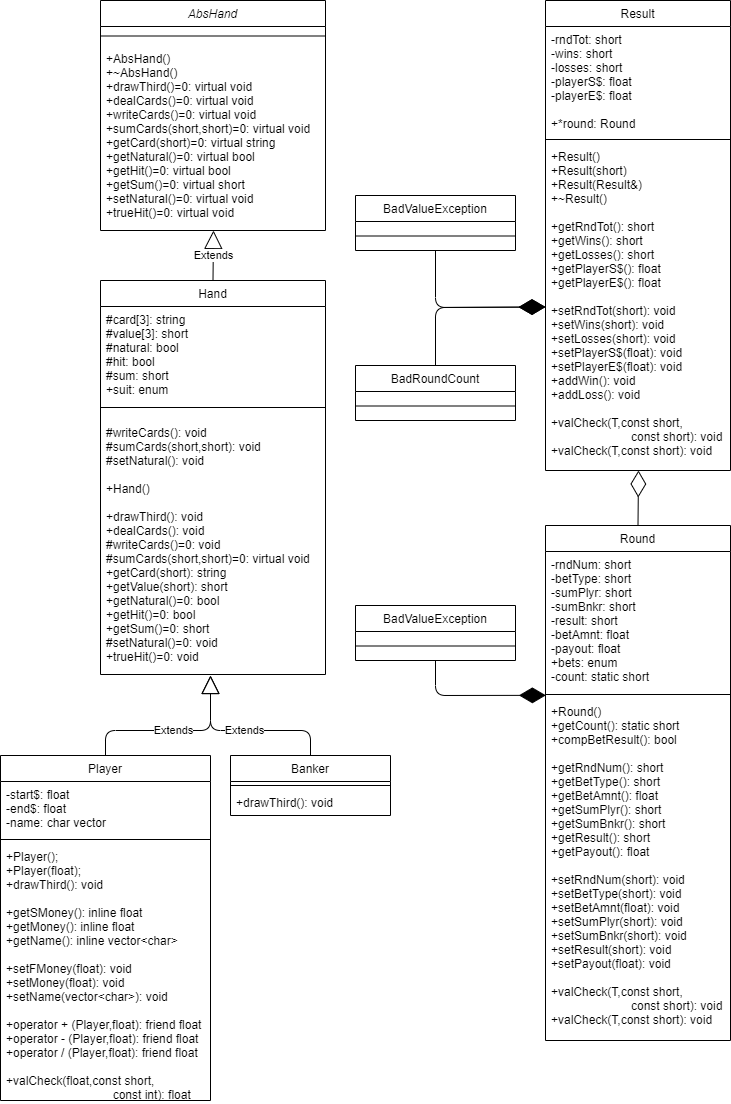
This C++ application emulates the casino game Baccarat to a greater extent than the first iteration. It now allows for name entry and storage to a file. The user can still choose a number of rounds to play, choose to bet on the “Player,” “Banker,” or on a “Tie,” and now, a monetary betting system has been implemented. This system allows the player to choose an amount of money with which to start, make individual bets each round, and compares his total from the beginning and the end of the game. Much of this information is now stored in the binary files that were previously used, such that they can be printed at the start of the application. The game will proceed according to the rules of Baccarat, and at the conclusion of each round, information about that round is displayed. The user may then choose how he would like to bet, and how much, for the next round, until the chosen number of rounds is exhausted. At that point, summary statistics for the game are displayed and stored in two binary files, and the program terminates. Sidenote: If he loses all of his money, he will be forced to continue to play the game, while being unable to bet any more money, as punishment.

The project was quite difficult, but it helped to have a functional program from which to work. There was a significant challenge in converting a once functional 400-line program into a greater-than 1000 line behemoth with the magic of object orientation, but with time, it came together. Simply converting the program and reimplementing the functionality through classes and the like took about three days and doubled the overall length. It took about two more days to finalize the program, having included each of the concepts from chapters 13 through 16. I spent a good couple of hours this time around tracking down a runtime error that turned out to be just a poorly-written destructor. Various internet resources were helpful in reviewing the exact syntax (and in some cases, the concepts) that I needed to use in this final program.

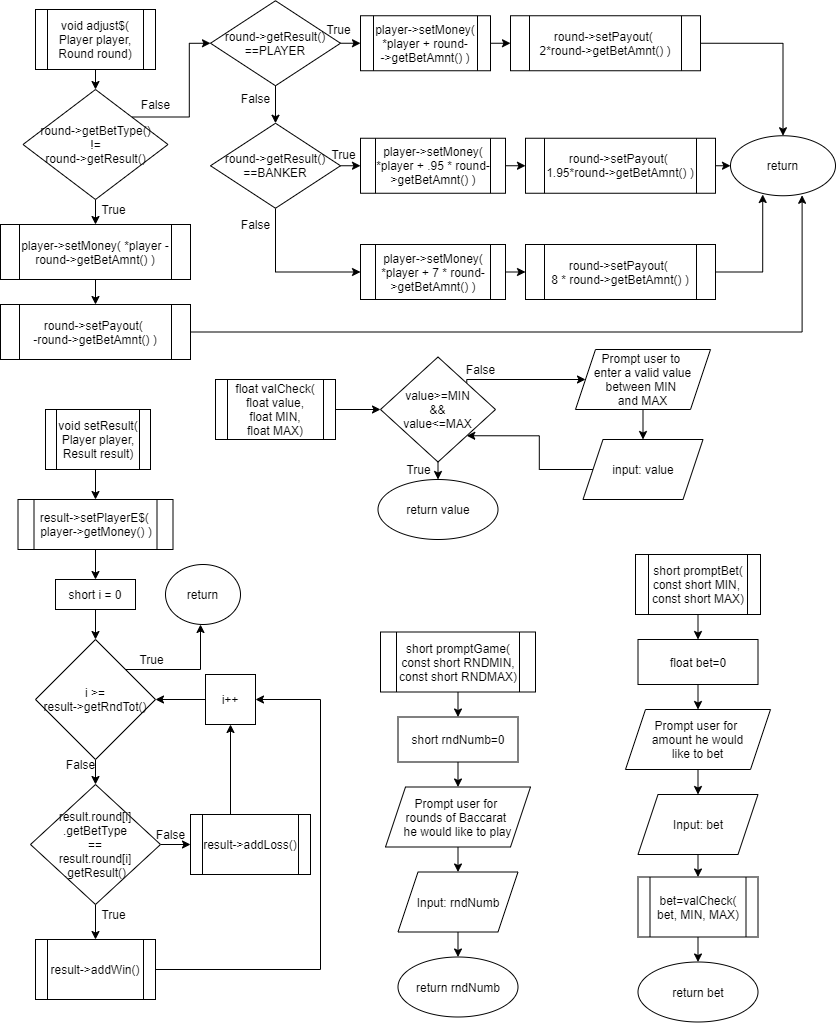
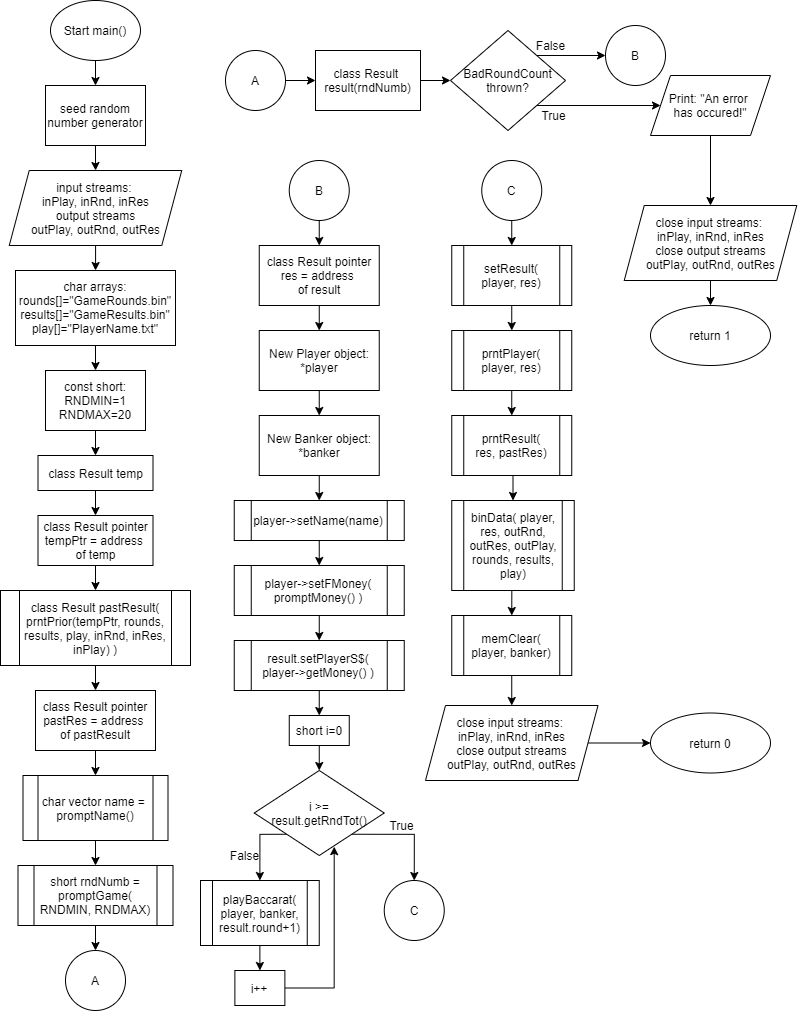
Overall, again I would say I am pleased with the product of my efforts. Given the choice, I think that this would be a good style of final project to complete in the future.

Description:The main point of this program was to implement Chapter 13 through 16 concepts from the Gaddis textbook in the form of a card game, with a focus on object orientation.

Class UML:



Flowchart examples:



Major Variables:

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable Name** | **Description** | **Location** |
| Vector<char> | Name | Char vector to store player name | Player.h |
|  | Name | Char vector to store player name | Main.cpp |
| Enum | Suit | Holds values for suit comparison | Hand.h |
|  | Bets | Holds values for bet entry comparison | Main.cpp |
| Constant Short Int | RNDMIN | Round minimum for user entry | Main.cpp |
|  | RNDMAX | Round maximum for user entry | Main.cpp |
|  | WIDTH | Hold value for setw formatting | Main.cpp |
| Static Short Int | Count | Keeps a count of the round, for rndNum assignment | Round.cpp |
| Short Int | Value[3] | Holds three values for given hand | Hand.h |
|  | Sum | Holds sum of cards currently in play | Hand.h |
|  | rndTot | Holds value for total number of rounds | Result.h |
|  | Wins | Holds number of wins | Result.h |
|  | Losses | Holds number of losses | Result.h |
|  | rndNum | Round number for this object | Round.h |
|  | betType | Type of bet the player made | Round.h |
|  | sumPlyr | Sum of player’s cards | Round.h |
|  | sumBnkr | Sum of banker’s cards | Round.h |
|  | rndNum | Total round number to be used for object instantiation | Main.cpp |
|  | Bet | Bet input to be assigned | Main.cpp |
| Boolean | Natural | Determines whether “natural” conditions have been met | Hand.h |
|  | Hit | Determines whether “hit” conditions have been met | Hand.h |
| Float | Start$ | Money with which the player starts | Player.h |
|  | End$ | Money with which the player ends | Player.h |
|  | playerS$ | Money with which the player starts | Result.h |
|  | playerE$ | Money with which the player ends | Result.h |
|  | betAmnt | Amount of money the player bets | Round.h |
|  | Payout | Amount of money the player gets back | Round.h |
|  | Money | Amount of money for player assignment | Main.cpp |
| String | Card[3] | Holds the names of the three cards for a given hand | Hand.h |
| C-string | Rounds[] | File name for GameRounds access | Main.cpp |
|  | Results[] | File name the GameResults access | Main.cpp |
|  | Play[] | File name for PlayerName access | Main.cpp |

C++ Course Concepts:

|  |  |  |  |
| --- | --- | --- | --- |
| **Chapter** | **Section** | **Topic** | **Line #''s** |
| 13 |  | Classes |  |
|  | 1 to 3 | Instance of a Class | main.cpp Ln 66 |
|  | 4 | Private Data Members | Player.h Ln 17-19 |
|  | 5 | Specification vs. Implementation | Hand.h + Hand.cpp |
|  | 6 | Inline | Player.h Ln 25-33 |
|  | 7, 8, 10 | Constructors | Result.cpp Ln 13-38 |
|  | 9 | Destructors | Result.cpp Ln 40-42 |
|  | 12 | Arrays of Objects | Result.cpp Ln 23 |
|  | 16 | UML | See above |
|  |  |  |  |
| 14 |  | More about Classes |  |
|  | 1 | Static | Round.cpp Ln 10 + 16-19 |
|  | 2 | Friends | Player.h Ln 38-40 |
|  | 4 | Copy Constructors | Result.h Ln 27 |
|  | 5 | Operator Overloading | Player.cpp Ln 37-48 |
|  | 7 | Aggregation | Result.h Ln 21 |
|  |  |  |  |
| 15 |  | Inheritance |  |
|  | 1 | Protected members | Hand.h Ln 18-27 |
|  | 2 to 5 | Base Class to Derived | Hand.h to Player.h |
|  | 6 | Polymorphic associations | main.cpp Ln 252-255 |
|  | 7 | Abstract Classes | AbsHand.h |
|  |  |  |  |
| 16 |  | Advanced Classes |  |
|  | 1 | Exceptions | Result.h Ln 22-23 |
|  | 2 to 4 | Templates | Result.cpp Ln 89-100 |
|  | 5 | STL | Player.h Ln 19 |

Main.cpp Pseudocode:

*Enumerated type, User Bets:*

*PLAYER=1, BANKER=2, TIE=3*

*Start main:*

*Seed random number generator, time(0)*

*Declare file streams:*

*in-streams: input for Rounds, input for Results, input for Player*

*out-streams: output for Rounds, output for Results, output for Player*

*Create C-strings:*

*rounds[] = “GameRounds.bin”*

*results[] = “GameResults.bin”*

*play[] = “PlayerName.txt”*

*Constant Integers:*

*ROUND MINIMUM (RNDMIN) = 1*

*ROUND MAXIMUM (RNDMAX) = 20*

*Create Result object: temp*

*Result pointer tempPtr assigned to temp address*

*Result object pastResult instantiated with:*

*Call function: PrintPrior(tempPtr, rounds, results, Rounds input, Results input, Player*

*input)*

*Result pointer pastRes assigned to pastResult address*

*Character vector name instantiated with:*

*Call function: promptName()*

*Short integer rndNum = Call function: PromptGame(RNDMIN, RNDMAX)*

*\*\*\*Start Try Block*

*Result object result instantiated with: rndNum*

*Result pointer res assigned to result address*

*Allocate memory for new Player object: player*

*Allocate memory for new Banker object: banker*

*Call function: player->setName(name)*

*Call function: player->setFMoney ( Call function:promptMoney() )*

*Call function: result->setPlayerS$( player->getMoney() )*

*For( i=0, i less than round number, increment i )*

*Call function: PlayBaccarat(Round array of i)*

*Call function: SetResult(player pointer, result pointer)*

*Call function: prntPlayer(player pointer, result pointer)*

*Call function: PrintResult(result pointer, pastResult pointer)*

*Call function: BinaryData(player pointer, result pointer, Rounds output, Results output,*

*Player output, rounds c-string, results c-string, play c-string)*

*Call function: MemoryClear(Round array, Result structure)*

*\*\*\*End Try Block*

*\*\*\*Start Catch Block*

*Output: “Error: Invalid round count entered. Exiting program…”*

*Close file streams: Rounds input, Results input, Player input, Rounds output, Results output*

*Player output)*

*Return 1*

*\*\*\*End Catch Block*

*Close file streams: Rounds input, Results input, Player input, Rounds output, Results output*

*Player output)*

*Return 0*

*End main*

*Start PrintPrior*

*Receives:*

*Result pointer: temp*

*c-string: rounds, results, player*

*input file stream: input for Rounds, input for Results, input for Player*

*Integer response = 0*

*Player object: player*

*Banker object: banker*

*Input for Player: open c-string:player file, in text mode*

*Input for Rounds: open c-string:rounds file, in binary mode*

*Input for Results: open c-string:results file, in binary mode*

*If cannot open c-string:rounds file for input:*

*Print: “No Round file to open. This must be the first run!”*

*Else if cannot open c-string:player file for input:*

*Print: “No Player file to open. This must be the first run!”*

*Else if cannot open c-string:results file for input:*

*Print: “No Results file to open. This must be the first run!”*

*Else*

*Create temp variables for result members*

*Take input from text file c-string:player*

*Concatinate string while input continues*

*Take input from binary file c-string:results*

*Store in Result object: Round total, Wins, Losses, Starting money, Ending money*

*Print Rounds, Wins, Losses, Starting money, Ending money*

*Do:*

*Move read cursor to beginning, Input for Round*

*Prompt user for round to read: response*

*response = call function: ValueCheck(response, 0, Round total)*

*For( i=0, i less than response, increment i)*

*Take temporary input for rndNum, betType, res, sumB, sumP, payout*

*If response does not equal 0*

*Assign temp variables to temp object round member*

*Print rndNum, betType, res, sumB, sumP, payout*

*While response does not equal 0*

*Return pointer to temp object*

*End PrintPrior*

*Start PromptName*

*String: temp*

*New char vector: name*

*Prompt user for name*

*Getline to temp string*

*For size of temp, assign temp element to vector with pushback function*

*Greet the user*

*Return name vector*

*End PromptName*

*Start PromptGame*

*Receives:*

*Constant integers: RNDMIN, RNDMAX*

*Integer rndNum = 0*

*Prompt user for number of games to play*

*Take input to rndNum*

*Return rndNum*

*End PromptGame*

*Start PlayBaccarat*

*Receives:*

*Player pointer: player*

*Banker pointer: banker*

*Round pointer: round*

*Call function: round->setBetType( call function: promptBet(PLAYER,TIE) )*

*Print amount of money the player currently has*

*Call function round->setBetAmnt( call function: promptBetAmnt( call function:*

*player->getMoney() ) )*

*Hand object array: load player and banker*

*Use for loop to polymorphically call function: dealCards()*

*If round:player:sum less than 6 and both player and banker naturals are false*

*Call function: player->drawThird()*

*If round:player:natural is false and round:banker:natural is false*

*If round:player:hit is false and round:banker:sum less than 6*

*Call function: banker->trueHit()*

*Else if round:player:hit is false and round:banker:sum greater than 5*

*// banker stands*

*Else if round:banker:sum less than 3*

*Call function: banker->trueHit()*

*Else if round:banker:sum is 3 and round:player:value of 2 not equal 8*

*Call function: banker->trueHit()*

*Else if round:banker:sum is 4 and round:player:value of 2 greater than 1, less than 8*

*Call function: banker->trueHit()*

*Else if round:banker:sum is 5 and round:player:value of 2 greater than 3, less than 8*

*Call function: banker->trueHit()*

*Else if round:banker:sum is 6 and round:player:value of 2 greater than 5, less than 8*

*Call function: banker->trueHit()*

*If player getSum() greater than banker getSum()*

*Round setResult( PLAYER )*

*Else if player getSum() less than banker getSum()*

*Round setResult ( BANKER )*

*Else Round setResult ( TIE )*

*Round setSumPlyr (player getSum() )*

*Round setSumBnkr (banker getSum() )*

*Call function: adjust$(player, round)*

*Call function: PrintRound(player, banker, round)*

*End PlayBaccarat*

*Start PromptBet*

*Receives:*

*Constant integers: MIN, MAX*

*Integer bet*

*Prompt user to place bet*

*Take input to bet variable*

*Bet = call function: ValueCheck(bet, MIN, MAX)*

*Return bet*

*End PromptBet*

*Start PromptBetAmnt*

*Receives:*

*Constant float MAX*

*Float bet*

*Prompt user for bet amount*

*Bet = call function: ValueCheck(bet, 0.00, MAX)*

*Return bet*

*End PromptBetAmnt*

*Start adjust$*

*Receives:*

*Player pointer: player*

*Round pointer: round*

*If round getBetType() and round getResult() don’t match*

*Player setMoney( player current money– round getBetAmnt() )*

*Else*

*If round getResult() == PLAYER*

*Player setMoney( player current money + round getBetAmnt() )*

*setPayout( 2\*getBetAmnt() )*

*Else if round getResult() == BANKER*

*Player setMoney( player current money + round getBetAmnt() \* .95)*

*setPayout( 1.95\*getBetAmnt() )*

*Else*

*Player setMoney( player current money + round getBetAmnt() \* 7)*

*setPayout( 8\*getBetAmnt() )*

*End adjust$*

*Start PrintRound*

*Receives:*

*Player pointer: player*

*Banker pointer: banker*

*Round pointer: round*

*Start format player and banker columns*

*Print cards 0 and 1*

*If player getHit() is true, print player card 2*

*If banker getHit() is true, print banker card 2*

*Print sums for player and banker*

*Print WINNER below greater sum*

*If Bet type equals Result*

*Print “Won bet!”*

*Else if Bet type not equal Result*

*Print “Lost bet!”*

*Print round payout()*

*Print player current money*

*Print “Press enter to continue…”*

*End PrintRound*

*Start SetResult*

*Receives:*

*Player pointer: player*

*Result pointer: result*

*Result setPlayerE$( player getMoney() )*

*For( i=0, i less than result getRndTot(), increment i)*

*If result round of i getBetType() equals getResult()*

*Call function result addWin()*

*Else if result round of i getBetType() not equal getResult()*

*Call function result addLoss()*

*End MakeResult*

*Start PrintPlayer*

*Char vector temp assigned to player getName()*

*Size char array output to temp size()*

*Transfer elements from temp vector to output char array*

*Print player name*

*Print average payout*

*End PrintPlayer*

*Start PrintResult*

*Receives:*

*Result pointer: result*

*Result pointer: pastRes*

*Print total rounds, wins, losses, starting money, ending money of result*

*Print total rounds, wins, losses, starting money, ending money of pastRes*

*End PrintResult*

*Start BinaryData*

*Receives:*

*Player pointer: player*

*Result pointer: result*

*Output file stream: output for Rounds, output for Results, output for Player*

*c-string: rounds, results, player*

*char vector temp assigned to player getName()*

*transfer elements of temp to output char array*

*output for Player: create/open file, save output char array*

*output for Results: create/open c-string:results file, in binary mode*

*output for Results:write Round total, wins, losses, player starting money, player ending money*

*output for Rounds: create/open c-string:rounds file, in binary mode*

*for( i=0, i less than result:round total, increment i)*

*output for Rounds: write Round number, bet type, result, sum of banker cards, sum of*

*player cards, payout amount*

*End Binary Data*

*Start MemoryClear*

*Receives:*

*Player pointer: player*

*Banker pointer: banker*

*delete player*

*delete banker*

*End MemoryClear*

*Start ValueCheck (for shorts)*

*Receives:*

*Integer value*

*Constant short: MIN, MAX*

*While value not between MIN and MAX*

*Prompt for valid value*

*Take input to value*

*Return value*

*End ValueCheck*

*Start ValueCheck (for floats)*

*Receives:*

*Float value*

*Constant float: MIN, MAX*

*While value not between MIN and MAX*

*Prompt for valid value*

*Take input to value*

*Return value*

*End ValueCheck*

Project Code:

/\*

\* File: main.cpp

\* Author: Jason Handen

\* Created on May 13, 2021, 1:21 PM

\* Purpose: Project 2 Version 9, Final

\* Line Count: 1081 including classes

\*/

#include <iostream> // for input/output

#include <iomanip> // for output formatting

#include <ctime> // for rand function

#include <fstream> // for binary file read/write

#include <vector> // for STL vector stuff

#include "Hand.h"

#include "Banker.h"

#include "Player.h"

#include "Round.h"

#include "Result.h"

using namespace std;

// Enums

enum bets{PLAYER=1,BANKER,TIE};

// Function Prototypes

Result& prntPrior(Result\*,char[],char[],char[],ifstream&,ifstream&,ifstream&); // pull bin file data to print

vector<char> promptName(); // prompt user for his name

short promptGame(const short,const short); // determine rounds to play

float promptMoney(); // prompt user for starting money value

void playBaccarat(Player\*,Banker\*,Round\*); // plays the game for a given round

short promptBet(const short,const short); // determine user bet

float promptBetAmnt(const float); // prompt for amount to bet

void adjust$(Player\*,Round\*); // adjusts money values

void prntRound(Player\*,Banker\*,Round\*); // print round

void setResult(Player\*,Result\*); // set up data in Result object

void prntPlayer(Player\*,Result\*); // print some player data

void prntResult(Result\*,Result\*); // print results

void binData(Player\*,Result\*,ofstream&,ofstream&,ofstream&,char[],char[],char[]);// save to binary file

void memClear(Player\*,Banker\*); // clean up memory allocations

short valCheck(short,const short, const short); // input validation 1

float valCheck(float,const float, const float); // input validation 2

int main(int argc, char\*\* argv)

{

// seed random number generator

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

// declare file and vars

ifstream inPlay;

ifstream inRnd;

ifstream inRes;

ofstream outPlay;

ofstream outRnd;

ofstream outRes;

// initialize vars

char rounds[]="GameRounds.bin";

char results[]="GameResults.bin";

char play[]="PlayerName.txt";

//char runtime

const short RNDMIN=1;

const short RNDMAX=20;

Result temp;

Result \*tempPtr = &temp;

// pull from binary file and print Result and Round information

Result pastResult(prntPrior(tempPtr,rounds,results,play,inRnd,inRes,inPlay));

Result \*pastRes = &pastResult;

// determine games to play

vector<char> name=promptName();

short rndNumb=promptGame(RNDMIN,RNDMAX);

// (ROUND COUNT OUT OF RANGE) substitute input validation for exception class

try{

// and create objects

Result result(rndNumb); // Result object

Result \*res=&result; // Result pointer for functions

Player \*player = new Player; // Player object

Banker \*banker = new Banker; // Banker object

player->setName(name);

player->setFMoney(promptMoney()); // set first money

result.setPlayerS$(player->getMoney()); // store in result object

for(short i=0;i<result.getRndTot();i++){

// replaced by static counter

playBaccarat(player,banker,result.round+i); // runs game until complete

}

setResult(player,res); // set Result object w/ data

prntPlayer(player,res); // print player name

prntResult(res,pastRes); // print Result object parts

binData(player,res,outRnd,outRes,outPlay,rounds,results,play); // output to binary file

memClear(player,banker); // memory cleanup

}catch(Result::BadRoundCount){

cout<<"Error: Invalid round count entered. Exiting program...\n\n";

inPlay.close(); // close file input

inRes.close(); // close file input

inRnd.close(); // close file input

outPlay.close(); // close file output

outRes.close(); // close file output

outRnd.close(); // close file outpu

return 1;

}

inPlay.close(); // close file input

inRes.close(); // close file input

inRnd.close(); // close file input

outPlay.close(); // close file output

outRes.close(); // close file output

outRnd.close(); // close file output

return 0;

}

Result& prntPrior(Result \*temp,char rounds[],char results[],char play[],

ifstream &inRnd,ifstream &inRes,ifstream &inPlay){

short response=0;

//Result result;

Player player;

Banker banker;

inPlay.open(play,ios::in);

inRnd.open(rounds,ios::binary|ios::in);

inRes.open(results,ios::binary|ios::in);

if(!inRnd){

cout<<"No Round file to open. This must be the first run!\n";

}else if(!inPlay){

cout<<"No Player file to open. This must be the first run!\n";

}else if(!inRes){

cout<<"No Result file to open. This must be the first run!\n";

}else{

short tempRnd,tempWin,tempLoss,tempS$,tempE$,leng;

// read string from PlayerName file

string name, s;

while(inPlay>>s){

name+=s;

name+=" ";

}

// read in values from Results file

inRes.read((char\*)&tempRnd,sizeof(short));

inRes.read((char\*)&tempWin,sizeof(short));

inRes.read((char\*)&tempLoss,sizeof(short));

inRes.read((char\*)&tempS$,sizeof(short));

inRes.read((char\*)&tempE$,sizeof(short));

// assign values to object

temp->setRndTot(tempRnd);

temp->setWins(tempWin);

temp->setLosses(tempLoss);

temp->setPlayerS$(tempS$);

temp->setPlayerE$(tempE$);

cout<<"Last player name: "<<name<<endl;

cout<<"Round total for the last game played: "

<<temp->getRndTot()<<endl;

cout<<"Total wins for last game: "<<temp->getWins()<<endl;

cout<<"Total losses for last game: "<<temp->getLosses()<<endl;

cout<<"Starting Money: $"<<fixed<<setprecision(2)<<temp->getPlayerS$()<<endl;

cout<<" Ending Money: $"<<temp->getPlayerE$()<<endl<<endl;

short rndNum,betType,res,sumB,sumP;

float payout;

do{

inRnd.seekg(0);

cout<<"Choose a round, 1-"<<temp->getRndTot()<<", or 0 (Zero) to exit: ";

cin>>response;

response=valCheck(response,0,temp->getRndTot());

for(short i=0;i<response;i++){

inRnd.read((char\*)&rndNum,sizeof(short));

inRnd.read((char\*)&betType,sizeof(short));

inRnd.read((char\*)&res,sizeof(short));

inRnd.read((char\*)&sumB,sizeof(short));

inRnd.read((char\*)&sumP,sizeof(short));

inRnd.read((char\*)&payout,sizeof(float));

}

if(response!=0){

temp->round->setRndNum(rndNum);

temp->round->setBetType(betType);

temp->round->setResult(res);

temp->round->setSumBnkr(sumB);

temp->round->setSumPlyr(sumP);

temp->round->setPayout(payout);

cout<<"For Round #"<<temp->round->getRndNum()<<":\n";

if(temp->round->getBetType()==PLAYER){

cout<<"Player bet with ";

}else if(temp->round->getBetType()==BANKER){

cout<<"Banker bet with ";

}else{

cout<<"Tie bet with ";

}

if(temp->round->getResult()==PLAYER){

cout<<"player win.\n";

}else if(temp->round->getResult()==BANKER){

cout<<"banker win.\n";

}else{

cout<<"tie game.\n";

}

cout<<"Player sum: "<<temp->round->getSumPlyr()<<endl;

cout<<"Banker sum: "<<temp->round->getSumBnkr()<<endl;

cout<<"Round payout: "<<fixed<<setprecision(2)

<<temp->round->getPayout()<<" dollars\n\n";

}

}while(response!=0);

cin.ignore();

}

return \*(temp);

}

vector<char> promptName(){

string temp;

vector<char> name;

cout<<"What is your name: ";

getline(cin, temp);

for(short i=0;i<temp.size();i++){

name.push\_back(temp[i]);

}

cout<<endl<<endl;

cout<<"Hello, "<<temp<<"!"<<endl;

return name;

}

short promptGame(const short RNDMIN,const short RNDMAX){

short rndNumb=0;

cout<<"How many rounds of Bacarrat would you like to play: ";

cin>>rndNumb;

// input validation canceled, exception handling for this case

return rndNumb;

}

float promptMoney(){

float money=0;

cout<<"How much money will you start out with: ";

cin>>money;

// input validation at assignment for object

return money;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////

void playBaccarat(Player \*player,Banker \*banker,Round \*round){

round->setBetType( promptBet(PLAYER,TIE) );

cout<<"You have $"<<fixed<<setprecision(2)<<player->getMoney()<<endl;

round->setBetAmnt( promptBetAmnt(player->getMoney()) );

Hand \*polyTest[]={player,banker};

for(short i=0;i<2;i++){

polyTest[i]->dealCards();

}

// if player's sum less than 6, draw third card, add to sum

if(player->getNatural()==false&&banker->getNatural()==false){

if(player->getSum()<6){

player->drawThird();

}

}

// run through banker conditions for drawing

if(player->getNatural()==false&&banker->getNatural()==false){

if(player->getHit()==false&&banker->getSum()<6){

banker->trueHit();

}else if(player->getHit()==false&&banker->getSum()>5){

// banker stands

}else if(banker->getSum()<3){

banker->trueHit();

}else if(player->getHit()==true&&banker->getSum()==3

&&player->getValue(2)!=8){

banker->trueHit();

}else if(player->getHit()==true&&banker->getSum()==4

&&player->getValue(2)>1&&player->getValue(2)<8){

banker->trueHit();

}else if(player->getHit()==true&&banker->getSum()==5

&&player->getValue(2)>3&&player->getValue(2)<8){

banker->trueHit();

}else if(player->getHit()==true&&banker->getSum()==6

&&player->getValue(2)>5&&player->getValue(2)<8){

banker->trueHit();

}

}

// set result based on sum comparisons

if(player->getSum() > banker->getSum()){

round->setResult(PLAYER);

}else if(player->getSum() < banker->getSum()){

round->setResult(BANKER);

}else{

round->setResult(TIE);

}

// round object assignments for later binary output

round->setSumPlyr(player->getSum());

round->setSumBnkr(banker->getSum());

// print results of that round

adjust$(player,round);

prntRound(player,banker,round);

}

//////////////////////////////////////////////////////////////////////////////////////////////////////

short promptBet(const short MIN,const short MAX){

short bet;

cout<<"\nPress 1 to bet on the player.\n"

<<"Press 2 to bet on the banker.\n"

<<"Press 3 to bet on a tie.\n"

<<" Choice: ";

cin>>bet;

bet=valCheck(bet,MIN,MAX);

return bet;

}

float promptBetAmnt(const float MAX){

float bet;

cout<<"How much would you like to bet: ";

cin>>bet;

bet=valCheck(bet,0.00,MAX);

return bet;

}

void adjust$(Player \*player,Round \*round){

if(round->getBetType() != round->getResult()){

player->setMoney(\*(player) - round->getBetAmnt());

round->setPayout(-round->getBetAmnt());

}else{

if(round->getResult()==PLAYER){

player->setMoney(\*(player) + round->getBetAmnt()); // Player bet win doubles

round->setPayout(2\*round->getBetAmnt());

}else if(round->getResult()==BANKER){

player->setMoney(\*(player) + round->getBetAmnt() \* .95); // Banker bet win -commission

round->setPayout(1.95\*round->getBetAmnt());

}else{

player->setMoney(\*(player) + round->getBetAmnt() \* 7); // Tie bet win 8 to 1 payout

round->setPayout(8\*round->getBetAmnt());

}

}

}

void prntRound(Player \*player,Banker \*banker,Round \*round){

const short WIDTH=25;

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

cout<<setw(WIDTH)<<left<<"Player"<<setw(WIDTH)<<"Banker"<<endl;

cout<<setw(WIDTH)<<left<<player->getCard(0)

<<setw(WIDTH)<<left<<banker->getCard(0)<<endl;

cout<<setw(WIDTH)<<left<<player->getCard(1)

<<setw(WIDTH)<<left<<banker->getCard(1)<<endl;

if(player->getHit()==true){

cout<<setw(WIDTH)<<left<<player->getCard(2);

if(banker->getHit()==true){

cout<<setw(WIDTH)<<left<<banker->getCard(2);

}

cout<<endl;

}else if(banker->getHit()==true){

cout<<setw(WIDTH)<<" "<<setw(WIDTH)<<left<<banker->getCard(2)<<endl;

}

cout<<setw(WIDTH-5)<<left<<"Sum: "<<player->getSum()<<" "

<<setw(WIDTH-5)<<left<<"Sum: "<<banker->getSum()<<endl;

if(player->getSum() > banker->getSum()){

cout<<setw(WIDTH)<<left<<"WINNER"<<endl;

}else if(player->getSum() < banker->getSum()){

cout<<setw(WIDTH)<<" "<<setw(WIDTH)<<left<<"WINNER"<<endl;

}else{

cout<<setw(WIDTH)<<left<<"TIE"<<setw(WIDTH)<<left<<"TIE"<<endl;

}

if(round->getBetType()==round->getResult()){

cout<<"Won Bet, payout of $"<<fixed<<setprecision(2)

<<round->getPayout()<<endl;

}else if(round->getBetType()!=round->getResult()){

cout<<"Lost Bet, at a cost of $"<<fixed<<setprecision(2)

<<-round->getPayout()<<endl;

}

cout<<"New total money available: $"<<fixed<<setprecision(2)

<<player->getMoney()<<endl;;

cout<<"Press Enter to Continue... ";

cin.ignore();

cin.get();

}

void setResult(Player \*player, Result \*result){

result->setPlayerE$(player->getMoney()); // set final money value

for(short i=0;i<result->getRndTot();i++){

if(result->round[i].getBetType()==result->round[i].getResult()){

result->addWin(); // total wins and store

}else if(result->round[i].getBetType()!=result->round[i].getResult()){

result->addLoss(); // total losses and store

}

}

}

void prntPlayer(Player \*player,Result \*result){

vector<char> temp=player->getName();

char output[temp.size()];

for(short i=0;i<temp.size();i++){

output[i]=temp[i];

}

cout<<"\n\nPlayer name: "<<output<<endl;

cout<<"Average payout: "<<\*(player)/result->getRndTot()<<endl;

}

void prntResult(Result \*result,Result\* pastRes){

float avg, pastAvg;

cout<<"\nTotal Rounds: "<<result->getRndTot()<<endl;

cout<<"Won bets: "<<result->getWins()<<endl;

cout<<"Lost bets: "<<result->getLosses()<<endl;

cout<<"Starting Money: $"<<fixed<<setprecision(2)<<result->getPlayerS$()<<endl;

cout<<"Ending Money: $"<<result->getPlayerE$()<<endl;

cout<<"---Compared with the prior game---\n";

cout<<"Total Rounds: "<<pastRes->getRndTot()<<endl;

cout<<"Won bets: "<<pastRes->getWins()<<endl;

cout<<"Lost bets: "<<pastRes->getLosses()<<endl;

cout<<"Starting Money: $"<<fixed<<setprecision(2)<<pastRes->getPlayerS$()<<endl;

cout<<"Ending Money: $"<<pastRes->getPlayerE$()<<endl<<endl;

}

void binData(Player \*player,Result \*result,ofstream &outRnd,ofstream &outRes,ofstream &outPlay,

char rounds[],char results[],char play[]){

vector<char> temp=player->getName();

short leng=temp.size();

char output[leng];

for(short i=0;i<leng;i++){

output[i]=temp[i];

}

outPlay.open(play,ios::out);

//outPlay.write((char\*)&leng,sizeof(short));

for(short i=0;i<leng;i++){

outPlay<<temp[i];

}

outRes.open(results,ios::binary|ios::out);

short tempTot=result->getRndTot();

short tempWins=result->getWins();

short tempLosses=result->getLosses();

short tempS$=result->getPlayerS$();

short tempE$=result->getPlayerE$();

outRes.write((char\*)&tempTot,sizeof(short));

outRes.write((char\*)&tempWins,sizeof(short));

outRes.write((char\*)&tempLosses,sizeof(short));

outRes.write((char\*)&tempS$,sizeof(short));

outRes.write((char\*)&tempE$,sizeof(short));

// temporary variables for assignment and binary output

short tempRndNum,tempBetType,tempResult,tempSumB,tempSumP;

float tempPayout;

outRnd.open(rounds,ios::binary|ios::out);

for(short i=0;i<tempTot;i++){

// temp variable assignment

tempRndNum=result->round[i].getRndNum();

tempBetType=result->round[i].getBetType();

tempResult=result->round[i].getResult();

tempSumB=result->round[i].getSumBnkr();

tempSumP=result->round[i].getSumPlyr();

tempPayout=result->round[i].getPayout();

// file output using temp variables

outRnd.write((char\*)&tempRndNum,sizeof(short));

outRnd.write((char\*)&tempBetType,sizeof(short));

outRnd.write((char\*)&tempResult,sizeof(short));

outRnd.write((char\*)&tempSumB,sizeof(short));

outRnd.write((char\*)&tempSumP,sizeof(short));

outRnd.write((char\*)&tempPayout,sizeof(float));

}

}

void memClear(Player \*player,Banker \*banker){

delete player;

delete banker;

}

short valCheck(short value,const short MIN,const short MAX){

while(value<MIN || value>MAX){

cout<<"Please choose a valid entry between "<<MIN<<" and "<<MAX<<": ";

cin>>value;

}

return value;

}

float valCheck(float value,const float MIN,const float MAX){

while(value<MIN || value>MAX){

cout<<"Please choose a valid entry between "<<MIN<<" and "<<MAX<<": ";

cin>>value;

}

return value;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////

/\*

\* File: AbsHand.h

\* Author: Jason Handen

\* Created on May 31, 2021, 8:48 PM

\* Purpose: full virtual class for inheritance

\*/

#ifndef ABSHAND\_H

#define ABSHAND\_H

#include <string>

using namespace std;

class AbsHand{

public:

AbsHand(){};

~AbsHand(){};

virtual void drawThird()=0; // different implementation for player/banker

virtual void dealCards()=0; // deal all three cards

virtual void writeCards()=0; // write card name to string

virtual void sumCards(short,short)=0;// sum given cards

virtual string getCard(short)=0; // return card name

virtual bool getNatural()=0; // return natural status, true or false

virtual bool getHit()=0; // return hit status, true or false

virtual short getSum()=0; // return sum of hand

virtual void setNatural()=0; // set Natural to true

virtual void trueHit()=0; // set Hit to true, call sum

};

#endif /\* ABSHAND\_H \*/

//////////////////////////////////////////////////////////////////////////////////////////////////////

/\*

\* File: Banker.h

\* Author: Jason Handen

\* Created on May 30, 2021, 8:40 PM

\* Purpose: Banker header file

\*/

#ifndef BANKER\_H

#define BANKER\_H

#include "Hand.h"

class Banker:public Hand{

private:

// no extra member variables

public:

void drawThird(); // overwrites virtual from Hand abstract class

};

#endif /\* BANKER\_H \*/

//////////////////////////////////////////////////////////////////////////////////////////////////////

/\*

\* File: Banker.cpp

\* Author: Jason Handen

\* Created on May 30, 2021, 8:40 PM

\* Purpose: Banker source file

\*/

#include "Banker.h"

void Banker::drawThird(){

trueHit();

}

//////////////////////////////////////////////////////////////////////////////////////////////////////

/\*

\* File: Hand.h

\* Author: Jason Handen

\* Created on May 30, 2021, 7:30 PM

\* Purpose: Hand header file (abstract class), for inheritance

\*/

#ifndef HAND\_H

#define HAND\_H

#include "AbsHand.h"

#include <string>

using namespace std;

class Hand:public AbsHand{

protected:

string card[3];

short value[3];

bool natural=false;

bool hit=false;

short sum;

// Utility Functions

void writeCards(); // write card name to string

void sumCards(short,short); // sum given cards

void setNatural(); // check to set Natural to true

public:

enum suit{CLUBS,DIAMONDS,HEARTS,SPADES}; // for card naming

Hand();

void drawThird(){}; // different implementation for player/banker

void dealCards(); // deal all three cards

string getCard(short); // return card name

short getValue(short); // return card value

bool getNatural(); // return natural status, true or false

bool getHit(); // return hit status, true or false

short getSum(); // return sum of hand

void trueHit(); // set Hit to true, call sum

// no input validation necessary

};

#endif /\* HAND\_H \*/

//////////////////////////////////////////////////////////////////////////////////////////////////////

/\*

\* File: Hand.cpp

\* Author: Jason Handen

\* Created on May 30, 2021, 7:30 PM

\* Purpose: Hand source file

\*/

#include "Hand.h"

#include <string>

using namespace std;

// CONSTRUCTOR

Hand::Hand(){}

// Work Function

void Hand::dealCards(){

for(short i=0;i<3;i++){

value[i]=rand()%52; // value[2] only used if hit set to true

}

Hand::writeCards();

Hand::sumCards(value[0],value[1]);

Hand::setNatural();

}

// GETTER FUNCTIONS

string Hand::getCard(short index){

return card[index];

}

short Hand::getValue(short index){

return value[index];

}

bool Hand::getNatural(){

return natural;

}

bool Hand::getHit(){

return hit;

}

short Hand::getSum(){

return sum;

}

//SETTER FUNCTION

void Hand::trueHit(){

hit=true;

Hand::sumCards(sum,value[2]);

}

// Protected Utility Functions

void Hand::writeCards(){

for(short i=0;i<3;i++){

short suit=value[i]/13;

// determine card face

value[i]%=13;

if(value[i]==0){ // hand value of 0 is "Ace"

card[i]="Ace"; // value corrected below

}else if(value[i]==1){

card[i]="Two";

}else if(value[i]==2){

card[i]="Three";

}else if(value[i]==3){

card[i]="Four";

}else if(value[i]==4){

card[i]="Five";

}else if(value[i]==5){

card[i]="Six";

}else if(value[i]==6){

card[i]="Seven";

}else if(value[i]==7){

card[i]="Eight";

}else if(value[i]==8){

card[i]="Nine";

}else if(value[i]==9){

card[i]="Ten";

}else if(value[i]==10){

card[i]="Jack";

}else if(value[i]==11){

card[i]="Queen";

}else if(value[i]==12){

card[i]="King";

}

// set to proper value for summing later

if(value[i]>=0 && value[i]<=8){

value[i]+=1; // pip cards have face value

}else{

value[i]=0; // 10s and face cards value 0

}

// add " of " for formatting

card[i]+=" of ";

// determine card suit and concatenate

switch(suit){

case CLUBS:card[i]+="Clubs"; break;

case DIAMONDS:card[i]+="Diamonds"; break;

case HEARTS:card[i]+="Hearts"; break;

case SPADES:card[i]+="Spades"; break;

}

}

}

void Hand::sumCards(short a,short b){

sum = a + b;

if(sum>9){

sum-=10;

}

}

void Hand::setNatural(){

if(sum>7){

natural=true;

}

}

//////////////////////////////////////////////////////////////////////////////////////////////////////

/\*

\* File: Player.h

\* Author: Jason Handen

\* Created on May 30, 2021, 8:47 PM

\* Purpose: Player header file

\*/

#ifndef PLAYER\_H

#define PLAYER\_H

#include "Hand.h"

#include <vector>

using namespace std;

class Player:public Hand{

private:

float start$=0; // to implement monetary betting system

float end$=0;

vector<char> name;

public:

Player();

Player(float);

void drawThird();

inline float getSMoney(){

return start$;

}

inline float getMoney(){

return end$;

}

inline vector<char> getName(){

return name;

}

void setFMoney(float);

void setMoney(float);

void setName(vector<char>);

friend float operator + (Player,float);

friend float operator - (Player,float);

friend float operator / (Player,float);

float valCheck(float,const short,const int);

};

#endif /\* PLAYER\_H \*/

//////////////////////////////////////////////////////////////////////////////////////////////////////

/\*

\* File: Player.cpp

\* Author: Jason Handen

\* Created on May 30, 2021, 8:47 PM

\* Purpose: Player source file

\*/

#include "Player.h"

#include <iostream>

using namespace std;

Player::Player(){}

Player::Player(float m){

Player::setFMoney(m);

Player::setMoney(m);

}

void Player::drawThird(){

trueHit();

}

// Replaced get function once it became inline

void Player::setFMoney(float m){

m=valCheck(m,0,100000);

start$=m;

end$=m;

}

void Player::setMoney(float m){

m=valCheck(m,0,100000);

end$=m;

}

void Player::setName(vector<char> s){

name=s;

}

// add money after won bet

float operator + (Player player,float a){

return player.end$+a;

}

// subtract money after lost bet

float operator - (Player player,float s){

return player.end$-s;

}

// for averaging money

float operator / (Player player,float d){

return (player.end$-player.start$)/d;

}

// input validation implementation

float Player::valCheck(float value,const short MIN,const int MAX){

while(value<MIN || value>MAX){

cout<<"Please choose a valid entry between "<<MIN<<" and "<<MAX<<": ";

cin>>value;

}

return value;

}

//////////////////////////////////////////////////////////////////////////////////////////////////////

/\*

\* File: Result.h

\* Author: Jason Handen

\* Created on May 31, 2021, 6:22 PM

\* Purpose: Result header file

\*/

#ifndef RESULT\_H

#define RESULT\_H

#include "Round.h"

class Result{

private:

short rndTot=-1;

short wins=0;

short losses=0;

float playerS$=0;

float playerE$=0;

public:

Round \*round;

class BadValueEx{};

class BadRoundCount{};

Result();

Result(short);

Result(Result&);

~Result();

short getRndTot();

short getWins();

short getLosses();

float getPlayerS$();

float getPlayerE$();

void setRndTot(short);

void setWins(short);

void setLosses(short);

void setPlayerS$(float);

void setPlayerE$(float);

void addWin();

void addLoss();

template<class T> void valCheck(T,const short,const short);

template<class T> void valCheck(T,const short);

};

#endif /\* RESULT\_H \*/

//////////////////////////////////////////////////////////////////////////////////////////////////////

/\*

\* File: Result.cpp

\* Author: Jason Handen

\* Created on May 31, 2021, 6:22 PM

\* Purpose: Result source file

\*/

#include "Result.h"

#include "Round.h"

#include <iostream>

// constructor for first use

Result::Result(){

rndTot=0;

Round \*rnd = new Round;

round=rnd;

}

// constructor for second use

Result::Result(short s){

if(s>0&&s<21){

rndTot=s;

Round \*rnd = new Round[rndTot];

round=rnd;

}else{

throw BadRoundCount();

}

}

Result::Result(Result& obj){

rndTot=obj.getRndTot();

wins=obj.getWins();

losses=obj.getLosses();

playerS$=obj.getPlayerS$();

playerE$=obj.getPlayerE$();

Round \*rnd = new Round;

round=rnd;

}

// destructor to clean up dynamic memory allocation

Result::~Result(){

delete round;

}

// GETTERS

short Result::getRndTot(){

return rndTot;

}

short Result::getWins(){

return wins;

}

short Result::getLosses(){

return losses;

}

float Result::getPlayerS$(){

return playerS$;

}

float Result::getPlayerE$(){

return playerE$;

}

// SETTERS

void Result::setRndTot(short t){

valCheck(t,1,20);

rndTot=t;

}

void Result::setWins(short w){

valCheck(w,0,20);

wins=w;

}

void Result::setLosses(short l){

valCheck(l,0,20);

losses=l;

}

void Result::setPlayerS$(float m){

valCheck(m,0);

playerS$=m;

}

void Result::setPlayerE$(float m){

playerE$=m;

}

// adds one to wins category

void Result::addWin(){

++wins;

}

// adds one to losses category

void Result::addLoss(){

++losses;

}

template<class T>

void Result::valCheck(T value,const short MIN,const short MAX){

if(value<MIN || value>MAX){

throw BadValueEx();

}

}

template<class T>

void Result::valCheck(T value,const short MIN){

if(value<MIN){

throw BadValueEx();

}

}

//////////////////////////////////////////////////////////////////////////////////////////////////////

/\*

\* File: Round.h

\* Author: Jason Handen

\* Created on May 30, 2021, 9:14 PM

\* Purpose: Round header file

\*/

#ifndef ROUND\_H

#define ROUND\_H

class Round{

private:

short rndNum;

short betType;

short sumPlyr;

short sumBnkr;

short result;

float betAmnt;

float payout;

public:

enum bets{PLAYER=1,BANKER,TIE};

class BadValueEx{};

Round();

static short getCount();

bool compBetResult();

short getRndNum();

short getBetType();

float getBetAmnt();

short getSumPlyr();

short getSumBnkr();

short getResult();

float getPayout();

void setRndNum(short);

void setBetType(short);

void setBetAmnt(float);

void setSumPlyr(short);

void setSumBnkr(short);

void setResult(short);

void setPayout(float);

template<class T> void valCheck(T,const short,const short);

template<class T> void valCheck(T,const short);

};

#endif /\* ROUND\_H \*/

//////////////////////////////////////////////////////////////////////////////////////////////////////

/\*

\* File: Round.cpp

\* Author: Jason Handen

\* Created on May 30, 2021, 9:14 PM

\* Purpose: Round source file

\*/

#include "Round.h"

static short count=-2; // aligns as index for program

Round::Round(){

rndNum=getCount();

}

short Round::getCount(){

count++;

return count;

}

bool Round::compBetResult(){

return betType==result;

}

short Round::getRndNum(){

return rndNum;

}

short Round::getBetType(){

return betType;

}

float Round::getBetAmnt(){

return betAmnt;

}

short Round::getSumPlyr(){

return sumPlyr;

}

short Round::getSumBnkr(){

return sumBnkr;

}

short Round::getResult(){

return result;

}

float Round::getPayout(){

return payout;

}

void Round::setRndNum(short rnd){

valCheck(rnd,0,20);

rndNum=rnd;

}

void Round::setBetType(short b){

valCheck(b,1,3);

betType=b;

}

void Round::setBetAmnt(float m){

valCheck(m,0);

betAmnt=m;

}

void Round::setSumPlyr(short sumP){

valCheck(sumP,0,9);

sumPlyr=sumP;

}

void Round::setSumBnkr(short sumB){

valCheck(sumB,0,9);

sumBnkr=sumB;

}

void Round::setResult(short res){

valCheck(res,1,3);

result=res;

}

void Round::setPayout(float p){

payout=p;

}

template<class T>

void Round::valCheck(T value,const short MIN,const short MAX){

if(value<MIN || value>MAX){

throw BadValueEx();

}

}

template<class T>

void Round::valCheck(T value,const short MIN){

if(value<MIN){

throw BadValueEx();

}

}

//////////////////////////////////////////////////////////////////////////////////////////////////////