Jason Handen

Dr. Lehr

CSC-17A (43396)

13 May 2021

**Project 1: Baccarat Documentation**

Introduction:The casino game Baccarat is a card game in which the player has little agency, aside from his initial bet. 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.

Summary:

*Project Size:* 418 lines, given whitespace formatting.

*Number of Variables:* about 20, including structures variables

*Number of Functions:* 12, not including “main”

This C++ application roughly emulates the casino game Baccarat. It allows for a review of the last game’s results and rounds via binary files. The user can choose a number of rounds to play, choose to bet on the “Player,” “Banker,” or on a “Tie,” although chip values and a running total for monetary bets was no implemented in this version. 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 bet on 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.

The project was relatively straightforward, once I had decided on a game to create. There was certainly some challenge in deciding how to implement various elements from the course, but many of those elements fell right into place. Across the five days I developed the program itself, I spent about twenty-five hours of serious time at the computer. About two of those hours were dedicated to tracking down a one-character typo, attempting to print an element outside the bounds of an array (should have been a 2 instead of a 3), causing a runtime error. StackOverflow was a great resource in reviewing some of the concepts and providing specific examples.

Overall, I am pleased with the product of my efforts, and feel that, while this project was not easy, I have a better grasp of the various concepts found therein.

Description:The main point of this program was to implement Chapter 9 through 12 concepts from the Gaddis textbook in the form of a card game.

Diagram, polygon

Description automatically generatedFlowchart examples:

Major Variables:

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable Name** | **Description** | **Location** |
| Enum | Bets | Numerical values for user bet entry | Prior to main function |
|  | Suit | Numerical values for use in card suit conversion | Prior to main function |
| Structure | Hand (5 variables included) | Holds information about a given hand: card strings, their values, “natural” status, “hit” status, and value sum | Prior to main function,  Instantiated in main(),  temporary examples in prntPrior function |
|  | Round (5 variables included) | Holds information about a given round: round number, user bet number, player and banker Hand, and the result number | Prior to main function,  Array instantiated in main(),  temporary examples in prntPrior function |
|  | Result (3 variables included) | Holds information about the total game’s results: total number of rounds, total wins, total losses | Prior to main function,  Instantiated in makeResult(round, rndNumb),  temporary examples in prntPrior function |
| C-string | Rounds[] | File name for GameRounds access | main() |
|  | Results[] | File name the GameResults access | main() |
| Constant Short Int | RNDMIN | Round minimum for user entry | main() |
|  | RNDMAX | Round maximum for user entry | main() |
|  | WIDTH | Hold value for setw formatting | prntRound(Round \*round) |
| Short Int | rndNumb | Holds number of rounds to be played | main(),promptGame(const short RNDMIN, const short RNDMAX) |
|  | response | Holds round to read from binary file, or escape | prntPrior(char rounds[], char results[], ifstream &inRnd, ifstream &inRes) |
|  | bet | Holds user bet choice | promptBet(const short MIN, const short MAX) |
|  | suit | Holds result of integer division for determination of card suit | writeCards(Hand \*hand) |
|  | value | used to determine validity of user entry | valCheck(short value, const short MIN, const short MAX) |

C++ Course Concepts:

|  |  |  |  |
| --- | --- | --- | --- |
| **Chapter** | **Section** | **Concept** | **Location in Code** |
|  |  |  |  |
| **9** |  | **Pointers/Memory Allocation** |  |
|  | 2 | Pointer Variables | Line 370 |
|  | 3 | Arrays/Pointers | Line 23 |
|  | 4 | Pointer Arithmetic | Line 88 |
|  | 5 | Pointer Initialization | Line 91 |
|  | 7 | Function Parameters | Line 87 |
|  | 8 | Memory Allocation | Line 80 |
|  | 9 | Return Parameters | Line 380 |
|  | 10 | Smart Pointers |  |
|  |  |  |  |
| **10** |  | **Char Arrays and Strings** |  |
|  | 3 | C-Strings | Line 67-68 |
|  | 7 | Strings | Line 22, concat. Line 288 |
|  |  |  |  |
| **11** |  | **Structured Data** |  |
|  | 5 | Arrays | Line 76 |
|  | 6 | Nested | Line 31-32 |
|  | 7 | Function Arguments | Line 263 |
|  | 8 | Function Return | Line 380 |
|  | 9 | Pointers | Line 88 |
|  | 11 | Enumeration | Line 17-18 |
|  |  |  |  |
| **12** |  | **Binary Files** |  |
|  | 1 | File Operations | Line 119-121 |
|  | 2 | Formatting | Line 391-393 |
|  | 3 | Function Parameters | Line 73 |
|  | 4 | Error Testing | Line 114-117 |
|  | 5 | Member Functions | Line 387 |
|  | 6 | Multiple Files | Line 111-112 |
|  | 7 | Binary Files | Line 389, 395 |
|  | 8 | Records with Structures | Line 391-393 |
|  | 9 | Random Access Files | Line 128-139 |
|  | 10 | Input/Output Simultaneous | Line 111-112, 389, 95-98 |

Project Pseudocode:

*Enumerated type, User Bets:*

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

*Enumerated type, Card Suits:*

*CLUBS, DIAMONDS, HEARTS, SPADES*

*Structure, Hand Data:*

*Three card names, strings*

*Three card values, integers*

*Natural status, boolean*

*Hit status, boolean*

*Sum of cards, integer*

*Structure, Round Data:*

*Round number, integer*

*Bet type, integer*

*Hand structure for player*

*Hand structure for banker*

*Game result, integer*

*Structure, Game Result:*

*Total number of rounds, integer*

*Number of wins, integer*

*Number of losses, integer*

*Start main:*

*Seed random number generator, time(0)*

*Declare file streams:*

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

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

*Create C-strings:*

*rounds[] = “GameRounds.bin”*

*results[] = “GameResults.bin”*

*Constant Integers:*

*ROUND MINIMUM (RNDMIN) = 1*

*ROUND MAXIMUM (RNDMAX) = 20*

*Call function: PrintPrior(rounds, results, input for Rounds, input for Results)*

*Integer round number = Call function: PromptGame(RNDMIN, RNDMAX)*

*Create new Round array of size: round number*

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

*Create new Hand, assign to Round array of i:player*

*Create new Hand, assign to Round array of i:banker*

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

*Assign value i+1 to Round array of i:Round number*

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

*New Result structure = Call function: MakeResult(Round array, Round number)*

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

*Call function: BinaryData(Round array, Result structure, output for Rounds, output for Results,*

*rounds c-string, results c-string)*

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

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

*Return 0*

*End main*

*Start PrintPrior*

*Receives:*

*c-string: rounds, results*

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

*Integer response = 0*

*New Result structure, New Round structure, New Hand structure:play, New Hand structure:bank*

*Assign Round structure:player to Hand Structure:play*

*Assign Round structure:banker to Hand Structure:bank*

*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:results file for input:*

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

*Else*

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

*Store in Results structure: Round total, Wins, Losses*

*Print Rounds, Wins, Losses*

*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)*

*Input to Round Structure:Round number, Bet type, Result, banker:sum,*

*player:sum*

*If response does not equal 0*

*Print Round number, Bet type, Result, player:sum, banker:sum*

*While response does not equal 0*

*Delete all structures created in this function*

*End PrintPrior*

*Start PromptGame*

*Receives:*

*Constant integers: RNDMIN, RNDMAX*

*Integer Round number = 0*

*Prompt user for number of games to play*

*Take input to Round number*

*Round number = ValueCheck(Round number, RNDMIN, RNDMAX)*

*Return Round number*

*End PromptGame*

*Start PlayBaccarat*

*Receives:*

*Single Round structure: round*

*Function call: DealCards(round:player)*

*Function call: DealCards(round:banker)*

*Round:player:sum = round:player:value of 0 + round:player: value of 1*

*If round:player:sum is greater than 9*

*Subtract 10 from round:player:sum*

*Round:banker:sum = round:banker:value of 0 + round:banker: value of 1*

*If round:banker:sum is greater than 9*

*Subtract 10 from round:banker:sum*

*If round:banker:sum is greater than 7*

*Round:banker:natural set to true*

*Else round:banker:natural set to false*

*If round:player:sum is greater than 7*

*Round:player:natural set to true*

*Else round:player:natural set to false*

*If round:player:sum less than 6*

*Player gets third card (value of 2), hit set to true, add to sum*

*If round:player:sum is greater than 9*

*Subtract 10 from round:player:sum*

*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*

*Banker gets third card (value of 2), hit set to true, add to sum*

*If round:banker:sum is greater than 9*

*Subtract 10 from round:banker:sum*

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

*// banker stands*

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

*Banker gets third card (value of 2), hit set to true, add to sum*

*If round:banker:sum is greater than 9*

*Subtract 10 from round:banker:sum*

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

*Banker gets third card (value of 2), hit set to true, add to sum*

*If round:banker:sum is greater than 9*

*Subtract 10 from round:banker:sum*

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

*Banker gets third card (value of 2), hit set to true, add to sum*

*If round:banker:sum is greater than 9*

*Subtract 10 from round:banker:sum*

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

*Banker gets third card (value of 2), hit set to true, add to sum*

*If round:banker:sum is greater than 9*

*Subtract 10 from round:banker:sum*

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

*Banker gets third card (value of 2), hit set to true, add to sum*

*If round:banker:sum is greater than 9*

*Subtract 10 from round:banker:sum*

*If round:player:sum greater than round:banker:sum*

*Round:result = PLAYER*

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

*Round:result = BANKER*

*Else round:result = TIE*

*Call function: PrintRound(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 DealCards*

*Receives:*

*Hand structure: hand*

*For( i=0, i less than 3, increment i)*

*Hand:value of I = rand()%52*

*Function call: WriteCards(hand)*

*End DealCards*

*Start WriteCards*

*Receives:*

*Hand structure: hand*

*For(i=0, i less than 3, increment i)*

*Integer suit = hand:value of i divided by 13 (integer division)*

*Hand:value of i %= 13*

*If hand:value of i is 0, hand:card of i = “Ace”*

*Else if hand:value of i is 1, hand:card of i = “Two”*

*Else if hand:value of i is 3, hand:card of i = “Three”*

*And so on…*

*Else if hand:value of i is 12, hand:card of i = “King”*

*If hand:value of i between 0 and 8, incremement hand:value of i*

*Else hand:value of i assigned to 0*

*Hand:card of i += “ of “ //string concatenation*

*Switch(suit)*

*Case CLUBS: hand:card of i += “Clubs”, break*

*Case DIAMONDS: hand:card of i += “Diamonds”, break*

*Case HEARTS: hand:card of i += “Hearts”, break*

*Case SPADES: hand:card of i += “Spades”, break*

*End WriteCards*

*Start PrintRound*

*Receives:*

*Single Round structure: round*

*Start format player and banker columns*

*Print cards 0 and 1*

*If round:player:hit is true, print player card 2*

*If round:banker:hit is true, print player 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 “Press enter to continue…”*

*End PrintRound*

*Start MakeResult*

*Receives:*

*Round Array: round*

*Constant integer: Round number*

*Create new Result Structure: result*

*Result:Round total = Round number*

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

*If round of i:Bet type equals round of i:result*

*Increment result:wins*

*Else if round of i:Bet type not equal round of i:result*

*Increment result:losses*

*Return result structure*

*End MakeResult*

*Start PrintResult*

*Print total rounds, wins, and losses*

*End PrintResult*

*Start BinaryData*

*Receives:*

*Round array: round*

*Result structure: result*

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

*c-string: rounds, results*

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

*output for Results:write Round total, wins, losses*

*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, banker:sum, player:sum*

*End Binary Data*

*Start MemoryClear*

*Receives:*

*Round Array: round*

*Result structure: result*

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

*delete round of i:player*

*delete round of i:banker*

*delete round array*

*delete result*

*End MemoryClear*

*Start ValueCheck*

*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*

Project Code:

/\*

\* File: main.cpp

\* Author: Jason Handen

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

\* Purpose: Project 1 Version 7, FINAL (add Random Access to bin file)

\*/

#include <iostream> // for input/output

#include <iomanip> // for output formatting

#include <ctime> // for rand function

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

using namespace std;

// Enums

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

enum suit{CLUBS,DIAMONDS,HEARTS,SPADES};

// Structures

struct Hand{

string card[3];

short value[3];

bool natural=false;

bool hit=false;

short sum;

};

struct Round{

short rndNum;

short betType;

Hand \*player;

Hand \*banker;

short result;

};

struct Result{

short rndTot;

short wins=0;

short losses=0;

};

// Function Prototypes

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

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

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

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

void dealCards(Hand\*); // deal cards (including extra) and sum

void writeCards(Hand\*); // write card name to string

void prntRound(Round\*); // print round

Result\* makeResult(Round[],const short); // make Result struct

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

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

void memClear(Round\*,Result\*); // deallocate memory

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

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

{

// seed random number generator

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

// declare file and vars

ifstream inRnd;

ifstream inRes;

ofstream outRnd;

ofstream outRes;

// initialize vars

char rounds[]="GameRounds.bin";

char results[]="GameResults.bin";

//char runtime

const short RNDMIN=1;

const short RNDMAX=20;

prntPrior(rounds,results,inRnd,inRes);

short rndNumb=promptGame(RNDMIN,RNDMAX);

Round \*round=new Round[rndNumb]; // new rounds according to input

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

Hand \*player=new Hand;

round[i].player=player; // assign pointers to rounds

Hand \*banker=new Hand;

round[i].banker=banker; // assign pointers to rounds

}

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

round[i].rndNum=i+1;

playBaccarat(round+i); // runs game until complete

}

Result \*result=makeResult(round,rndNumb);

prntResult(result);

binData(round,result,outRnd,outRes,rounds,results);

inRes.close(); // close file input

inRnd.close(); // close file input

outRes.close(); // close file output

outRnd.close(); // close file output

memClear(round,result); // deallocate memory

return 0;

}

void prntPrior(char rounds[],char results[],ifstream &inRnd,ifstream &inRes){

short response=0;

Result \*result=new Result;

Round \*round=new Round;

Hand \*play=new Hand;

Hand \*bank=new Hand;

round->banker=bank;round->player=play;

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(!inRes){

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

}else{

inRes.read((char\*)&result->rndTot,sizeof(short));

inRes.read((char\*)&result->wins,sizeof(short));

inRes.read((char\*)&result->losses,sizeof(short));

cout<<"This was the round total for the last game played: "

<<result->rndTot<<endl;

cout<<"Total wins for last game: "<<result->wins<<endl;

cout<<"Total losses for last game: "<<result->losses<<endl<<endl;

do{

inRnd.seekg(0);

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

cin>>response;

response=valCheck(response,0,result->rndTot);

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

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

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

inRnd.read((char\*)&round->result,sizeof(short));

inRnd.read((char\*)&round->banker->sum,sizeof(short));

inRnd.read((char\*)&round->player->sum,sizeof(short));

}

if(response!=0){

cout<<"For Round #"<<round->rndNum<<":\n";

if(round->betType==PLAYER){

cout<<"Player bet with ";

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

cout<<"Banker bet with ";

}else{

cout<<"Tie bet with ";

}

if(round->result==PLAYER){

cout<<"player win.\n";

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

cout<<"banker win.\n";

}else{

cout<<"tie game.\n";

}

cout<<"Player sum: "<<round->player->sum<<endl;

cout<<"Banker sum: "<<round->banker->sum<<endl<<endl;

}

}while(response!=0);

}

delete round->player;

delete round->banker;

delete round;

delete result;

}

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

short rndNumb=0;

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

cin>>rndNumb;

rndNumb=valCheck(rndNumb,RNDMIN,RNDMAX);

return rndNumb;

}

void playBaccarat(Round \*round){

round->betType=promptBet(PLAYER,TIE);

dealCards(round->player);

dealCards(round->banker);

round->player->sum=round->player->value[0]+round->player->value[1];

if(round->player->sum>9){

round->player->sum-=10;

}

round->banker->sum =round->banker->value[0] +round->banker->value[1];

if(round->banker->sum>9){

round->banker->sum-=10;

}

// checking baccarat rules

// if either hand sum is 8 or 9, both hands stand

if(round->banker->sum>7){

round->banker->natural=true;

}else{

round->banker->natural=false;

}

if(round->player->sum>7){

round->player->natural=true;

}else{

round->player->natural=false;

}

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

if(round->player->natural==false&&round->banker->natural==false){

if(round->player->sum<6){

round->player->hit=true;

round->player->sum+=round->player->value[2];

if(round->player->sum>9){

round->player->sum-=10;

}

}

}

// run through banker conditions for drawing

if(round->player->natural==false&&round->banker->natural==false){

if(round->player->hit==false&&round->banker->sum<6){

round->banker->hit=true;

round->banker->sum+=round->banker->value[2];

if(round->banker->sum>9){

round->banker->sum-=10;

}

}else if(round->player->hit==false&&round->banker->sum>5){

// banker stands

}else if(round->banker->sum<3){

round->banker->hit=true;

round->banker->sum+=round->banker->value[2];

if(round->banker->sum>9){

round->banker->sum-=10;

}

}else if(round->player->hit==true&&round->banker->sum==3

&&round->player->value[2]!=8){

round->banker->hit=true;

round->banker->sum+=round->banker->value[2];

if(round->banker->sum>9){

round->banker->sum-=10;

}

}else if(round->player->hit==true&&round->banker->sum==4

&&round->player->value[2]>1&&round->player->value[2]<8){

round->banker->hit=true;

round->banker->sum+=round->banker->value[2];

if(round->banker->sum>9){

round->banker->sum-=10;

}

}else if(round->player->hit==true&&round->banker->sum==5

&&round->player->value[2]>3&&round->player->value[2]<8){

round->banker->hit=true;

round->banker->sum+=round->banker->value[2];

if(round->banker->sum>9){

round->banker->sum-=10;

}

}else if(round->player->hit==true&&round->banker->sum==6

&&round->player->value[2]>5&&round->player->value[2]<8){

round->banker->hit=true;

round->banker->sum+=round->banker->value[2];

if(round->banker->sum>9){

round->banker->sum-=10;

}

}

}

if(round->player->sum > round->banker->sum){

round->result=PLAYER;

}else if(round->player->sum < round->banker->sum){

round->result=BANKER;

}else{

round->result=TIE;

}

prntRound(round); // print results of that 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;

}

void dealCards(Hand \*hand){

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

hand->value[i]=rand()%52; // value[2] only used if bool set to true

}

writeCards(hand);

}

void writeCards(Hand \*hand){

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

short suit=hand->value[i]/13;

// determine card face

hand->value[i]%=13;

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

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

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

hand->card[i]="Two";

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

hand->card[i]="Three";

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

hand->card[i]="Four";

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

hand->card[i]="Five";

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

hand->card[i]="Six";

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

hand->card[i]="Seven";

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

hand->card[i]="Eight";

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

hand->card[i]="Nine";

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

hand->card[i]="Ten";

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

hand->card[i]="Jack";

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

hand->card[i]="Queen";

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

hand->card[i]="King";

}

// set to proper value for summing later

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

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

}else{

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

}

// add " of " for formatting

hand->card[i]+=" of ";

// determine card suit and concatenate

switch(suit){

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

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

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

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

}

}

}

void prntRound(Round \*round){

const short WIDTH=25;

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

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

cout<<setw(WIDTH)<<left<<round->player->card[0]

<<setw(WIDTH)<<left<<round->banker->card[0]<<endl;

cout<<setw(WIDTH)<<left<<round->player->card[1]

<<setw(WIDTH)<<left<<round->banker->card[1]<<endl;

if(round->player->hit==true){

cout<<setw(WIDTH)<<left<<round->player->card[2];

if(round->banker->hit==true){

cout<<setw(WIDTH)<<left<<round->banker->card[2];

}

cout<<endl;

}else if(round->banker->hit==true){

cout<<setw(WIDTH)<<" "<<setw(WIDTH)<<left<<round->banker->card[2]<<endl;

}

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

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

if(round->player->sum>round->banker->sum){

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

}else if(round->player->sum<round->banker->sum){

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

}else{

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

}

if(round->betType==round->result){

cout<<" Won Bet!\n";

}else if(round->betType!=round->result){

cout<<" Lost Bet!\n";

}

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

cin.ignore();

cin.get();

}

Result\* makeResult(Round round[],const short rndNumb){

Result \*result=new Result;

result->rndTot=rndNumb;

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

if(round[i].betType==round[i].result){

result->wins+=1;

}else if(round[i].betType!=round[i].result){

result->losses+=1;

}

}

return result;

}

void prntResult(Result \*result){

cout<<"\n\nTotal Rounds: "<<result->rndTot<<endl;

cout<<"Won bets: "<<result->wins<<endl;

cout<<"Lost bets: "<<result->losses<<endl;

}

void binData(Round round[],Result \*result,ofstream &outRnd, ofstream &outRes,

char rounds[],char results[]){

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

outRes.write((char\*)&result->rndTot,sizeof(short));

outRes.write((char\*)&result->wins,sizeof(short));

outRes.write((char\*)&result->losses,sizeof(short));

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

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

outRnd.write((char\*)&round[i].rndNum,sizeof(short));

outRnd.write((char\*)&round[i].betType,sizeof(short));

outRnd.write((char\*)&round[i].result,sizeof(short));

outRnd.write((char\*)&round[i].banker->sum,sizeof(short));

outRnd.write((char\*)&round[i].player->sum,sizeof(short));

}

}

void memClear(Round \*round,Result \*result){

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

delete round[i].player;

delete round[i].banker;

}

delete []round;

delete result;

}

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;

}