**Beginners**

**Blackjack**

Name

Christopher Herre

Assignment

Project 1

Section

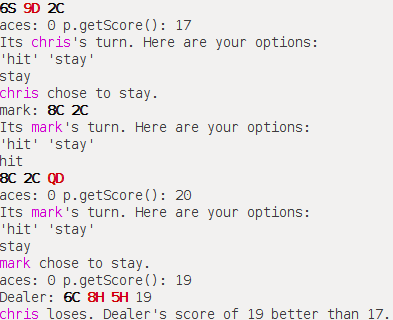
47986

Introduction

The objective of blackjack is to get a higher score than the dealer, you are not competing against other players. In this version, the rules are simplified to two options hit or stay. You are playing with poker chips as well so you can visualize your bets and see them increase/decrease as you make decisions. When the game begins, you will be asked the number of card decks you want to play with; you may enter any number between 1-5. The cards will be shuffled after this and the game will ask players for their names and bet amounts.

Then players will each be dealt two cards and may enter hit or stay to get closer to 21 than the dealer. Any score above 21 loses automatically and is called a bust. In the event the dealer and a player have the same score, its called a push and the player will neither lose nor gain any chips. If the player gets a blackjack (21 on the first two cards) the player’s bet will be matched plus another half. If the player wins without getting blackjack, the bet will just be matched.

White chips are worth $1, red $5, blue $10, and green $25. The amount each player starts with depends on the number of players playing the game. With two players, this number is $292.

The first player to enter their name goes first, and the dealer always goes last.

Summary

**Lines of code: 752**

**Functions: 11**

**Classes: 5**

I spent about a week and a half programming and debugging this game. I used the majority of STL containers and algorithms we covered in lab. I also used concepts like operator overloading, classes, Object-Oriented Programming, and more.

**STL Concepts**

The first STL concept I used was a stack to represent the poker chips. I created a class called Bank to represent the player’s collection of poker chips. This class contains four stacks that represent green, blue, red, and white poker chips. Items are pushed onto the stacks by the addReward function if the player wins, or popped off of the stacks by the subBet function if the player loses. This is all done using while loops until the proper number have been pushed/popped to correspond to what the player won or lost.

The next concept I used was a vector to represent the actual deck. I chose a vector because it plays nicely with the STL shuffle algorithm. The vector is a vector of Cards contained within a Deck class.

I used an STL pair and STL list to represent the first two cards dealt, and all subsequent hit cards respectively. I chose a list because inserting into a list is fast, and there are a small number of hits for each player so iterating the list will not take much time. These are both contained in the Player class.

In the main class, I created a method called populateVals that uses a map to map characters to shorts. This is used to tie face values of cards to their corresponding score values.

I used a queue to represent burn cards, or cards that have been removed from the deck for one reason or another. This is found in the main function.

I used deques primarily to keep track of players that are currently playing, and players that have bust and need to be held separately for the next round. These are declared in the main function.

I used a set to keep track of the players options such as hit or stay. This is useful in case I want to add more options later in the future, they are already sorted alphabetically with the set data structure upon insertion. This is found in the gameLoop function in the main class.

I used iterators to iterate through a deque of players, also in the gameLoop function.

I also used enhanced for loops to iterate over STL containers such as the set mentioned above.

References

Josuttis, Nicolai M. The C++ Standard Library Extensions: a Tutorial and Reference. Addison-Wesley, 2006.

Major Variables

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable Name** | **Description** | **Location** |
| map<char, short> | vals | Maps face values to score values. | Main function of main.cpp |
| Deck | deck | Creates and shuffles a deck of Card objects. | Main function of main.cpp |
| bool | first | Signifies the first round or not. | Main function of main.cpp |
| deque<Player> | players | The currently playing Player instances. | Main function of main.cpp |
| queue<Player> | burnCards | Cards removed from the deck for any reason. Can be added back to the original deck later on. | Main function of main.cpp |
| deque<Player> | nextRound | Players removed from the previous round for busting or finishing turn. To be added to the next round. | Main function of main.cpp |
| set<string, less<string>> | options | An alphabetically organized set of string options. Such as hit or stay; more to come. | gameLoop function of main.cpp |
| stack<char>  stack<char>  stack<char>  stack<char> | green  blue  red  white | These stacks represent a player’s poker chips. | Bank class |
| int | total | Represents the total value of all poker chips. | Bank class |
| const static short | NUM\_PLAYERS | The number of players playing the game. | Bank class |
| short | modBet | The additional bet to add to the player’s poker chips. | Bank class, addReward function |
| string | name | The player’s username. | Player class |
| Bank | bank | The player’s instance of Bank for their poker chips. | Player class |
| pair<Card, Card> | cards | Holds the player’s first two dealt cards. | Player class |
| list<Card> | hits | Doubly linked list of hit Cards. Fast insertions, small number of elements to iterate through. | Player class |
| bool | blackjack | Does the player have a score of 21 before any hit Cards are dealt? | Player class |
| short | score | The player’s score towards/over/under 21. | Player class |
| short | bet | The user inputted amount for the player’s bet. | Player class |
| short | acesSubbed | The number of aces that have been reduced from 11 to 1. To prevent counting them again. | Player class |
| char | Suit | The suit of the Card. | Card class |
| char | value | The face value of the Card. Such as King/Queen/Ten. | Card class |
| vector<Card> | deck | The underlying data structure of the Deck class. | Deck class |

Pseudocode

main:

Populate an STL map of card face values to score values

Create and shuffle a Deck of Cards

Prompt the user for the number of card decks

Display the cards being used and reshuffle the deck after

Initialize variables: first, players, burnCards, nextRound

while true:

Call gameLoop

while burnCards is not empty:

Remove all burn cards and push them to the original deck

Reshuffle deck

gameLoop:

Loop three times:

Print “----------------”

Call initPlayers

Instantiate a dealer Player

Print the dealer’s hand

Create an STL set of options to choose from

Insert “hit” and “stay” as options

Call handleOptions

Call handleDealer

Call handleOutcome

initPlayers:

If first is true

Prompt all players for bet amounts

Setup nextRound deque<Player> for all in players deque

Else

For all players in nextRound deque<Player>

Re-initialize player for the next round

Setup players deque<Player> for all in nextRound deque

handleOutcome:

For all players

If dealer has blackjack and player does not

Player loses

Else if player has blackjack or player’s score > dealer’s and less than or equal to 21 or dealer has bust and player hasn’t

Player wins

Else if player’s score equals dealer’s score that is less than or equal to 21

Player pushes

Else

Player loses

Print player’s poker chips remaining

handleDealer:

If dealer’s score is less than 16

Dealer receives a card from the deck

aces = amount of aces dealer has

If aces >= 1 and dealer’s score > 21

While dealer’s score > 21 and get subtracted aces != aces

Subtract 10 from dealer’s score

Set aces subtracted++

Print subtracting 10 from dealer’s score

Print all of dealer’s cards

if dealer has blackjack

Print dealer has blackjack

handleOptions:

If player hits are less than or equal to 0

Print cards in hand

If player has blackjack

Print player has blackjack

Print set of options

Prompt for option input

If input is not valid

Re-prompt for option input

Return

If input equals “hit”

Call handleHit

Else if input equals “stay”

Print player chose to stay

Return;

Else

Print input error

Exit with error

handleHit:

Player receives a card from the deck

Print all of player’s cards

aces = amount of aces player has

If aces >= 1 or player score > 21

While player score > 21 and get subtracted aces != aces

Print subtracting 10 from dealer’s score

Set aces subtracted++

Subtract 10 from player’s score

If player’s score > 21

Print player has bust

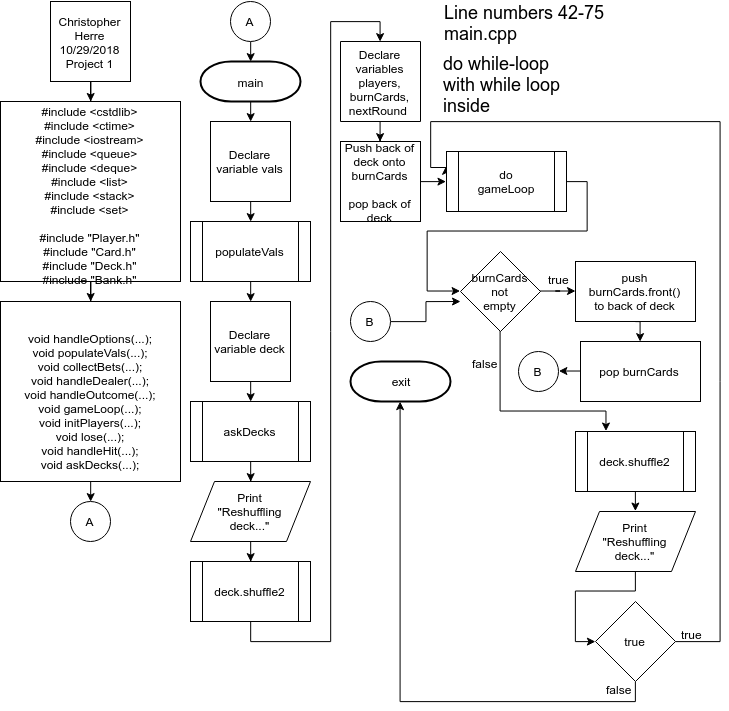
Print player’s poker chips remaining

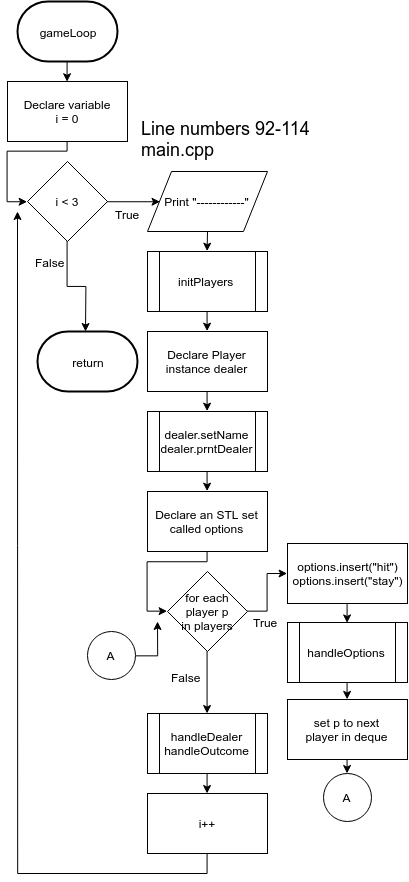
Remove player from players list

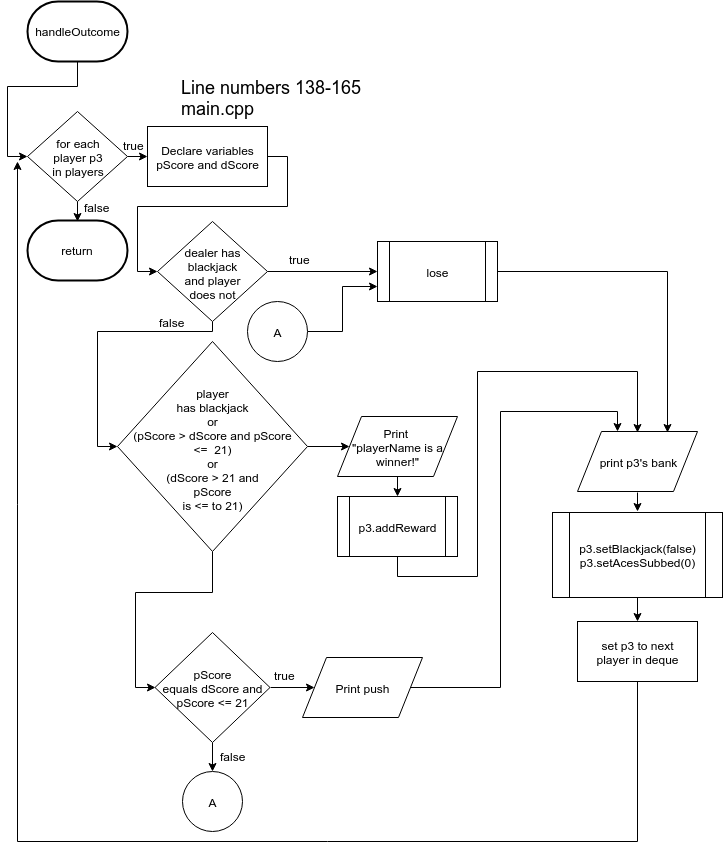
Else

Call handleOptions

Flowchart







Code

/\*

\* File: main.cpp

\* Author: chris

\*

\* Created on October 20, 2018, 7:29 PM

\*/

#include <cstdlib>

#include <ctime>

#include <iostream>

#include <queue>

#include <deque>

#include <list>

#include <stack>

#include <set>

#include "Player.h"

#include "Card.h"

#include "Deck.h"

#include "Bank.h"

using namespace std;

void handleOptions(Player&, set<string, less<string>>&, Deck&,

map<char, short>&, deque<Player>&,

queue<Card>&);

void populateVals(map<char, short>&);

void collectBets(Deck&, deque<Player>&, map<char, short>&,

queue<Card>&);

void handleDealer(Deck&, Player&, map<char, short>&,

queue<Card>&);

void handleOutcome(Player& dealer, deque<Player>& players);

void gameLoop(bool&, Deck&, deque<Player>&, deque<Player>&,

map<char, short>&, queue<Card>&);

void initPlayers(bool&, Deck&, deque<Player>&, deque<Player>&,

map<char, short>&, queue<Card>&);

void lose(Player&, short, short);

void handleHit(Player&, set<string, less<string>>&, Deck&,

map<char, short>&, deque<Player>&, queue<Card>&);

void askDecks(Deck&);

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

map<char, short> vals;

// populate a map of face values to score values

populateVals(vals);

// create a deck and shuffle

Deck deck;

// ask the user how many decks

// and add to current deck

askDecks(deck);

cout << endl << BOLDCYAN << "Reshuffling deck..." << RESET << endl;

deck.shuffle2();

cout << endl;

// the first round or not?

bool first = true;

// currently playing

deque<Player> players;

// disgarded cards

queue<Card> burnCards;

deque<Player> nextRound;

burnCards.push(deck.getDeck().back());

// pop the back of the deck

deck--;

while (true) {

gameLoop(first, deck, players, nextRound, vals, burnCards);

// deck is getting low so recreate it

while (!burnCards.empty()) {

deck.getDeck().push\_back(burnCards.front());

burnCards.pop();

}

deck.shuffle2();

cout << BOLDCYAN << "Reshuffling deck..." << RESET << endl;

}

return 0;

}

void askDecks(Deck& deck) {

cout << "How many decks do you want to play with? ";

short numDecks;

cin >> numDecks;

if (numDecks < 1 || numDecks > 5) {

cout << "Input error!" << endl;

exit(0);

}

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

Deck deck2;

deck.addDeck(deck2);

}

deck.prntDeck();

}

void gameLoop(bool& first, Deck& deck, deque<Player>& players,

deque<Player>& nextRound, map<char, short>& vals,

queue<Card>& burnCards) {

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

// denotes a new round

cout << "-------------------" << endl;

initPlayers(first, deck, players, nextRound, vals, burnCards);

Player dealer(deck, true, vals, burnCards);

dealer.setName("Dealer");

dealer.prntDealer();

// sort options alphabetically upon insert with a set

set<string, less<string>> options;

for (Player& p : players) {

options.insert("hit");

options.insert("stay");

handleOptions(p, options, deck, vals, players, burnCards);

}

// dealers turn is last

handleDealer(deck, dealer, vals, burnCards);

// decide win, lose, or push

handleOutcome(dealer, players);

}

}

void initPlayers(bool& first, Deck& deck, deque<Player>& players,

deque<Player>& nextRound, map<char, short>& vals,

queue<Card>& burnCards) {

if (first) {

collectBets(deck, players, vals, burnCards);

nextRound = players;

first = false;

} else {

for (Player &p : nextRound) {

p.init(deck, false, vals, burnCards);

}

players = nextRound;

}

}

void lose(Player& p3, short pScore, short dScore) {

cout << MAGENTA << p3.getName() << RESET

<< " loses. Dealer's score of " << dScore

<< " better than " << pScore << "." << endl;

p3.getBank().subBet(p3.getBet());

}

void handleOutcome(Player& dealer, deque<Player>& players) {

for (Player& p3 : players) {

short pScore = p3.getScore();

short dScore = dealer.getScore();

if (dealer.isBlackjack() && !p3.isBlackjack()) {

lose(p3, pScore, dScore);

} else if (p3.isBlackjack() || (pScore > dScore && pScore <= 21) ||

(dScore > 21 && pScore <= 21)) {

cout << MAGENTA << p3.getName() << RESET << " is a winner!" << endl;

p3.getBank().addReward(p3.getBet(), p3.isBlackjack() ? 1 : 2);

} else if (pScore == dScore && pScore <= 21) {

cout << "PUSH! " << MAGENTA << p3.getName() << RESET

<< "'s score is equal to dealers score of "

<< dScore << endl;

// player does not lose bet or gain

} else {

lose(p3, pScore, dScore);

}

Bank bank = p3.getBank();

cout << "Printing " << MAGENTA << p3.getName() << RESET

<< "'s bank..." << endl;

p3.getBank().prntBank(bank.getRed(), bank.getBlue(),

bank.getGreen(), bank.getWhite());

// reset some variables

p3.setBlackjack(false);

p3.setAcesSubbed(0);

}

}

void handleDealer(Deck& deck, Player& dealer, map<char, short>& vals,

queue<Card>& burnCards) {

if (dealer.getScore() < 16) {

dealer.getHits().push\_back(deck.getDeck().back());

dealer.setScore(dealer.getScore()

+ vals[deck.getDeck().back().getValue()]);

burnCards.push(deck.getDeck().back());

deck--;

// deal with aces having two values, 11 or 1

int aces = dealer.countAces();

cout << "aces: " << aces << " p.getScore(): " << dealer.getScore()

<< endl;

if (aces >= 1 && dealer.getScore() > 21) {

while (dealer.getScore() > 21 && dealer.getAcesSubbed() != aces) {

cout << "Subtracting 10 from dealer's score" << endl;

dealer.setAcesSubbed(dealer.getAcesSubbed() + 1);

cout << "aces: " << aces<< " acesSubbed: "

<< dealer.getAcesSubbed() << endl;

dealer.setScore(dealer.getScore() - 10);

}

}

}

// print dealer's hand and score

cout << "Dealer: " << dealer.getCards().first.toString() << " "

<< dealer.getCards().second.toString() << " ";

for (Card c : dealer.getHits()) {

cout << c.toString() << " ";

}

cout << dealer.getScore() << endl;

if (dealer.isBlackjack()) {

cout << "Dealer has blackjack!" << endl;

}

}

void collectBets(Deck& deck, deque<Player>& players, map<char, short>& vals,

queue<Card>& burnCards) {

for (short i = 0; i < Bank::NUM\_PLAYERS; i++) {

cout << "Enter player " << i + 1 << "'s name: ";

string input;

short bet = 0;

cin >> input;

if (input == "Dealer") {

cout << "Input error!" << endl;

exit(1);

}

Player p(deck, false, vals, burnCards);

p.setName(input);

Bank bank = p.getBank();

cout << "Printing " << MAGENTA << p.getName() << RESET

<< "'s bank..." << endl;

p.getBank().prntBank(bank.getRed(), bank.getBlue(),

bank.getGreen(), bank.getWhite());

cout << "Place your bet 1-" << p.getBank().getTotal() << ": ";

cin >> bet;

if (bet < 1 || bet > p.getBank().getTotal()) {

cout << "Input error!" << endl;

exit(1);

}

p.setBet(bet);

players.push\_back(p);

}

}

void populateVals(map<char, short>& vals) {

vals.insert(pair<char, short>('K', 10));

vals.insert(pair<char, short>('J', 10));

vals.insert(pair<char, short>('Q', 10));

vals.insert(pair<char, short>('T', 10));

vals.insert(pair<char, short>('9', 9));

vals.insert(pair<char, short>('8', 8));

vals.insert(pair<char, short>('7', 7));

vals.insert(pair<char, short>('6', 6));

vals.insert(pair<char, short>('5', 5));

vals.insert(pair<char, short>('4', 4));

vals.insert(pair<char, short>('3', 3));

vals.insert(pair<char, short>('2', 2));

vals.insert(pair<char, short>('1', 11));

}

void handleOptions(Player& p, set<string, less<string>>& options, Deck& deck,

map<char, short>& vals, deque<Player>& players,

queue<Card>& burnCards) {

if (p.getHits().size() <= 0) {

cout << p.getName() << ": " << p.getCards().first.toString() << " "

<< p.getCards().second.toString() << endl;

if (p.isBlackjack()) {

cout << p.getName() << " has blackjack!" << endl;

}

}

cout << "Its " << MAGENTA << p.getName() << RESET

<< "'s turn. Here are your options:\n";

for (string o : options) {

cout << "'" << o << "' ";

}

cout << endl;

string input;

cin >> input;

if (!(options.find(input) != options.end())) {

cout << "You entered an invalid option!" << endl;

handleOptions(p, options, deck, vals, players, burnCards);

return;

}

if (input == "hit") {

handleHit(p, options, deck, vals, players, burnCards);

} else if (input == "stay") {

cout << MAGENTA << p.getName() << RESET << " chose to stay." << endl;

return;

} else {

cout << "Input error!" << endl;

exit(1);

}

}

void handleHit(Player& p, set<string, less<string>>& options, Deck& deck,

map<char, short>& vals, deque<Player>& players,

queue<Card>& burnCards) {

p.setScore(p.getScore() + vals[deck.getDeck().back().getValue()]);

p.getHits().push\_back(deck.getDeck().back());

burnCards.push(deck.getDeck().back());

deck--;

cout << p.getCards().first.toString() << " "

<< (p.getName() == "Dealer" ? "\*\*"

: p.getCards().second.toString()) << " ";

for (Card c : p.getHits()) {

cout << c.toString() << " ";

}

cout << endl;

int aces = p.countAces();

cout << "aces: " << aces << " p.getScore(): " << p.getScore() << endl;

if (aces >= 1 && p.getScore() > 21) {

while (p.getScore() > 21 && p.getAcesSubbed() != aces) {

cout << "Subtracting 10 from score" << endl;

p.setAcesSubbed(p.getAcesSubbed() + 1);

cout << "aces: " << aces<< " acesSubbed: " << p.getAcesSubbed()

<< endl;

p.setScore(p.getScore() - 10);

}

}

if (p.getScore() > 21) {

cout << "BUST! Your score is " << p.getScore() << ". You're out!"

<< endl;

p.getBank().subBet(p.getBet());

cout << "Printing " << MAGENTA << p.getName() << RESET

<< "'s bank..." << endl;

Bank bank = p.getBank();

p.getBank().prntBank(bank.getRed(), bank.getBlue(),

bank.getGreen(), bank.getWhite());

// remove player from playing list if bust

// iterators used here

auto it = players.begin();

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

Player& p2 = players[i];

if (p2.getName() == p.getName()) break;

it++;

}

players.erase(it);

} else {

handleOptions(p, options, deck, vals, players, burnCards);

}

}

/\*

\* File: Bank.h

\* Author: chris

\*

\* Created on October 18, 2018, 6:46 AM

\*/

#ifndef BANK\_H

#define BANK\_H

#include <cstdlib>

#include <iostream>

#include <stack>

#include "Player.h"

#include "Colors.h"

using namespace std;

class Bank {

private:

// STL stacks used as poker chips

stack<char> green; // 25

stack<char> blue; // 10

stack<char> red; // 5

stack<char> white; // 1

int total;

public:

const static short NUM\_PLAYERS = 2;

Bank() {

for (int i = 0; i < 8 / NUM\_PLAYERS; i++)

green.push('G');//25

for (int i = 0; i < 16 / NUM\_PLAYERS; i++)

blue.push('B'); //10

for (int i = 0; i < 32 / NUM\_PLAYERS; i++)

red.push('R'); //5

for (int i = 0; i < 64 / NUM\_PLAYERS; i++)

white.push('W');//1

total = (green.size() \* 25) + (blue.size() \* 10)

+ (red.size() \* 5) + white.size();

}

void subBet(short bet) {

while (!green.empty() && bet / 25 > 0) {

green.pop();

bet-=25;

total-=25;

}

while (!blue.empty() && bet / 10 > 0) {

blue.pop();

bet-=10;

total-=10;

}

while (!red.empty() && bet / 5 > 0) {

red.pop();

bet-=5;

total-=5;

}

while (!white.empty() && bet / 1 > 0) {

white.pop();

bet--;

total--;

}

}

void addReward(short bet, short type) {

short modBet = bet;

if (type == 1) { // normal

modBet = bet \* 1.5;

} else if (type == 2) { // blackjack

modBet = bet;

}

bet = modBet;

cout << "modBet: " << modBet << endl;

while (!green.empty() && bet / 25 > 0) {

green.push('G');

bet-=25;

total+=25;

}

while (!blue.empty() && bet / 10 > 0) {

blue.push('B');

bet-=10;

total+=10;

}

while (!red.empty() && bet / 5 > 0) {

red.push('R');

bet-=5;

total+=5;

}

while (!white.empty() && bet / 1 > 0) {

white.push('W');

bet--;

total++;

}

}

// not passed by reference!

void prntBank(stack<char> r, stack<char> b, stack<char> g, stack<char> w) {

int i;

for (i = 0; !g.empty(); i++) {

if (!g.empty()) {

cout << BOLDGREEN << g.top() << RESET;

g.pop();

}

}

cout << " " << i << endl;

for (i = 0; !b.empty(); i++) {

if (!b.empty()) {

cout << BOLDBLUE << b.top() << RESET;

b.pop();

}

}

cout << " " << i << endl;

for (i = 0; !r.empty(); i++) {

if (!r.empty()) {

cout << BOLDRED << r.top() << RESET;

r.pop();

}

}

cout << " " << i << endl;

for (i = 0; !w.empty(); i++) {

if (!w.empty()) {

cout << BGBLACK << BOLDWHITE << w.top() << RESET;

w.pop();

}

}

cout << " " << i << endl;

cout << "Total: " << total << endl;

}

stack<char>& getRed() {

return red;

}

stack<char>& getWhite() {

return white;

}

stack<char>& getGreen() {

return green;

}

stack<char>& getBlue() {

return blue;

}

short getTotal() {

return total;

}

};

#endif /\* BANK\_H \*/

/\*

\* File: Card.h

\* Author: chris

\*

\* Created on October 17, 2018, 11:26 PM

\*/

#ifndef CARD\_H

#define CARD\_H

#include <cstdlib>

#include <string>

#include <sstream>

#include "Colors.h"

using namespace std;

class Card {

private:

char suit;

char value;

public:

Card() { };

Card(char suite, char value) {

this->suit = suite;

this->value = value;

}

string toString() {

stringstream ss;

string color = suit == 'H' || suit == 'D' ? BOLDRED : BOLDBLACK;

ss << color << value << suit << RESET;

return ss.str();

}

char getSuit() {

return suit;

}

char getValue() {

return value;

}

bool operator==(Card card) {

if (this->value == card.value && this->suit == card.suit)

return true;

else

return false;

}

bool operator!=(Card card) {

if (this->value == card.value && this->suit == card.suit)

return false;

else

return true;

}

};

#endif /\* CARD\_H \*/

#ifndef COLORS\_H

#define COLORS\_H

#define RESET "\033[0m"

#define BLACK "\033[30m" /\* Black \*/

#define RED "\033[31m" /\* Red \*/

#define GREEN "\033[32m" /\* Green \*/

#define YELLOW "\033[33m" /\* Yellow \*/

#define BLUE "\033[34m" /\* Blue \*/

#define MAGENTA "\033[35m" /\* Magenta \*/

#define CYAN "\033[36m" /\* Cyan \*/

#define WHITE "\033[37m" /\* White \*/

#define BOLDBLACK "\033[1m\033[30m" /\* Bold Black \*/

#define BOLDRED "\033[1m\033[31m" /\* Bold Red \*/

#define BOLDGREEN "\033[1m\033[32m" /\* Bold Green \*/

#define BOLDYELLOW "\033[1m\033[33m" /\* Bold Yellow \*/

#define BOLDBLUE "\033[1m\033[34m" /\* Bold Blue \*/

#define BOLDMAGENTA "\033[1m\033[35m" /\* Bold Magenta \*/

#define BOLDCYAN "\033[1m\033[36m" /\* Bold Cyan \*/

#define BOLDWHITE "\033[1m\033[37m" /\* Bold White \*/

#define INVERSE "\033[7m" /\* Swap Bacground and Text colors \*/

#define UNDERLINE "\033[4m" /\* Underline Single \*/

#define BGBLACK "\033[40m" /\* BLACK Background \*/

#define BGRED "\033[41m" /\* RED Background \*/

#define BGGREEN "\033[42m" /\* GREEN Background \*/

#define BGYELLOW "\033[43m" /\* YELLOW Background \*/

#define BGBLUE "\033[44m" /\* BLUE Background \*/

#define BGMAGENTA "\033[45m" /\* MAGENTA Background \*/

#define BGCYAN "\033[46m" /\* CYAN Background \*/

#define BGWHITE "\033[47m" /\* WHITE Background \*/

#endif /\* COLORS\_H \*/

/\*

\* File: Deck.h

\* Author: chris

\*

\* Created on October 17, 2018, 11:52 PM

\*/

#ifndef DECK\_H

#define DECK\_H

#include <cstdlib>

#include <vector>

#include <algorithm>

#include "Card.h"

class Deck {

private:

const short SUITES = 4;

const short VALUES = 13;

vector<Card> deck;

public:

Deck() {

const char suites[SUITES] = {'H','S','C','D'};

const char values[VALUES] = {'1','2','3','4','5','6','7','8','9',

'T','J','K','Q'};

for (int s = 0; s < SUITES; s++) {

for (int v = 0; v < VALUES; v++) {

Card card(suites[s], values[v]);

this->deck.push\_back(card);

}

}

shuffle2();

}

void shuffle2() {

default\_random\_engine dr;

dr.seed(static\_cast<double>(time(0)));

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

shuffle(this->deck.begin(), this->deck.end(), dr);

}

void addDeck(Deck deck2) {

for (short i = 0; i < 52; i++) {

deck.push\_back(deck2[i]);

}

}

void prntDeck() {

for (int i = 0; i < deck.size(); i++) {

if (i > 0 && i % 10 == 0) cout << endl;

cout << this->getDeck()[i].toString() << " ";

}

}

Card& operator[] (short index) {

return this->deck[index];

}

vector<Card>& operator--(int i) {

vector<Card>& d2 = deck;

d2.pop\_back();

return d2;

}

vector<Card>& getDeck() {

return deck;

}

};

#endif /\* DECK\_H \*/

/\*

\* File: Player.h

\* Author: rcc

\*

\* Created on October 17, 2018, 6:13 PM

\*/

#ifndef PLAYER\_H

#define PLAYER\_H

#include <cstdlib>

#include <iostream>

#include <string>

#include <stack>

#include <map>

#include <list>

#include "Bank.h"

#include "Deck.h"

using namespace std;

class Player {

private:

string name;

Bank bank;

pair<Card, Card> cards;

list<Card> hits;

bool blackjack = false;

short score = 0, bet = 0, acesSubbed = 0;

public:

Player(Deck& deck, bool dealer, map<char, short>& vals,

queue<Card>& burnCards) {

init(deck, dealer, vals, burnCards);

}

void init(Deck& deck, bool dealer, map<char, short>& vals,

queue<Card>& burnCards) {

hits.clear();

blackjack = false;

acesSubbed = 0;

score = 0;

// treat the back as the front

burnCards.push(deck.getDeck().back());

cards.first = deck.getDeck().back();

deck--;

burnCards.push(deck.getDeck().back());

cards.second = deck.getDeck().back();

deck--;

score += vals[cards.first.getValue()];

score += vals[cards.second.getValue()];

int aces = countAces();

if (aces > 1 && acesSubbed <= aces) {

acesSubbed++;

cout << "aces: " << aces<< " acesSubbed: " << acesSubbed << endl;

cout << "Subtracting 10 from score" << endl;

setScore(getScore() - 10);

}

if ((cards.first.getValue() == '1'

&& vals[cards.second.getValue()] == 10)

|| (cards.second.getValue() == '1'

&& vals[cards.first.getValue()] == 10)) {

score = 21;

blackjack = true;

}

}

int countAces() {

list<Card> aces;

Card first = cards.first;

Card second = cards.second;

if (first.getValue() == '1')

aces.push\_back(first);

if (second.getValue() == '1')

aces.push\_back(second);

for (Card c : hits) {

if (c.getValue() == '1') {

aces.push\_back(c);

}

}

return aces.size();

}

void prntDealer() {

cout << "Dealer: " << cards.first.toString() << " \*\*" << endl;

}

string getName() {

return name;

}

void setName(string name) {

this->name = name;

}

Bank& getBank() {

return bank;

}

pair<Card, Card>& getCards() {

return cards;

}

list<Card>& getHits() {

return hits;

}

short getScore() {

return score;

}

void setScore(short score) {

this->score = score;

}

short getBet() {

return bet;

}

void setBet(short bet) {

this->bet = bet;

}

bool isBlackjack() {

return blackjack;

}

void setBlackjack(bool blackjack) {

this->blackjack = blackjack;

}

short getAcesSubbed() {

return acesSubbed;

}

void setAcesSubbed(short acesSubbed) {

this->acesSubbed = acesSubbed;

}

};

#endif /\* PLAYER\_H \*/