Project 1 Write Up John-Paul McDonald CIS17A 11/14/2021

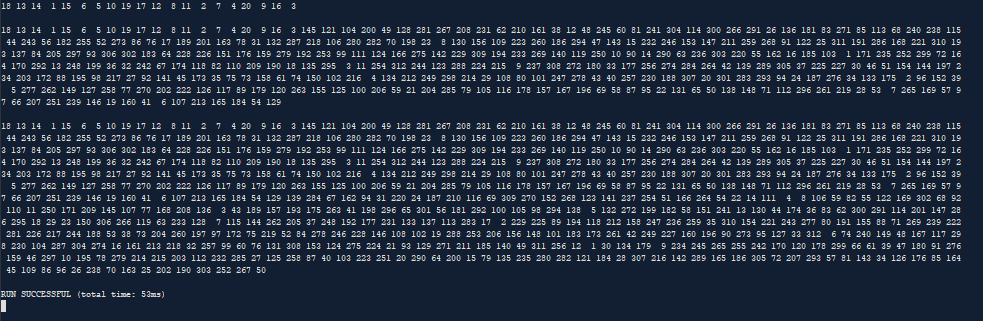
Introduction:

I created a 6 deck blackjack game. It’s currently missing the ability to split doubles, but the program allows for single player blackjack. When 6 decks are added to the stack of cards the dealer is holding, a card is initially burned and a ledger is created of the card order. The user plays the first hand, if you bust you lose. If you stand, the dealer plays their turn. The dealer is forced to stand at 17 or higher. When the number of cards in the dealer’s stack drops below 100, 6 more decks are shuffled into the game. When play is concluded, the game will report back to you which games you won or lost.

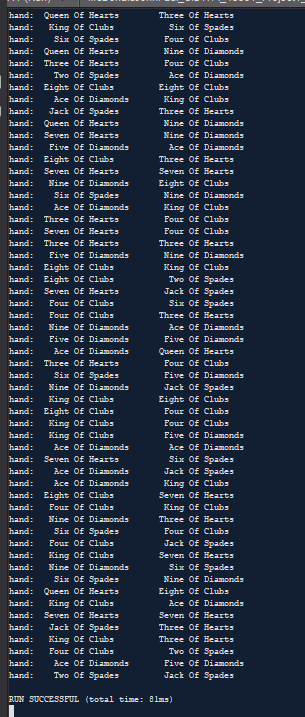
Summary:

The program is 395 lines, 3 structures, 10 functions, 2 sets of enumerations, 1 array of structures, 1 array of bools. Creating the program to play blackjack was fun and interesting, I still want to add a betting system, some classes for multiple players, and a splitting function so you can play blackjack as if you were in vegas. I’d like to take this a step further and implement a GUI. I spent too much time creating the game itself without regards to the required checklist. A lot of the checklist items didn’t have a use in my program, so I had to create instances that would demonstrate their uses. This lead me to many dead ends where the implementations felt meaningless and just made the code worse.

In version 1 I implemented the shuffling function and adding the shuffled cards into a dealer’s deck.

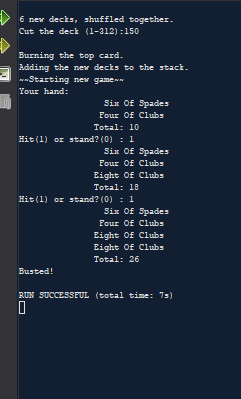


This version was ‘finished’ when I successfully was able to shuffle new decks without shuffling the preexisting decks.

Version 2 was my demonstration of my ability to Display the cards successfully. Here I used enumerations to take a 312 card deck and output the results as specific cards and suits.   


The 312 cards of the original deck are all individually numbered. I figured it didn’t matter which deck the specific card came from, so to ensure the cards are randomly sorted I shuffled all 312 numbers and then I output cards by N%4 to give me the suit, and N%12 to display the value of the card.

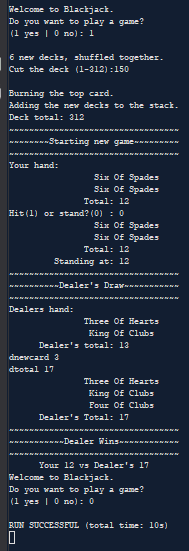
Version 3 is a single instance of playing the game. In this version, your cards are outputted, and it declares if you’ve won or lost upon completion.



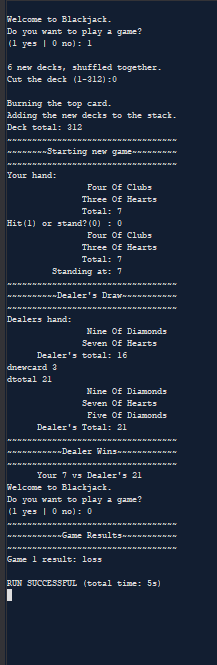
This stage involved formatting the game and implementing the rules, albeit not including splitting or betting.

Version 4

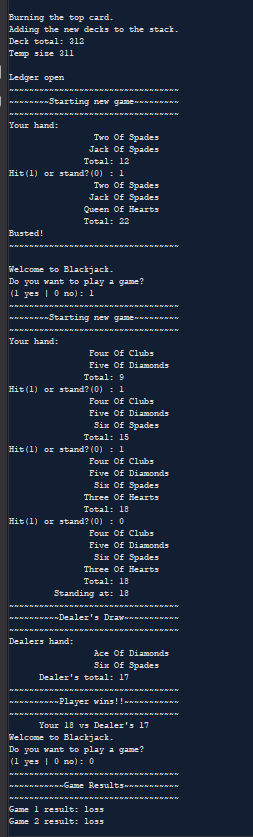
Version 4 includes the first working version of the Dealer. In this version you play a single hand, if you bust the game ends, but if you stand the dealer will play their turn. Upon completion of the game, the program will declare a winner.

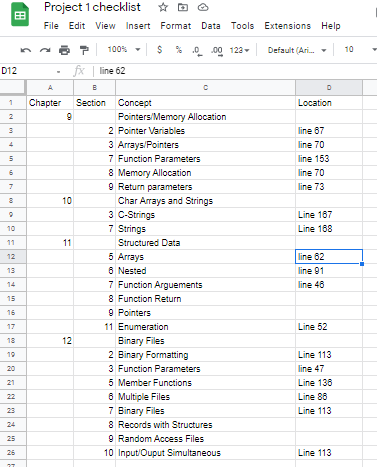


Version 5 is when I completed the ability for the game to loop & display game results. After that quick implementation, I started to work on implementing some of the checklist. Primarily converting the vectors I was holding into structures, and converting some variables to pointers.



Version 6 was all about me trying to find a use for the structures I wanted to use. Versions v6 and v7 go together hand in hand in that I continuously deleted and rebuilt the ways I wanted to handle structures and how I could pass them between functions. I didn’t really have a need for this, so I was trying to create a need or substitute an already working idea with a structured idea. A significant portion of versions 6 and 7 were a waste of time. So much so, at one point I stopped working on v7 entirely and started a new version 7.1 from an iteration of v6. This 7.1 is the final version, but it would go through many subversions before being submitted. The primary initial focus of v7.1 was to move away from trying to mess with my initial functions built without structure implementation and focus on creating a new function with structures. This is how I ended up creating file outputs that pass their contents through structures. There isn’t really a point to this, but it was an attempt to satisfy the checklist.



The final version will tell you how many cards are in the stack when new cards are shuffled, the total added is 311 because the first card is burned. After the games conclude, it will tell you the results of each game instance. When the cards are shuffled, a ledger of the entire stack is outputted to a file. When the game ends, the results are outputted to a binary file. 

Here’s a checklist of the things I was able to add and the lines you may find them on. I omitted the concepts that did not yield points.

References:

line 77 I was given the idea that I could reallocate the size of my boolean array to fit the number of games played.

Line 86/112 I found a neat call that allows me to give unique names to my files by formatting the file name based on the time it’s called.

Lines 386-395 I asked for help from you for the core engine behind my shuffle function. I modified it slightly so that it shuffles only the cards I want it to, and ignores the cards already in the stack.

/\*

\* file:

\* Author: John-Paul McDonald

\* Date:

\* Purpose:

\*/

//System Libraries

#include <iostream>

#include <vector>

#include <cstdlib>

#include <iomanip>

#include <string>

#include <fstream>

#include <stdio.h>

#include <time.h>

using namespace std;

//User Libraries

//Global Constants

//Universal Math, Physics, Conversions, Higher Dimensions

//Prototypes

struct CardHolder{

vector<int> thedeck;

};

struct CardSuits{

int specificcard;

int thesuit;

int thevalue;

};

struct CardOrder{

int cardnum;

CardSuits specifics;

};

void addDecks(vector<int> &);

void Shuffle(vector<int> &);

bool Game(vector<int>&,vector<int>&);

void CardDisplay(int,int);

void Deal(vector<int>&,vector<int>&);

int Check(vector<int>);

void Hitme(vector<int>&,vector<int>&);

int \*getPlays(int \*);

void getList(CardHolder,CardOrder[]);

void showFile(fstream&);

enum Card {Ace,Two,Three,Four,Five,Six,Seven,Eight,Nine,Jack,Queen,King};

enum Suit {OfDiamonds,OfSpades,OfHearts,OfClubs};

//Execution Begins Here

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

//Initialize Random Number Seed

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

//variables

CardHolder deck;

CardOrder list[600];

vector<int> hand;

bool playing = true;

bool playerw = false;

int plays=0;

int \*ptr = nullptr;

int numgames=0;

bool \*wins = nullptr;

wins = new bool[numgames];

//Processing

while(playing==true){

ptr=getPlays(&plays);

cin.clear();

ptr = &plays;

if(\*ptr>=1){

wins=(bool\*) realloc (wins,numgames); //if you play too many games I think this steps on the toes of your memory

if(deck.thedeck.size()<100){

addDecks(deck.thedeck);

getList(deck,list);

int listnum=list[0].cardnum;

fstream dataFile;

time\_t t = time(0);

struct tm \* now = localtime(&t);

char buffer[80];

strftime(buffer,80,"Ledger Creation %r.txt",now);

dataFile.open(buffer,ios::out);

if(dataFile.is\_open()){

cout<<"\nLedger open\n";

for(int j=1;j<listnum+1;j++){

dataFile<<"Card "<<list[j].cardnum<<": "<<list[j].specifics.specificcard<<" suit: "<<list[j].specifics.thesuit<<" value: "<<list[j].specifics.thevalue<<endl;

}

}

dataFile.close();

}

playerw=Game(deck.thedeck,hand);

wins[numgames]=playerw;

playerw=false;

numgames++;

hand.clear();

}

if(plays<=0){

playing=false;

}

}

//outputs

cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n~~~~~~~~~~~Game Results~~~~~~~~~~~\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

fstream outFile;

time\_t t = time(0);

struct tm \* now = localtime(&t);

char buffer1[80];

strftime(buffer1,80,"Win Record %r.bin",now);

outFile.open(buffer1,ios::binary | ios::out);

if(outFile.is\_open()){

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

outFile<<"Game "<<i+1<<" result:";

wins[i]?outFile<<" win" : outFile<<" loss";

outFile<<endl;

}

if(numgames == 0){

outFile<<"No games to report.";

}

}

outFile.close();

fstream myfile;

myfile.open ("testing.txt");

if(myfile.is\_open()){

showFile(myfile);

}

myfile.close();

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

cout<<"Game "<<i+1<<" result:";

wins[i]?cout<<" win" : cout<<" loss";

cout<<endl;

}

if(numgames == 0){

cout<<"No games to report.";

}

//Clean up and exit

delete [] wins;

wins = nullptr;

return 0;

}

void showFile(fstream& file){

string read1;

while(file >> read1){

cout<< read1 << endl;

}

}

void getList(CardHolder i,CardOrder \*temp2){

int temp=i.thedeck.size();

cout<<"Temp size "<<temp<<endl;

temp2[0].cardnum=temp;

for(int j=1;j<temp+1;j++){

temp2[j].cardnum=j;

temp2[j].specifics.specificcard=i.thedeck[j];

temp2[j].specifics.thesuit=temp2[j].specifics.specificcard%4;

temp2[j].specifics.thevalue=temp2[j].specifics.specificcard%12;

}

}

int \*getPlays(int \*plays){

char greeting[22] = "Welcome to Blackjack.";

string question = "\nDo you want to play a game?\n(1 yes | 0 no): ";

cout<<"\n"<<greeting<<question;

cin>> \*plays;

return plays;

}

bool Game(vector<int>& deck,vector<int>& hand){

vector<int> dhand;

bool dwinner = false;

bool winner = false;

bool ingame = true;

bool bust = false;

bool dbust = false;

int hit = 1;

int total=0;

int dtotal=0;

int newcard=0;

int dnewcard=0;

do{

cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n~~~~~~~~Starting new game~~~~~~~~~\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

hand.clear();

Deal(hand,deck);

cout<<"Your hand: \n";

CardDisplay(hand[0]%12,hand[0]%4);

CardDisplay(hand[1]%12,hand[1]%4);

total=Check(hand);

if(total==21){

ingame=false;

winner=true;

}

cout<<setw(22)<<"Total: "<<total<<endl;

while(total<=20&&ingame==true){

cout<<"Hit(1) or stand?(0) : ";

cin>>hit;

if(hit>0){

Hitme(hand,deck);

}

if(hit==0){

ingame=false;

}

newcard=hand.size();

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

CardDisplay(hand[i]%12,hand[i]%4);

}

total=Check(hand);

cout<<setw(22)<<"Total: "<<total<<endl;

if(total==21){

ingame=false;

winner=true;

}

if(total>21){

ingame=false;

bust=true;

}

}

}while(ingame==true);

if(bust==true){

cout<<"Busted!"<<endl;

}

if(winner==true){

cout<<"Winner Winner Chicken Dinner! 21!\n";

}

if(bust!=true&&winner!=true){

cout<<setw(22)<<"Standing at: "<<total<<endl;

cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n~~~~~~~~~~Dealer's Draw~~~~~~~~~~~\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

cout<<"Dealers hand: \n";

Deal(dhand,deck);

CardDisplay(dhand[0]%12,dhand[0]%4);

CardDisplay(dhand[1]%12,dhand[1]%4);

dtotal=Check(dhand);

cout<<setw(22)<<"Dealer's total: "<<dtotal<<endl;

while(dtotal<17){

Hitme(dhand,deck);

dtotal=Check(dhand);

dnewcard=dhand.size();

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

CardDisplay(dhand[i]%12,dhand[i]%4);

}

cout<<setw(22)<<"Dealer's Total: "<<dtotal<<endl;

}

if(dtotal==21||dtotal>total){

dwinner=true;

}

if(dtotal>21){

dbust=true;

dwinner=false;

}

}

cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

if(dwinner==true){

cout<<"~~~~~~~~~~~Dealer Wins~~~~~~~~~~~~\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

cout<<" Your "<<total<<" vs Dealer's "<<dtotal;

}

if(dbust==true){

cout<<"~~~~~~~~~~~Dealer Bust~~~~~~~~~~~~\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

cout<<" Dealer's total "<<dtotal<<"."<<endl;

}

if(total==dtotal){

cout<<"~~~~~~~~~~~~Draw game~~~~~~~~~~~~~\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

cout<<" Your "<<total<<" = Dealer's "<<dtotal;

}

if(bust!=true&&total>dtotal){

cout<<"~~~~~~~~~~Player wins!!~~~~~~~~~~~\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";

if(dtotal!=0){cout<<" Your "<<total<<" vs Dealer's "<<dtotal;}

}

return winner;

}

void Hitme(vector<int>&hand,vector<int>&deck){

int temp=deck[0]%52;

hand.push\_back(temp);

deck.erase(deck.begin());

}

int Check(vector<int> hand){

int handsize = hand.size();

int count=0;

int countace=0;

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

if(hand[i]%12==0){

count+=1;

countace+=11;

}

if(hand[i]%12>8){

count+=10;

countace+=10;

}

if(hand[i]%12<9&&hand[i]%12>0){

count+=(hand[i]%12);

countace+=(hand[i]%12);

count+=1;

countace+=1;

}

}

if(countace<22){

return countace;

}

if(countace>=22){

return count;

}

}

void Deal(vector<int> &hand,vector<int> &deck){

int temp=deck[0]%52;

hand.push\_back(temp);

deck.erase(deck.begin());

temp=deck[0]%52;

hand.push\_back(temp);

deck.erase(deck.begin());

}

void CardDisplay(int i,int j){

Card cards;

Suit suits;

cards=static\_cast<Card>(i);

suits=static\_cast<Suit>(j);

switch(cards){

case Ace:cout<<setw(21)<<"Ace ";break;

case Two:cout<<setw(21)<<"Two ";break;

case Three:cout<<setw(21)<<"Three ";break;

case Four:cout<<setw(21)<<"Four ";break;

case Five:cout<<setw(21)<<"Five ";break;

case Six:cout<<setw(21)<<"Six ";break;

case Seven:cout<<setw(21)<<"Seven ";break;

case Eight:cout<<setw(21)<<"Eight ";break;

case Nine:cout<<setw(21)<<"Nine ";break;

case Jack:cout<<setw(21)<<"Jack ";break;

case Queen:cout<<setw(21)<<"Queen ";break;

case King:cout<<setw(21)<<"King ";break;

}

switch(suits){

case OfDiamonds:cout<<"Of Diamonds";break;

case OfSpades:cout<<"Of Spades";break;

case OfHearts:cout<<"Of Hearts";break;

case OfClubs :cout<<"Of Clubs";break;

}

cout<<endl;

}

void addDecks(vector<int> &stack){

vector<int> newdecks;

int cut=0;

int count1=1;

int temp=0;

for(int i=0;i<312;i++){

newdecks.push\_back(count1);

count1++;

}

Shuffle(newdecks);

cout<<"\n6 new decks, shuffled together.\n";

cout<<"Cut the deck (1-312):";

cin>>cut;

if(cut<0||cut>312){

cut=156;

cout<<"\nInvalid input. Cutting deck in half.\n";

}

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

temp=newdecks[0];

newdecks.push\_back(temp);

newdecks.erase(newdecks.begin());

}

cout<<"\nBurning the top card.";

newdecks.erase(newdecks.begin());

cout<<"\nAdding the new decks to the stack.\n";

int vec=newdecks.size();

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

stack.push\_back(newdecks[i]);

}

temp=stack.size();

temp+=1;

cout<<"Deck total: "<<temp<<endl;

}

void Shuffle(vector<int> &stack){

int nShuffle = stack.size();

for(int i=1;i<=nShuffle;i++){

for(int j=0;j<nShuffle;j++){

int temp=rand()%nShuffle;

int card=stack[j];

stack[j]=stack[temp];

stack[temp]=card;

}

}

}