Joseph Reimbold
CSC-17C 48596
11/11/2014

Project 1

Black Jack

Introduction

Game: Black Jack
The objective of this card game is to have a hand with a value as close to the number 21 as possible with an instant win if you do get 21 and the dealer (computer) does not. If you tie the dealer, you get your bet back. If your hand goes over 21 you bust and you automatically lose.

Summary

Project Size: Approx 750 lines
# of Variables Used: 20+
# of Functions Used: 5
# of Classes Used: 4
Black Jack is a favorite among casino visitors. It's actually a bit more complex than I initially anticipated due to a few factors like when deciding an ace was worth 1 or 11. Many of the games steps are in functions and classes used to store information in decks and cards as well as player information.

Description

I decided to go with Black Jack because I thought I would be able to demonstrate a number of the things we have learned this semester so far. Nearly every step is in a function with the exception to determining a hit or stand for both the player and dealer(computer). A player class stores your name, funds and the total value of your hand. A Card class was used to store all card information (name, suit, value). The deck was created using a vector of Cards which was run through with the random_shuffle algorithm to get it fully randomized. It was then shoved into a dynamic stack (linked list) that I made so that the top card of the deck would always be the next one drawn. Each player's hand was stored in a dynamic queue (linked list) as the first card you received would then be the first discarded.

Things To Improve

- A cleaner method of displaying, couldn't get it working with stacks/queue how I wanted
- Create a GUI with actual cards
- Add in extra elements of the game, like doubling down, splitting, etc

Psuedo Code

```
//Libraries
//Function Prototypes

//Begin Execution Here
        //Declare Variables
        //Create map for score
        //Introduce the game
        //Get name from user
        //Begin play while user selects yes
        //Create the deck using vector
        //Shuffle the deck and put into stack to deal from
        //Ask player how much they want to bet
        //Deal to both players two cards
        //Player makes decisions on hitting or standing until done
```

```
            //Output results of hitting/standing
        //Dealer's turn to hit/stand as needed following rules of game
            //Determine if dealer has black jack from initial deal
            //If dealer has 17 or higher, stand
            //If dealer has 16 or lower, hit
        //Determine both player's totals and see who wins
        //Reset all important values in order to prepare for another game
        //Ask if user would like to play the game again, repeat as desired
        //Output final score and players ending funds from betting
//End of Main

//build deck function
        //Create an un-shuffled deck using a vector using card class

//shuffle function
        //Shuffle vector deck
        //Convert shuffled vector deck into DeckStack class
        //Use iterator to move through vector and push

//firstDeal function
        //First two cards dealt to player and dealer
        //If first draw is ace set val to 11 instead of 1
        //If first draw is not ace and second is, set ace to 11 value
        //If first two draws are aces
        //Add any other value
        //Increment cards dealt
        //Alternate between each player two times each

//deal function
        //Deal a single card to whichever player is called to function

//result function
        //If you bust, you lose
        //If the dealer busts, you win
        //If both have blackjack, push
        //If both have same value, push
            //If p1 has blackjack, p1 wins
            //If cpu has blackjack, cpu wins
            //If just same value, push
        //If p1 is closer to 21, p1 wins
        //If cpu is closer to 21, cpu wins

//reset function
        //Empty player and CPU hands
        //Clear current deck stack to put fresh shuffled deck in
        //Clear player info and cards dealt
```

Major Variables

Type	Variable Name	Description	Location
Player	p1, cpu	Class which holds name, hand value and funds	Main(), deal(), firstDeal(), result(), reset()
Card	vector<Card> unshflDck	Card holds suit, name, value and number created, vector holds them in an unshuffled deck	Main(), bldDck(), shuffle(),
	cardCatch	Used to store popped cards from stack and queue	FirstDeal(), deal(), reset()
DeckStack	myDeck	Stacked deck	Main(), shuffle(), firstDeal(), deal(), reset()
HandQueue	p1Hand, cpuHand	Queue stack, holds player and dealer hand	Main(), firstDeal(), deal(), reset()
map<string,int>	score_map	Held wins/losses/ties	Main(), result()
int	nCards	Number of cards in the deck	Main(), shuffle(), bldDck(), reset()
	nDealt	Number of cards dealt this game	Main(), deal(), firstDeal(), reset()
	choice	Used for hit or stand	Main()
	betAmnt	Stores amount player bets	Main(), result()
char	again	Holds play again value	Main()
bool	p1bj	Used if player gets a black jack	Main(), result(), reset()
	cpubj	Used if player gets a black jack	Main(), result(), reset()

Reference

1. *Starting out with C++* - Tony Gaddis
2. http://www.cprogramming.com/tutorial/stl/stlmap.html

Program

```
/*
 * File:   main.cpp
 * Author: Joe
 * Created on November 8, 2014, 10:46 AM
 *
 * Project 1: Black Jack
 *
 * To Do List:
 * Project and writeup.
 * Utilize STL library - DONE
 * Maps - score keeping - DONE
 * Dynamic Stack - (linked list) shuffled deck to deal cards from - DONE
 * Queues - (dynamic queue) player hand - DONE
 * Iterators - use to traverse where applicable - DONE
 * Algorithms - use random_shuffle - DONE
 * Post to your repository, here and send an email.
 */

//Libraries
#include <cstdlib>
#include <iostream>
#include <vector>
#include <iomanip>
#include <algorithm>
#include <ctime>
#include <string>
#include <map>
using namespace std;

//Our Libraries
#include "Card.h"
#include "DeckStack.h"
#include "HandQueue.h"
#include "Player.h"

//Function Prototypes
vector<Card> bldDck(int);
void shuffle(vector<Card> &,int,DeckStack &);
void frstDeal(DeckStack &,HandQueue &,HandQueue &,int &, Player &, Player &);
void deal(DeckStack &,HandQueue &,int &,Player &);
void result(Player &,Player &,bool &,bool &,map<string,int> &,int);
void reset(HandQueue &,HandQueue &,DeckStack &,Player &,Player &,int,int &,bool &,bool &);

//Execution Begins Here
int main(int argc, char** argv) {
    //Declare variables
    int nCards=52;              //number of cards in deck
```

```
int nDealt=0;               //number of cards dealt
vector<Card> unshflDck;      //holds an un-shuffled deck
DeckStack myDeck;           //holds a shuffled deck in stack form
std::srand(std::time(0));    //seed random number generator
Card crdCatch;              //holds popped cards from stack/queue
HandQueue p1Hand;           //player's hand in queue form
HandQueue cpuHand;          //cpu's hand in queue form
Player p1;                  //holds player's info
string p1name;              //take input for player's name
Player cpu;                 //holds cpu's info
int choice;                 //holds decision to hit, stand, etc
char again;                 //holds decision to play again
bool p1bj=false;            //player got a blackjack
bool cpubj=false;           //cpu got a blackjack
int betAmnt;

//Create map for score
map <string,int> score_map; //holds wins, losses and pushes
score_map["P1 Wins: "]=0;
score_map["P1 Losses: "]=0;
score_map["CPU Wins: "]=0;
score_map["CPU Losses: "]=0;
score_map["Pushes: "]=0;

//Introduce the game
cout<<"Welcome to Black Jack!"<<endl<<endl;
//Get name from user
cout<<"Please enter your name:";
cin>>p1name;
p1.setName(p1name);
cout<<endl;

//Begin Play
do{
   //Create an un-shuffled deck using a vector
   unshflDck=bldDck(nCards);
   //shuffle the deck and put into a DeckStack to begin play with
   shuffle(unshflDck,nCards,myDeck);
   //deal and continue play
   cout<<"Current funds: "<<p1.getFunds()<<endl;
   do{
      cout<<"How much would you like to bet (min 5)? ";
      cin>>betAmnt;
   }while(betAmnt>p1.getFunds()||betAmnt<5);
   frstDeal(myDeck,p1Hand,cpuHand,nDealt,p1,cpu);
   if(p1.getValue()==21){
      cout<<"Blackjack!"<<endl;
      p1bj=true;
   }
```

```
    //Player decides hit or stand if no blackjack drawn
    if(p1.getValue()<21){
       do{
          cout<<p1.getName()<<" hand total: "<<p1.getValue()<<endl;
          cout<<"What would you like to do?"<<endl;
          cout<<"1. Hit"<<endl<<"2. Stand"<<endl;
          cin>>choice;
          cout<<endl;
          if(choice==1){
             deal(myDeck,p1Hand,nDealt,p1);
          }
       }while(choice==1&&p1.getValue()<=21);
       //output result of player's hits/stand
       if(choice==2)cout<<"You stayed at "<<p1.getValue()<<"."<<endl;
       if(p1.getValue()==21)cout<<"You got 21!"<<endl;
    }
    cout<<endl;
    //CPUs hit/stand phase
    if(p1.getValue()<=21){
       cout<<"The CPU is now going to hit/stand."<<endl;
       //if cpu has blackjack
       if(cpu.getValue()==21){
          cout<<"The CPU has a blackjack!"<<endl;
          cpubj=true;
       }
       //if cpu has 17or greater, stand
       else if(cpu.getValue()>=17){
          cout<<"The CPU stands at "<<cpu.getValue()<<"."<<endl;
       }
       //draw if under 17
       else if(cpu.getValue()<=16){
          deal(myDeck,cpuHand,nDealt,cpu);
       }
    }
    //determine winner/loss/push
    result(p1,cpu,p1bj,cpubj,score_map,betAmnt);
    cout<<"Total cards dealt: "<<nDealt<<endl;

    //reset game for additional plays
    reset(p1Hand,cpuHand,myDeck,p1,cpu,nCards,nDealt,p1bj,cpubj);
    shuffle(unshflDck,nCards,myDeck);

    //Ask if user would like to play the game again
    cout<<"Would you like to play again? (Y/N)"<<endl;
    cin>>again;
    cout<<endl<<endl;
 }while(again=='y'||again=='Y');

 //output overall score and end funds
```

```
    cout<<"Final Score:"<<endl;
    map<string,int>::iterator iter;
    for (iter=score_map.begin(); iter!=score_map.end(); ++iter){
        cout<<iter->first<<": "<<iter->second<<endl;
    }
    cout<<"Funds: "<<p1.getFunds()<<endl;
    if(p1.getFunds()<0)cout<<"You're now in debt! Woops!"<<endl;
    else if(p1.getFunds()>100)cout<<"You made a profit! Congrats!"<<endl;
    else if(p1.getFunds()==100)cout<<"You broke even!"<<endl;
    else
        cout<<"You lost money..."<<endl;

    //Exit stage right
    return 0;
}

vector<Card> bldDck(int n){
    //Create an un-shuffled deck using a vector
    vector<Card> deck;
    for(int i=0;i<n;i++){
        Card card(i);
        deck.push_back(card);
    }
    return deck;
}

void shuffle(vector<Card> &deck,int n,DeckStack &dckStck){
    //shuffle vector card deck
    random_shuffle(deck.begin(),deck.end());
    //take vector deck and convert to stack to deal from
    vector<Card>::iterator iter;       //iterator to move through vector
    for(iter=deck.begin();iter!=deck.end();iter++){
        dckStck.push(*iter);
    }
}

void frstDeal(DeckStack &myDeck,HandQueue &p1Hand,HandQueue &cpuHand,int &nDealt, Player
&p1, Player &cpu){
    //initial deal to players
    Card crdCatch;          //holds cards popped
    for(int i=0;i<4;i++){
        if(i%2==0){
            myDeck.pop(crdCatch);
            p1Hand.enqueue(crdCatch);
            //if first draw is ace set val to 11 instead of 1
            if(crdCatch.getName()=='A'&&nDealt==0)
                p1.setValue(11);
            //if first draw is not ace and second is, set ace to 11 value
            else if(crdCatch.getName()=='A'&&p1.getValue()<11)
```

```
            p1.setValue(11);
         //if first two draws are aces
         else if(crdCatch.getName()=='A'&&p1.getValue()==11)
            p1.setValue(1);
         else
            p1.setValue(crdCatch.getValue());
         nDealt++;
         cout<<p1.getName()<<" receives card: "<<crdCatch.getName()<<crdCatch.getSuit()
            <<endl<<"P1 hand value: "<<p1.getValue()<<endl;
         cout<<p1.getName()<<" has "<<p1Hand.getNumHand()<<" cards in hand."<<endl<<endl;
      }
      if(i%2==1){
         myDeck.pop(crdCatch);
         cpuHand.enqueue(crdCatch);
         //if first draw is ace set val to 11 instead of 1
         if(crdCatch.getName()=='A'&&nDealt==0)
            cpu.setValue(11);
         //if first draw is not ace and second is, set ace to 11 value
         else if(crdCatch.getName()=='A'&&cpu.getValue()<11)
            cpu.setValue(11);
         //if first two draws are aces
         else if(crdCatch.getName()=='A'&&cpu.getValue()==11)
            cpu.setValue(1);
         else
            cpu.setValue(crdCatch.getValue());
         if(nDealt==1)
            cout<<"CPU receives hidden card."<<endl<<endl;
         else{
            cout<<"CPU receives card: "<<crdCatch.getName()<<crdCatch.getSuit()<<endl;
            cout<<"CPU has "<<cpuHand.getNumHand()<<" cards in hand."<<endl<<endl;
         }
         nDealt++;
      }
   }
}

void deal(DeckStack &myDeck,HandQueue &hand,int &nDealt,Player &plyr){
   Card crdCatch;       //holds popped cards
   myDeck.pop(crdCatch);
   hand.enqueue(crdCatch);
   //determine value of aces drawn
   if(crdCatch.getName()=='A'){
      if(plyr.getValue()<=10)
         plyr.setValue(11);
      else
         plyr.setValue(1);
   }
   else
      plyr.setValue(crdCatch.getValue());
```

```
   cout<<plyr.getName()<<" hits and receives card:
"<<crdCatch.getName()<<crdCatch.getSuit()<<endl;
   cout<<plyr.getName()<<" has "<<hand.getNumHand()<<" cards in hand."<<endl<<endl;
   nDealt++;
}

void result(Player &p1, Player &cpu,bool &p1bj,bool &cpubj,map<string,int> &score,int bet){
   //if you bust, you lose
   cout<<"Your Hand: "<<p1.getValue()<<"     CPU's Hand: "<<cpu.getValue()<<endl;
   if(p1.getValue()>21){
      cout<<"You busted with "<<p1.getValue()<<"!"<<endl;
      p1.setLosses(1);
      cpu.setWins(1);
      score["P1 Losses: "]++;
      score["CPU Wins: "]++;
      p1.setFunds(-bet);
   }
   else if(cpu.getValue()>21){
      cout<<"CPU busted with "<<cpu.getValue()<<"!"<<endl;
      p1.setWins(1);
      cpu.setLosses(1);
      score["P1 Wins: "]++;
      score["CPU Losses: "]++;
      p1.setFunds(bet);
   }
   //if both have blackjack, push
   else if(p1bj&&cpubj){
      cout<<"This game is a push!"<<endl;
      p1.setPushes(1);
      cpu.setPushes(1);
      score["Pushes: "]++;
   }
   //if both have same value, push
   else if(p1.getValue()==cpu.getValue()){
      //if p1 has blackjack, p1 wins
      if(p1bj){
         cout<<p1.getName()<<" wins!"<<endl;
         p1.setWins(1);
         cpu.setLosses(1);
         score["P1 Wins: "]++;
         score["CPU Losses: "]++;
         p1.setFunds(bet*3/2);
      }
      //if cpu has blackjack, cpu wins
      else if(cpubj){
         cout<<"CPU wins with Blackjack!"<<endl;
         p1.setLosses(1);
         cpu.setWins(1);
         score["P1 Losses: "]++;
```

```
            score["CPU Wins: "]++;
            p1.setFunds(-bet);
        }
        //if just same value, push
        else{
            cout<<"This game is a push!"<<endl;
            p1.setPushes(1);
            cpu.setPushes(1);
            score["Pushes: "]++;
        }
    }
    //if p1 is closer to 21, p1 wins
    else if(p1.getValue()>cpu.getValue()&&p1.getValue()<=21){
        cout<<p1.getName()<<" wins!"<<endl;
        p1.setWins(1);
        cpu.setLosses(1);
        score["P1 Wins: "]++;
        score["CPU Losses: "]++;
        p1.setFunds(bet);
    }
    //if cpu is closer to 21, cpu wins
    else{
        cout<<"CPU wins with "<<cpu.getValue()<<"!"<<endl;
        p1.setLosses(1);
        cpu.setWins(1);
        score["P1 Losses: "]++;
        score["CPU Wins: "]++;
        p1.setFunds(-bet);
    }
}

void reset(HandQueue &p1Hand,HandQueue &cpuHand,DeckStack &dckStck,Player &p1,Player
&cpu,int n,int &nDealt,bool &p1bj,bool &cpubj){
    Card card;       //used to pop cards out of deck
    //empty player and CPU hands
    p1Hand.clear();
    cpuHand.clear();
    //clear current deck stack to put fresh shuffled deck in
    while(!dckStck.isEmpty()){
        dckStck.pop(card);
    }
    //for(int i=0;i<(n-nDealt);i++){
    //    dckStck.pop(card);
    //}
    //clear player info and cards dealt
    p1Hand.resetNumHand();
    cpuHand.resetNumHand();
    p1.resetValue();
    cpu.resetValue();
```

```
    p1bj=false;
    cpubj=false;
    nDealt=0;
}
```

Card.h

```
#ifndef CARD_H
#define CARD_H

class Card{
private:
    char number;     //card's creation number (used to determine following)
    char name;       //holds card's "name(1,5,10,J,A,etc)
    char suit;       //holds card's suit
    int value;      //holds card's value
public:
    //Constructors
    Card();
    Card(int);
    //Mutators
    char nameCard();
    char suitCard();
    int valueCard();
    //Accessor
    char getNumber(){return number;}
    char getName(){return name;}
    char getSuit(){return suit;}
    char getValue(){return value;}
};

#endif /* CARD_H */
```

Card.cpp

```
//Our Libraries
#include "Card.h"

//Default Constructor
Card::Card(){
    number=' ';
    name=' ';
    suit=' ';
    value=0;
}

//Constructor
```

```
Card::Card(int num){
   if(num<0)num=0;
   if(num>51)num%=52;
   this->number=num;
   name=nameCard();
   suit=suitCard();
   value=valueCard();
}

//Determine card's "name"
char Card::nameCard(){
   //Declare card name array
   char aName[]={'A','2','3','4','5','6','7','8','9','T','J','Q','K'};
   return aName[number%13];
}

//Determine suit of card
char Card::suitCard(){
   if(number<13)return 'S';
   if(number<26)return 'H';
   if(number<39)return 'C';
   else return 'D';
}

//Determine card value
int Card::valueCard(){
   int n=(number)%13+1;

   if(n>10)return 10;
   return n;
}
```

DeckStack.h

```
//Libraries
#include <iostream>
#include "Card.h"
using namespace std;

#ifndef DECKSTACK_H
#define          DECKSTACK_H

class DeckStack{
private:
   //Structure for the deck nodes
   struct DeckNode{
      Card data;      //value in node
      DeckNode *next;  //Pointer to next node
```

```
    };
    DeckNode *top;       //Points to top of deck stack
    DeckNode *worker;    //used to traverse
public:
    //Constructor
    DeckStack()
    {top=NULL;}
    //Destructor
    ~DeckStack();
    //Mutators
    void push(Card);
    void pop(Card &);
    //Accessors
    bool isEmpty()const;
};
#endif /* DECKSTACK_H */
```

DeckStack.cpp

```
#include <iostream>
#include "DeckStack.h"
using namespace std;

//Destructor
DeckStack::~DeckStack(){
    DeckNode *nextNode;
    worker=top;
    //traverse the list and delete each node
    while(worker!=NULL){
        nextNode=worker->next;
        delete worker;
        worker=nextNode;
    }
}

//Mutators
//Add card to top of deck
void DeckStack::push(Card crd){
    //create new node to hold a new card
    DeckNode *newNode=new DeckNode;
    newNode->data=crd;
    //if no nodes make newNode the first
    if(isEmpty()){
        top=newNode;
        newNode->next=NULL;
    }
    //if not, insert before top
```

```
    else{
        newNode->next=top;
        top=newNode;
    }
}

//Remove card from top of deck and pass by reference out
void DeckStack::pop(Card &crd){
    DeckNode *temp;
    //Make sure not empty
    if(isEmpty())
        cout<<"The deck is empty!"<<endl;
    //if not, pop value off top and return card to deal
    else{
        crd=top->data;
        temp=top->next;
        delete top;
        top=temp;
    }
}

//Check to see if deck is empty
bool DeckStack::isEmpty() const{
    bool empty;
    if(!top)
        empty=true;
    else
        empty=false;
    return empty;
}
```

HandQueue.h

```
#include <iostream>
#include "Card.h"
using namespace std;

#ifndef HANDQUEUE_H
#define          HANDQUEUE_H

class HandQueue{
private:
    //structure for hand queue nodes
    struct HandNode{
        Card data;       //holds card data
        HandNode *next;   //pointer to next node
    };
    HandNode *front;      //front of the queue
```

```
    HandNode *rear;       //rear of the queue
    int numHand;          //number of cards in hand
public:
    //Constructor
    HandQueue();
    //Destructor
    ~HandQueue();
    //Mutators
    void enqueue(Card);   //place card in hand from deck
    void dequeue(Card &); //remove card from hand, used to put back in deck
    bool isEmpty()const;
    void clear();
    void resetNumHand();
    //Accessors
    int getNumHand();
};

#endif /* HANDQUEUE_H */
```

HandQueue.cpp

```
#include <iostream>
#include "HandQueue.h"
using namespace std;

//Constructor
HandQueue::HandQueue(){
    front=NULL;
    rear=NULL;
    numHand=0;
}

//Destructor
HandQueue::~HandQueue(){
    clear();
}

void HandQueue::clear(){
    Card val;    //temp for dequeue
    while(!isEmpty())
        dequeue(val);
}

//Mutators
void HandQueue::enqueue(Card crd){
    //create new hand node and store card info
    HandNode *newNode=new HandNode;
    newNode->data=crd;
```

```
    newNode->next=NULL;
    //adjust front and rear
    if(isEmpty()){
        front=newNode;
        rear=newNode;
    }
    else{
        rear->next=newNode;
        rear=newNode;
    }
    //increment numHand
    numHand++;
}

void HandQueue::dequeue(Card& crd){
    HandNode *temp;
    if(isEmpty())
        cout<<"The hand is empty!"<<endl;
    else{
        //save front card into crd to pass back by reference
        crd=front->data;
        //remove front node and delete it
        temp=front;
        front=front->next;
        delete temp;
        //decrement numHand
        numHand--;
    }
}

bool HandQueue::isEmpty() const{
    bool status;
    if(numHand>0)
        status=false;
    else
        status=true;
    return status;
}

void HandQueue::resetNumHand(){
    numHand=0;
}

//Accessors
int HandQueue::getNumHand(){
    return numHand;
}
```

Player.h

```
#include <iostream>
#include <string>
#include <queue>
using namespace std;

#ifndef PLAYER_H
#define          PLAYER_H

class Player{
private:
   string name;       //holds player name
   int funds;         //holds player's funds
   int value;         //holds player's hand value
   int wins;          //holds player's wins
   int losses;        //holds player's losses
   int pushes;        //holds player's pushes
public:
   //Constructor
   Player();
   //Mutators
   void setName(string);
   void setFunds(int);
   void resetValue();
   void setValue(int);
   void setWins(int);
   void setLosses(int);
   void setPushes(int);
   //Accessors
   string getName();
   int getFunds();
   int getValue();
   int getWins();
   int getLosses();
   int getPushes();
};

#endif /* PLAYER_H */
```

Player.cpp

```
#include <iostream>
#include "Player.h"
using namespace std;

//Constructor
```

```
Player::Player(){
   name="CPU";
   funds=100;
   value=0;
   wins=0;
   losses=0;
   pushes=0;
}

//Mutators
void Player::setName(string n) {
   name=n;
}

void Player::setFunds(int f){
   funds+=f;
}

void Player::resetValue(){
   value=0;
}

void Player::setValue(int v){
   value+=v;
}

void Player::setWins(int win){
   wins+=win;
}

void Player::setLosses(int loss){
   losses+=loss;
}

void Player::setPushes(int push){
   pushes+=push;
}

//Accessors
string Player::getName(){
   return name;
}
int Player::getFunds(){
   return funds;
}

int Player::getValue(){
   return value;
}
```

```
int Player::getWins(){
    return wins;
}

int Player::getLosses(){
    return losses;
}

int Player::getPushes(){
    return pushes;
}
```