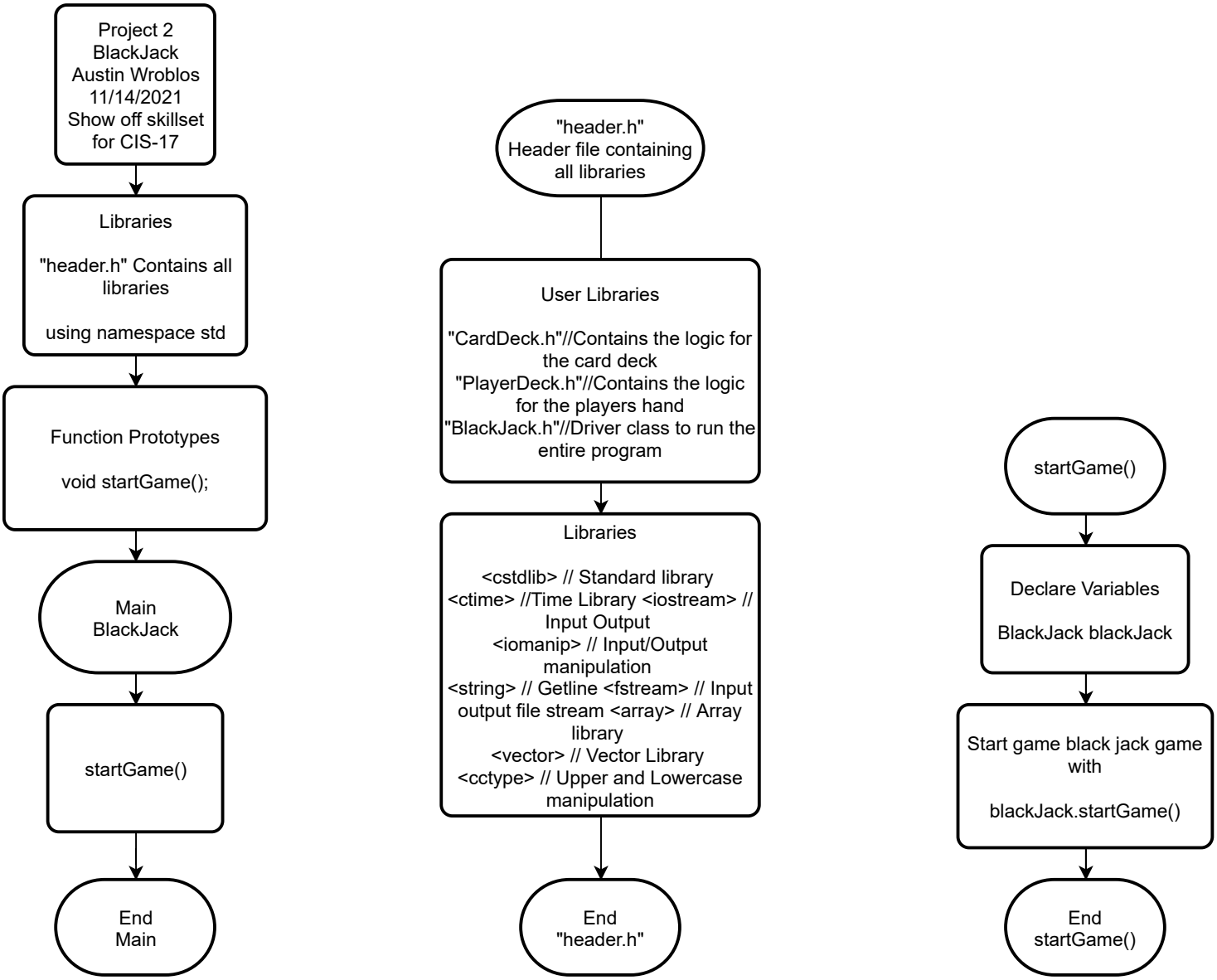
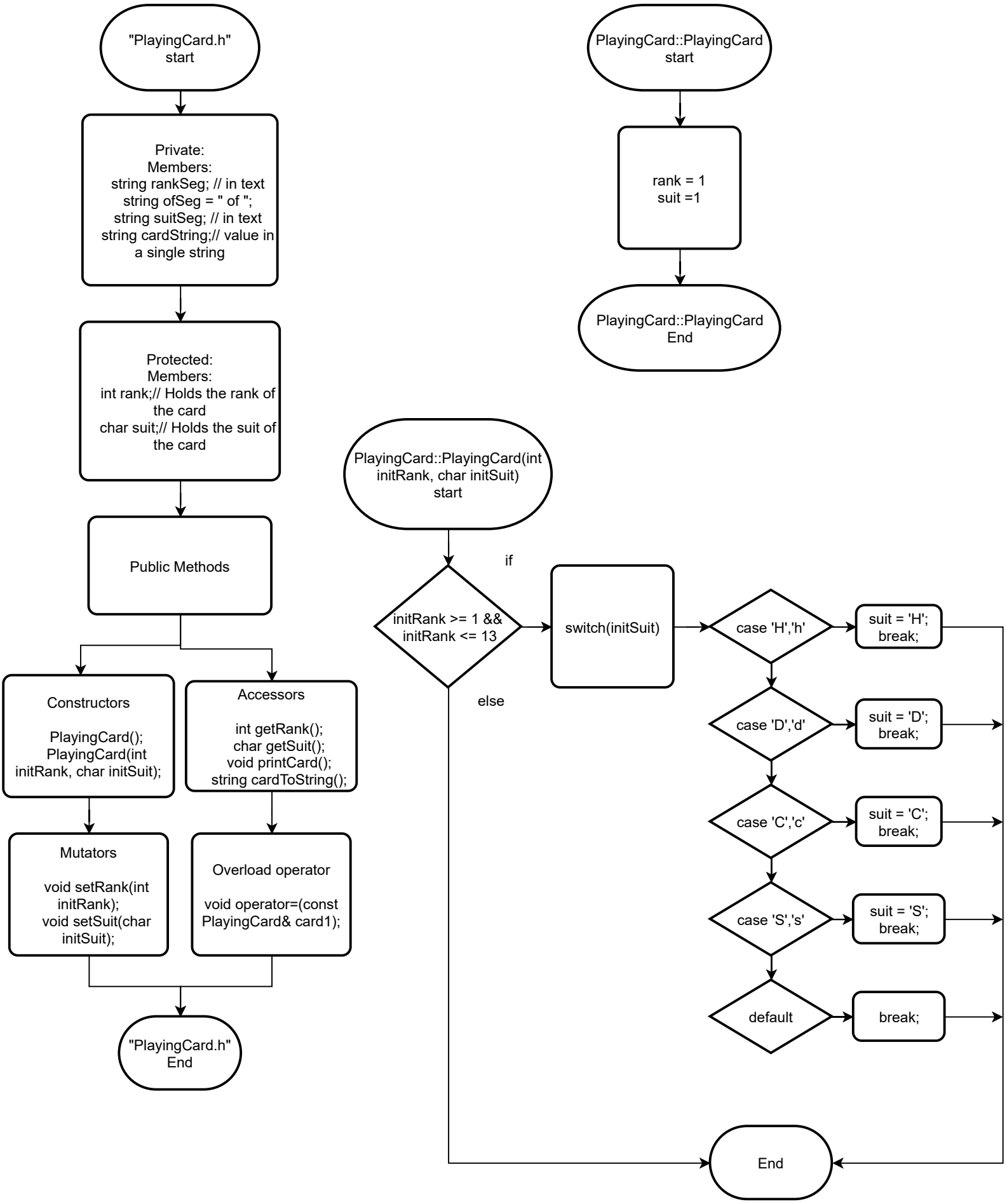
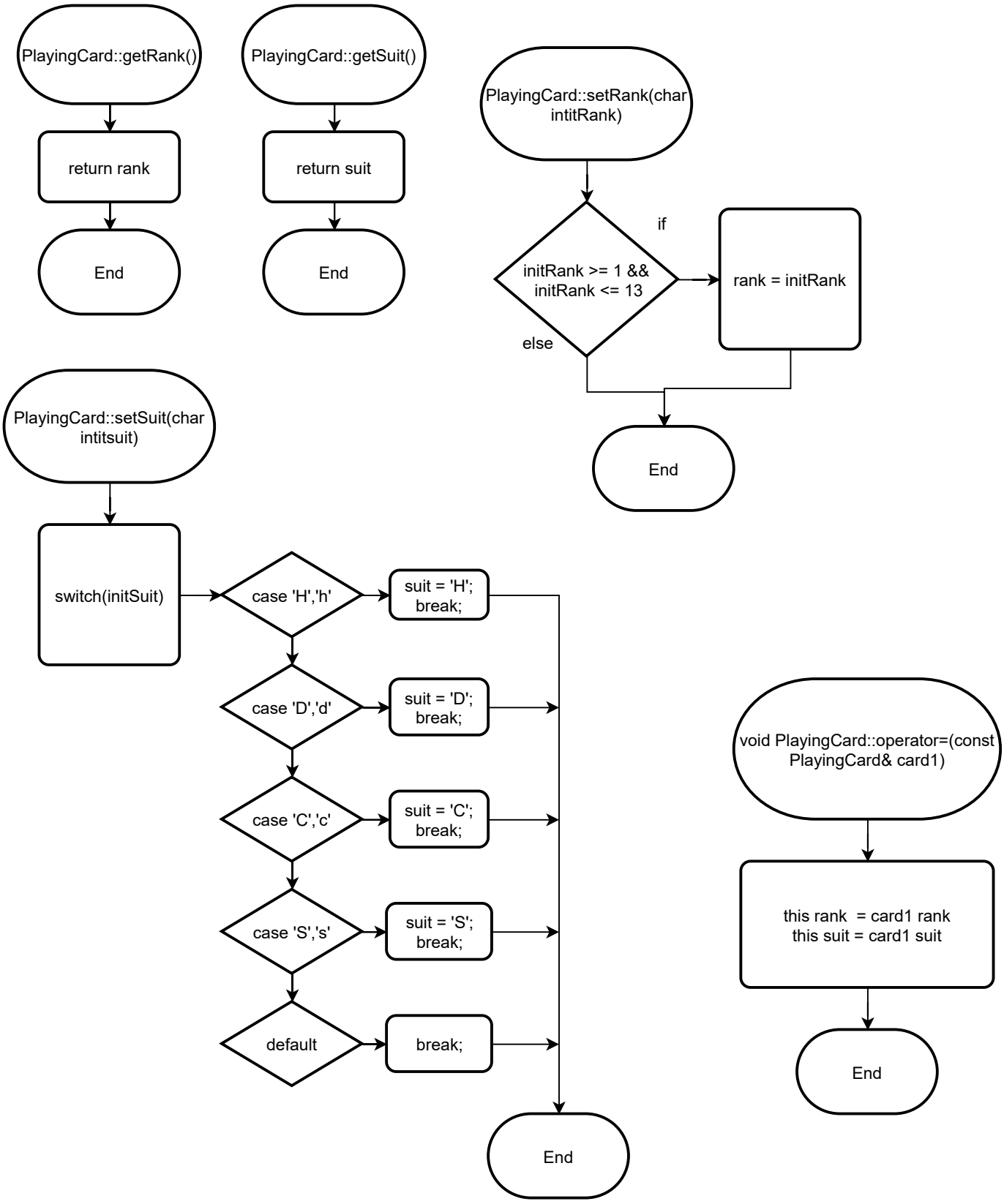
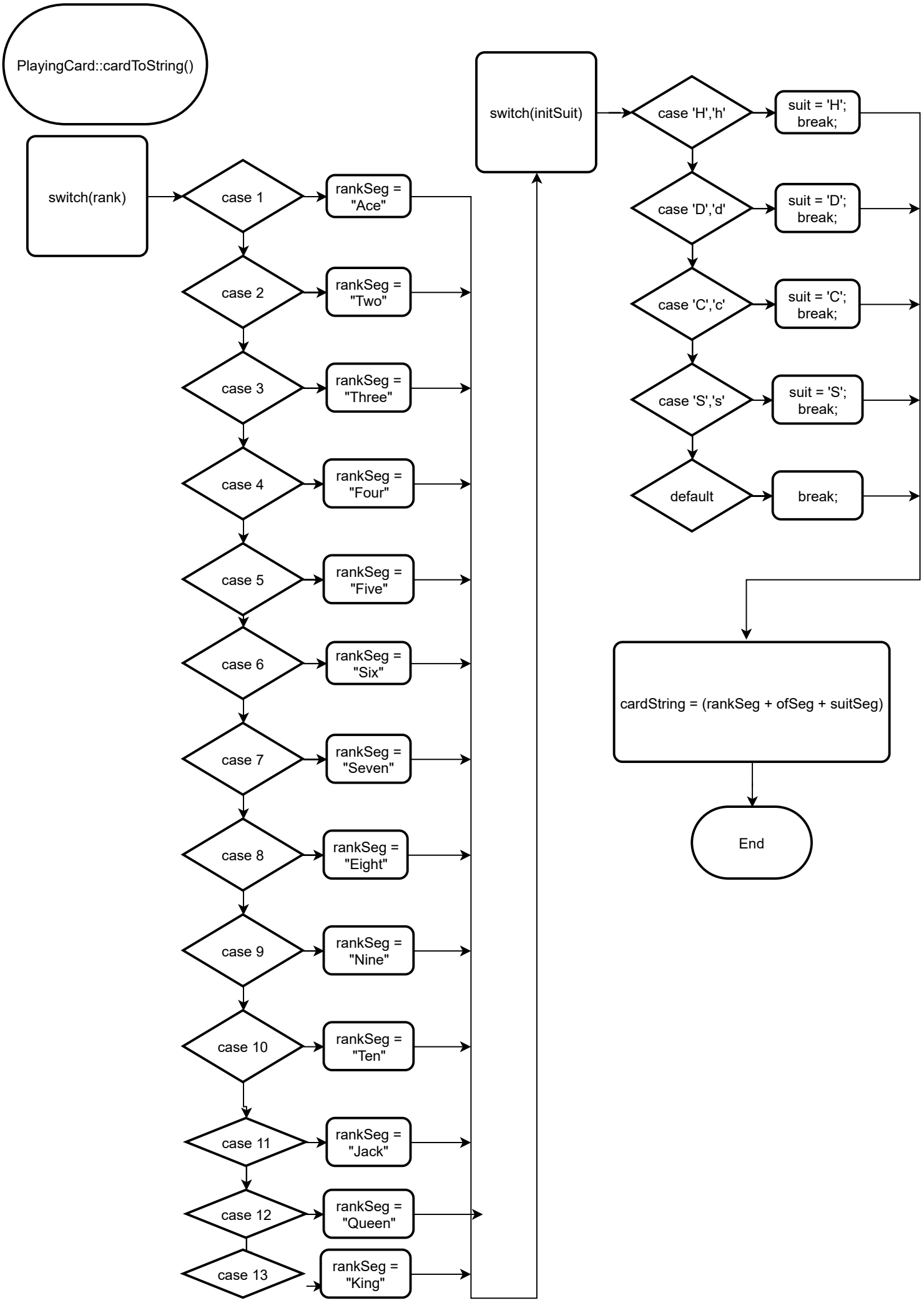
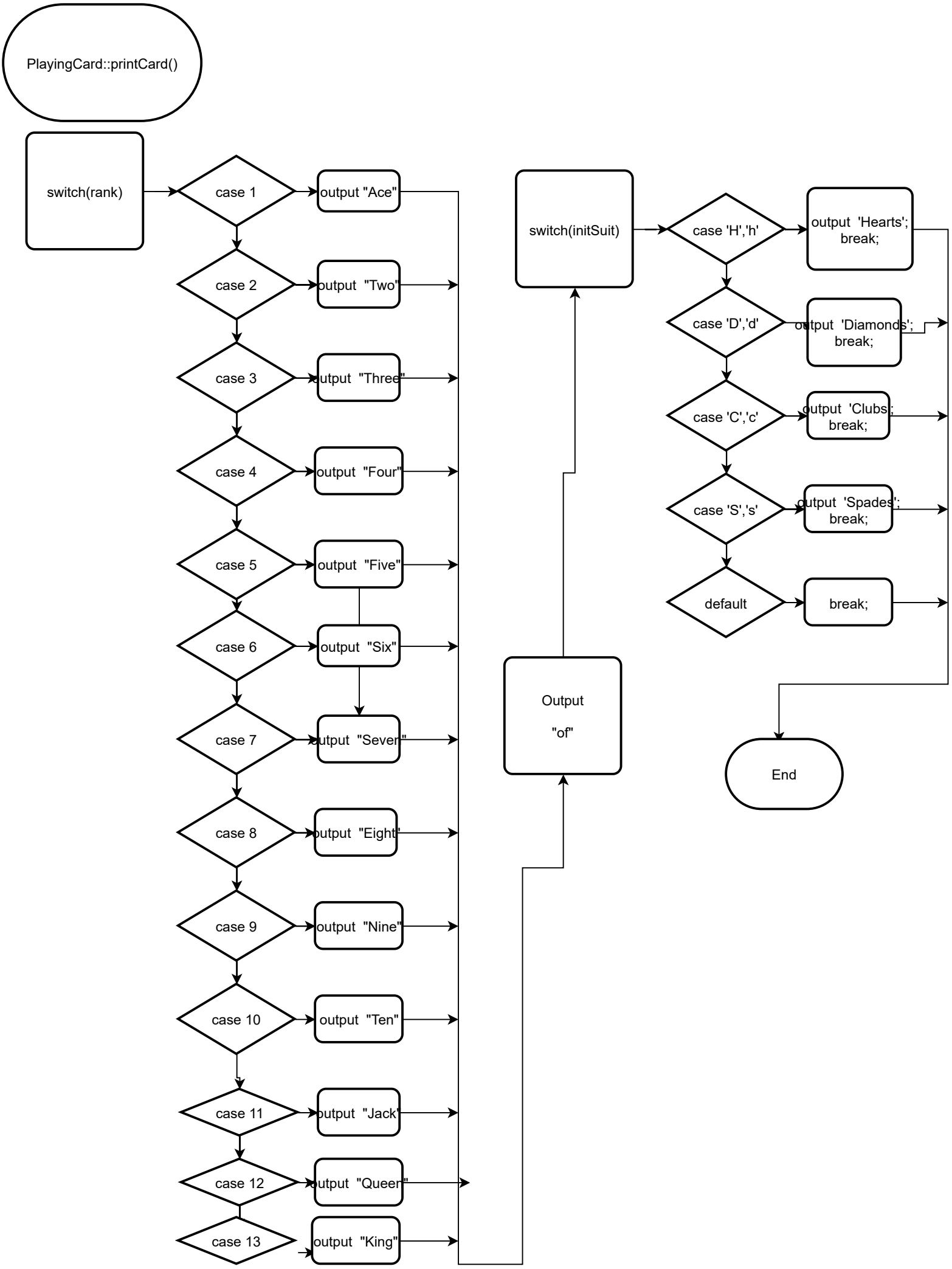
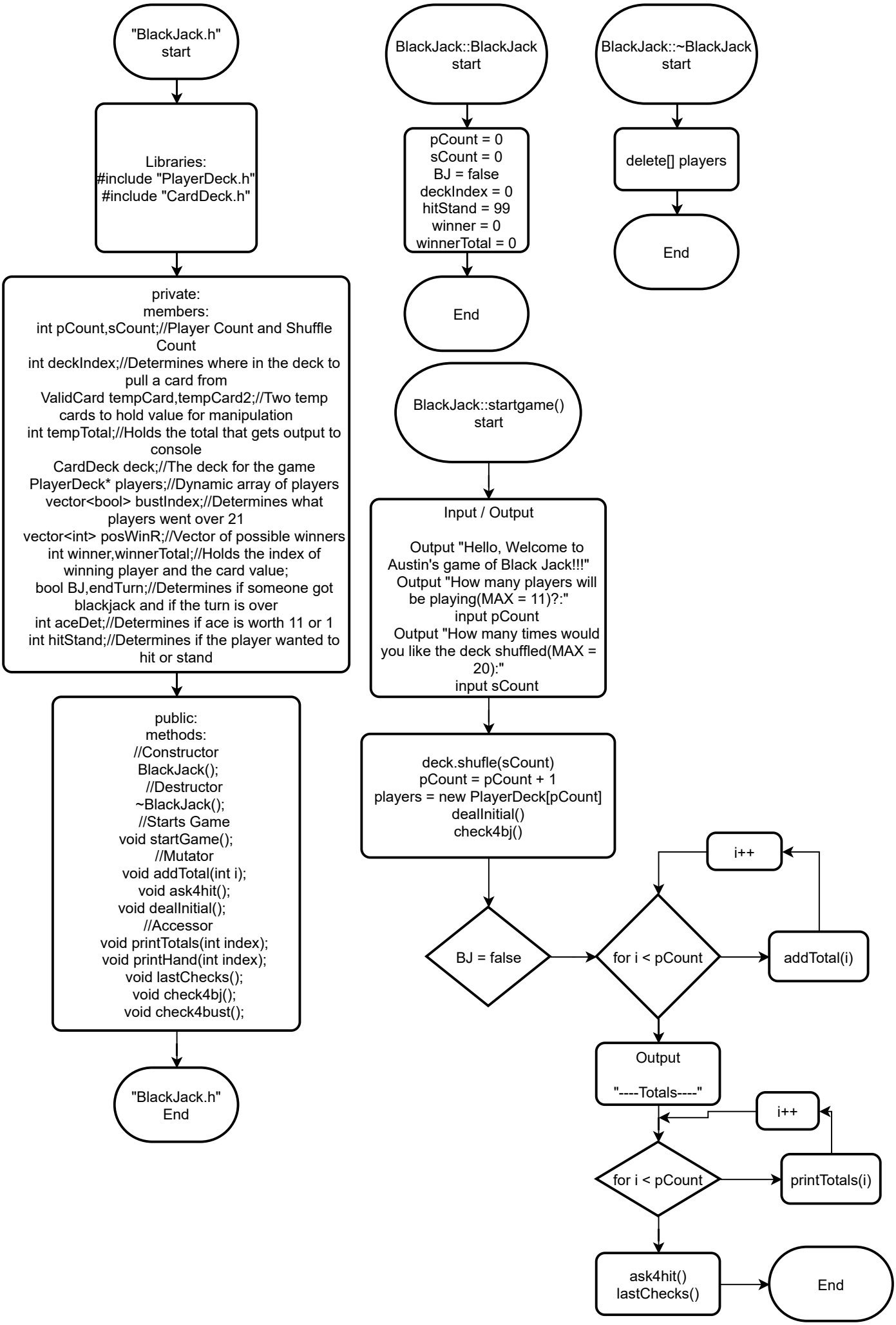

Project 2
BlackJack
Austin Wroblos
11/14/2021
Show off skillset
for CIS-17
Libraries
"header.h" Contains all libraries
using namespace std
Function Prototypes
void startGame();
Main
BlackJack
startGame()
End
Main
"header.h"
Header file containing all libraries
User Libraries
"CardDeck.h"//Contains the logic for the card deck
"PlayerDeck.h"//Contains the logic for the players hand
"BlackJack.h"//Driver class to run the entire program
Libraries
<cstdlib> // Standard library
<ctime> //Time Library <iostream> // Input Output
<iomanip> // Input/Output manipulation
<string> // Getline <fstream> // Input output file stream <array> // Array library
<vector> // Vector Library
<cctype> // Upper and Lowercase manipulation
End
"header.h"
startGame()
Declare Variables
BlackJack blackJack
Start game black jack game with
blackJack.startGame()
End
startGame()

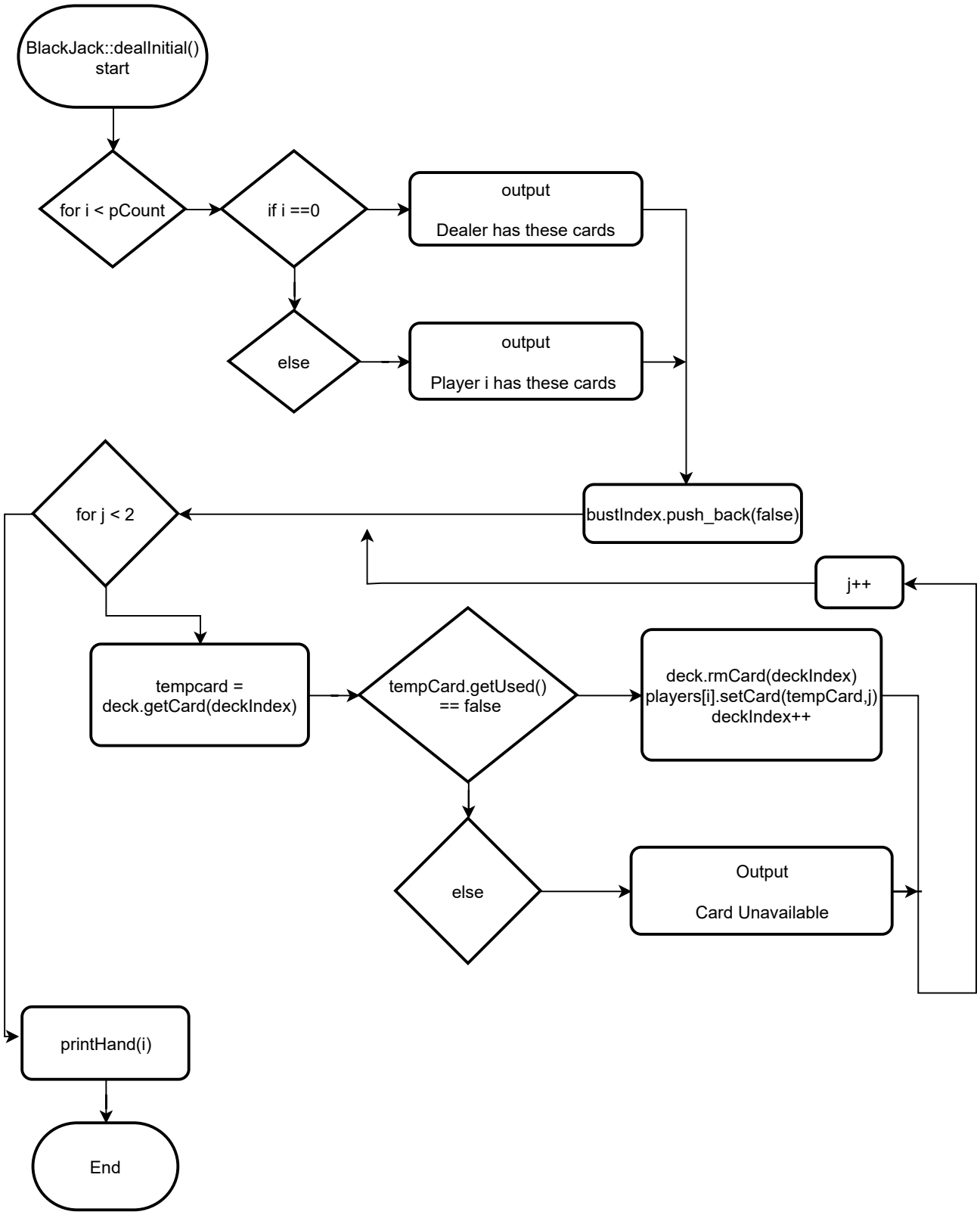

"PlayingCard.h"
start
Private:
Members:
string rankSeg; // in text
string ofSeg = " of ";
string suitSeg; // in text
string cardString;// value in a single string
Protected:
Members:
int rank;// Holds the rank of the card
char suit;// Holds the suit of the card
Public Methods
Constructors
PlayingCard();
PlayingCard(int initRank, char initSuit);
Accessors
int getRank();
char getSuit();
void printCard();
string cardToString();
Mutators
void setRank(int initRank);
void setSuit(char initSuit);
Overload operator
void operator=(const PlayingCard& card1);
"PlayingCard.h"
End
PlayingCard::PlayingCard
start
rank = 1
suit =1
PlayingCard::PlayingCard
End
PlayingCard::PlayingCard(int initRank, char initSuit)
start
if
initRank >= 1 && initRank <= 13
else
switch(initSuit)
case 'H','h'
suit = 'H';
break;
case 'D','d'
suit = 'D';
break;
case 'C','c'
suit = 'C';
break;
case 'S','s'
suit = 'S';
break;
default
break;
End

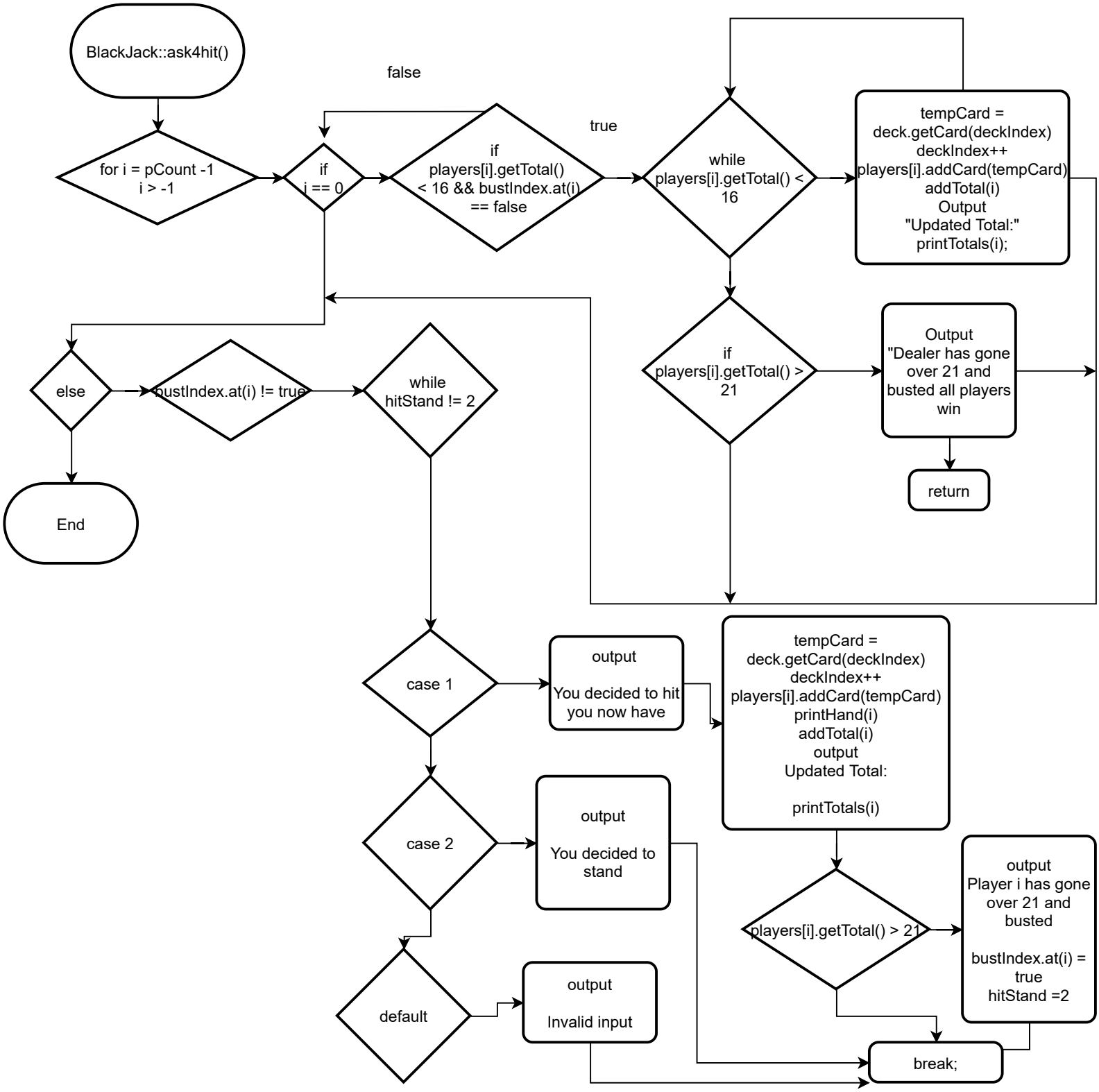

PlayingCard::getRank()
return rank
End
PlayingCard::getSuit()
return suit
End
PlayingCard::setRank(char intitRank)
initRank >= 1 && initRank <= 13
if
rank = initRank
else
End
PlayingCard::setSuit(char intitsuit)
switch(initSuit)
case 'H','h'
suit = 'H'; break;
case 'D','d'
suit = 'D'; break;
case 'C','c'
suit = 'C'; break;
case 'S','s'
suit = 'S'; break;
default
break;
End
void PlayingCard::operator=(const PlayingCard& card1)
this rank = card1 rank
this suit = card1 suit
End

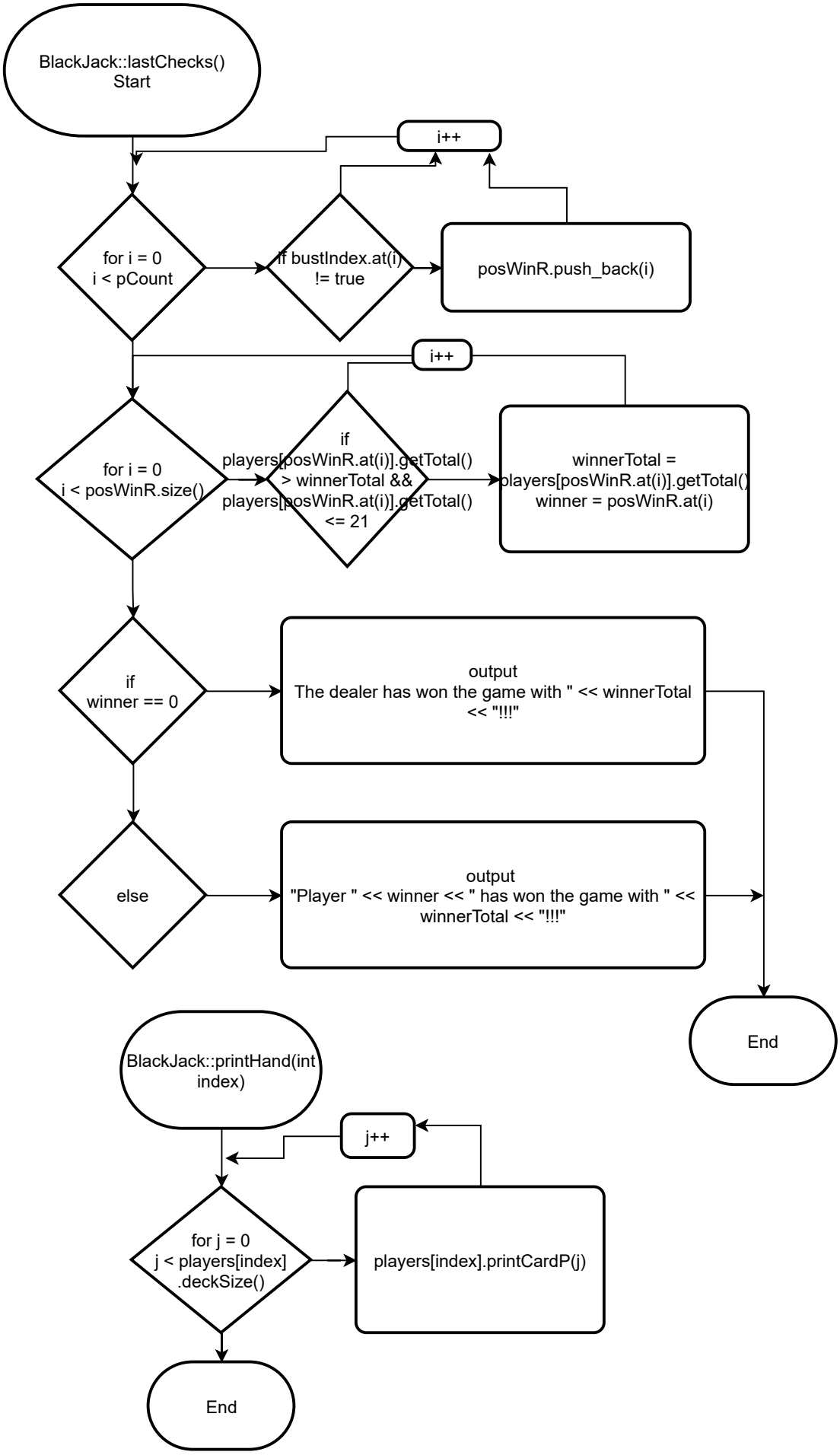

PlayingCard::cardToString()
switch(rank)
case 1
rankSeg = "Ace"
case 2
rankSeg = "Two"
case 3
rankSeg = "Three"
case 4
rankSeg = "Four"
case 5
rankSeg = "Five"
case 6
rankSeg = "Six"
case 7
rankSeg = "Seven"
case 8
rankSeg = "Eight"
case 9
rankSeg = "Nine"
case 10
rankSeg = "Ten"
case 11
rankSeg = "Jack"
case 12
rankSeg = "Queen"
case 13
rankSeg = "King"
switch(initSuit)
case 'H','h'
suit = 'H'; break;
case 'D','d'
suit = 'D'; break;
case 'C','c'
suit = 'C'; break;
case 'S','s'
suit = 'S'; break;
default
break;
cardString = (rankSeg + ofSeg + suitSeg)
End

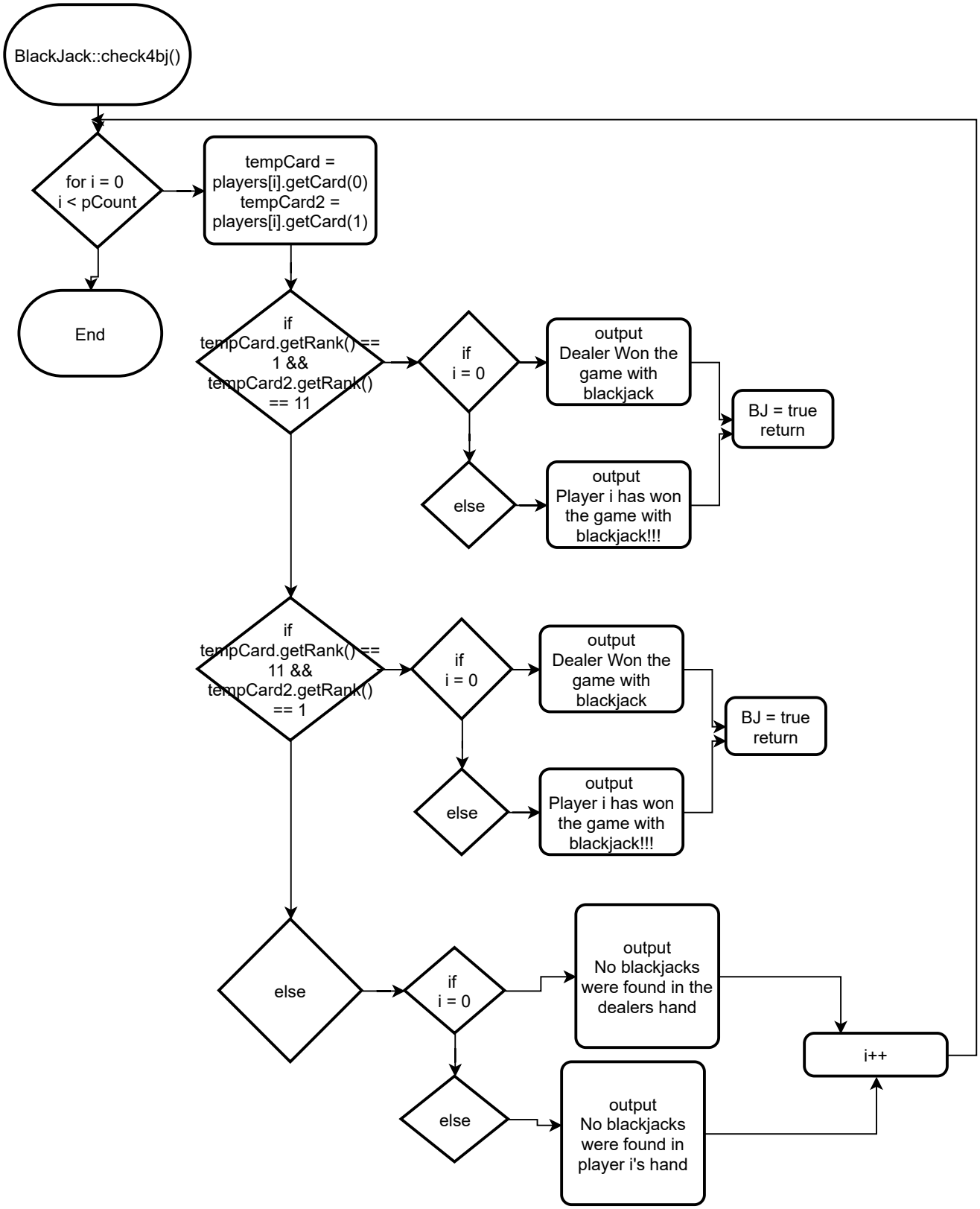

PlayingCard::printCard()
switch(rank)
case 1
output "Ace"
case 2
output "Two"
case 3
output "Three"
case 4
output "Four"
case 5
output "Five"
case 6
output "Six"
case 7
output "Seven"
case 8
output "Eight"
case 9
output "Nine"
case 10
output "Ten"
case 11
output "Jack"
case 12
output "Queen"
case 13
output "King"
Output
"of"
switch(initSuit)
case 'H','h'
output 'Hearts';
break;
case 'D','d'
output 'Diamonds';
break;
case 'C','c'
output 'Clubs';
break;
case 'S','s'
output 'Spades';
break;
default
break;
End

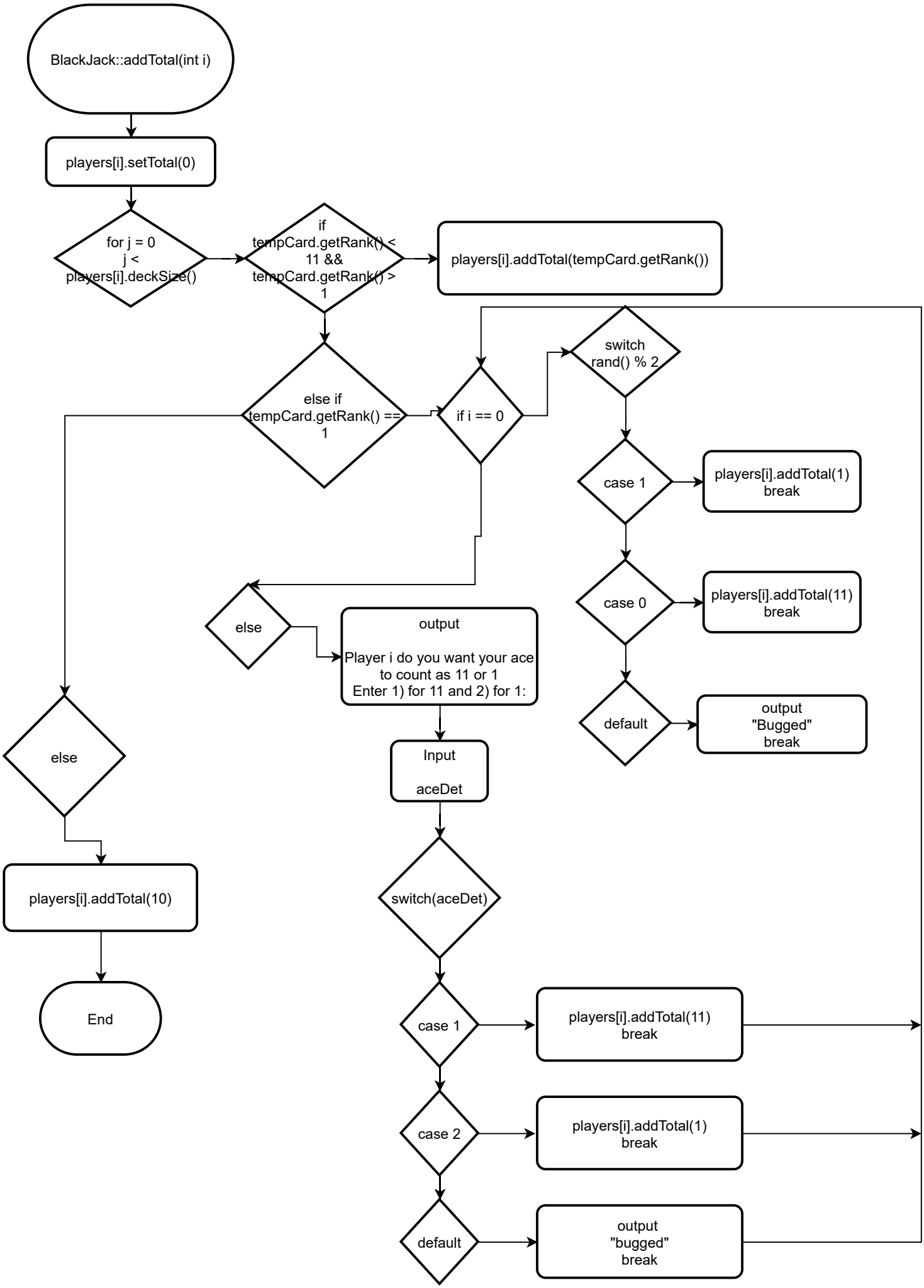

"BlackJack.h"
start
Libraries:
#include "PlayerDeck.h"
#include "CardDeck.h"
private:
members:
int pCount,sCount;//Player Count and Shuffle Count
int deckIndex;//Determines where in the deck to pull a card from
ValidCard tempCard,tempCard2;//Two temp cards to hold value for manipulation
int tempTotal;//Holds the total that gets output to console
CardDeck deck;//The deck for the game
PlayerDeck* players;//Dynamic array of players
vector<bool> bustIndex;//Determines what players went over 21
vector<int> posWinR;//Vector of possible winners
int winner,winnerTotal;//Holds the index of winning player and the card value;
bool BJ,endTurn;//Determines if someone got blackjack and if the turn is over
int aceDet;//Determines if ace is worth 11 or 1
int hitStand;//Determines if the player wanted to hit or stand
public:
methods:
//Constructor
BlackJack();
//Destructor
~BlackJack();
//Starts Game
void startGame();
//Mutator
void addTotal(int i);
void ask4hit();
void dealInitial();
//Accessor
void printTotals(int index);
void printHand(int index);
void lastChecks();
void check4bj();
void check4bust();
"BlackJack.h"
End
BlackJack::BlackJack
start
pCount = 0
sCount = 0
BJ = false
deckIndex = 0
hitStand = 99
winner = 0
winnerTotal = 0
End
BlackJack::~BlackJack
start
delete[] players
End
BlackJack::startgame()
start
Input / Output
Output "Hello, Welcome to Austin's game of Black Jack!!!"
Output "How many players will be playing(MAX = 11)?:"
input pCount
Output "How many times would you like the deck shuffled(MAX = 20):"
input sCount
deck.shufle(sCount)
pCount = pCount + 1
players = new PlayerDeck[pCount]
dealInitial()
check4bj()
BJ = false
for i < pCount
addTotal(i)
i++
Output
"----Totals----"
for i < pCount
printTotals(i)
i++
ask4hit()
lastChecks()
End

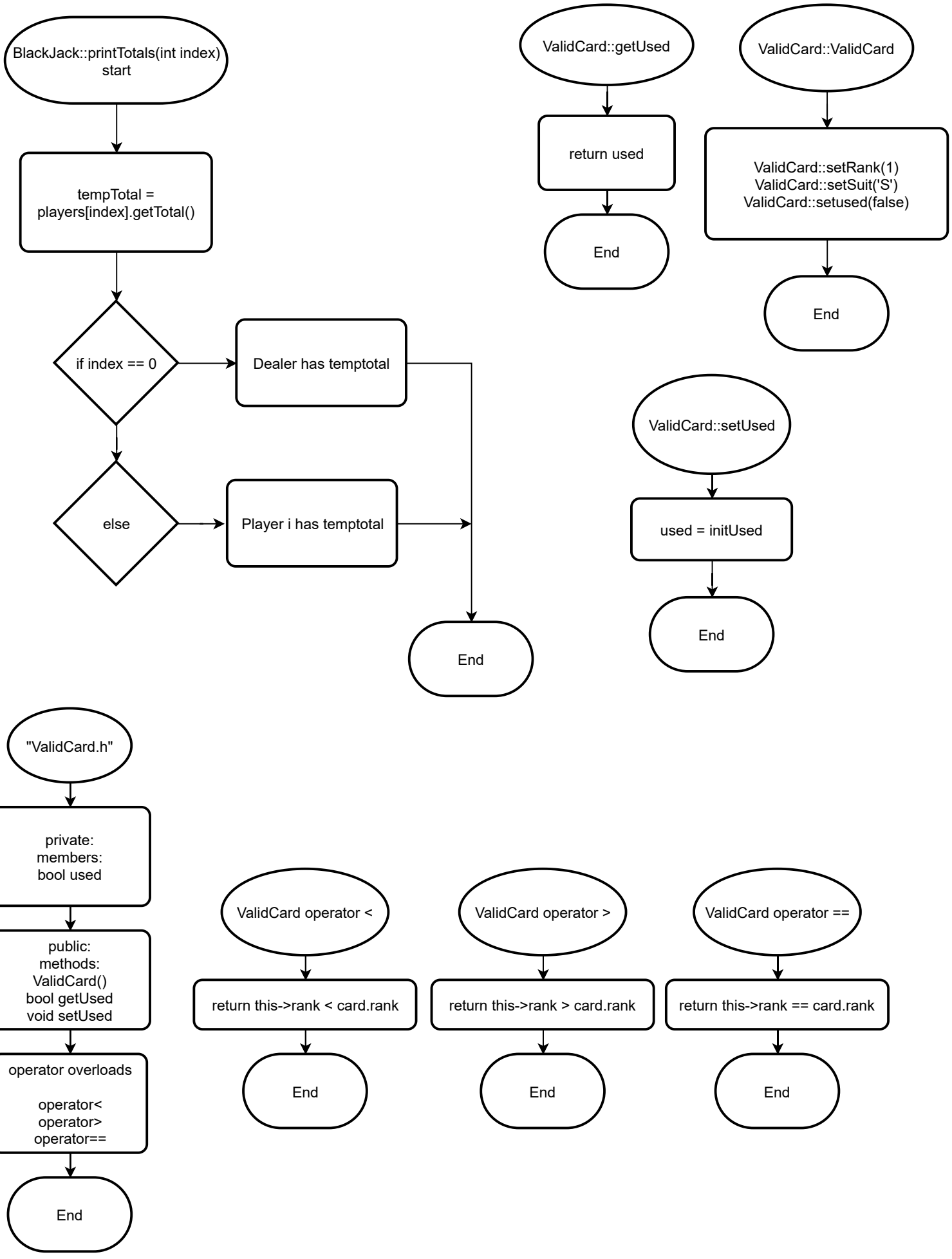

BlackJack::dealInitial()
start
for i < pCount
if i ==0
output
Dealer has these cards
else
output
Player i has these cards
for j < 2
bustIndex.push_back(false)
j++
tempcard =
deck.getCard(deckIndex)
tempCard.getUsed()
== false
deck.rmCard(deckIndex)
players[i].setCard(tempCard,j)
deckIndex++
else
Output
Card Unavailable
printHand(i)
End

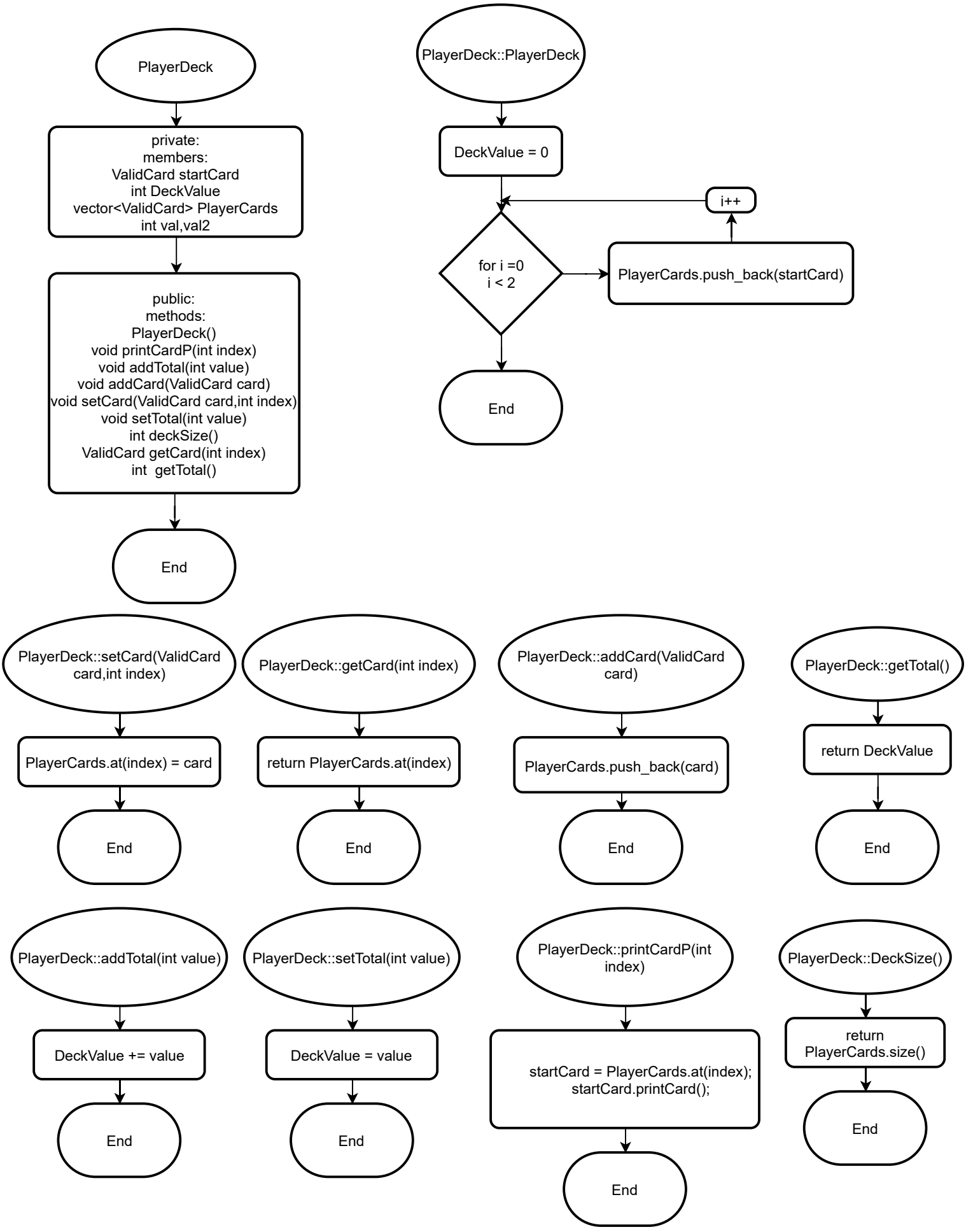

BlackJack::ask4hit()
for i = pCount -1
i > -1
if
i == 0
false
if
players[i].getTotal()
< 16 && bustIndex.at(i)
== false
true
while
players[i].getTotal() <
16
tempCard =
deck.getCard(deckIndex)
deckIndex++
players[i].addCard(tempCard)
addTotal(i)
Output
"Updated Total:"
printTotals(i);
if
players[i].getTotal() >
21
Output
"Dealer has gone
over 21 and
busted all players
win
return
else
bustIndex.at(i) != true
while
hitStand != 2
End
case 1
output
You decided to hit
you now have
tempCard =
deck.getCard(deckIndex)
deckIndex++
players[i].addCard(tempCard)
printHand(i)
addTotal(i)
output
Updated Total:
printTotals(i)
case 2
output
You decided to
stand
players[i].getTotal() > 21
output
Player i has gone
over 21 and
busted
bustIndex.at(i) =
true
hitStand =2
default
output
Invalid input
break;

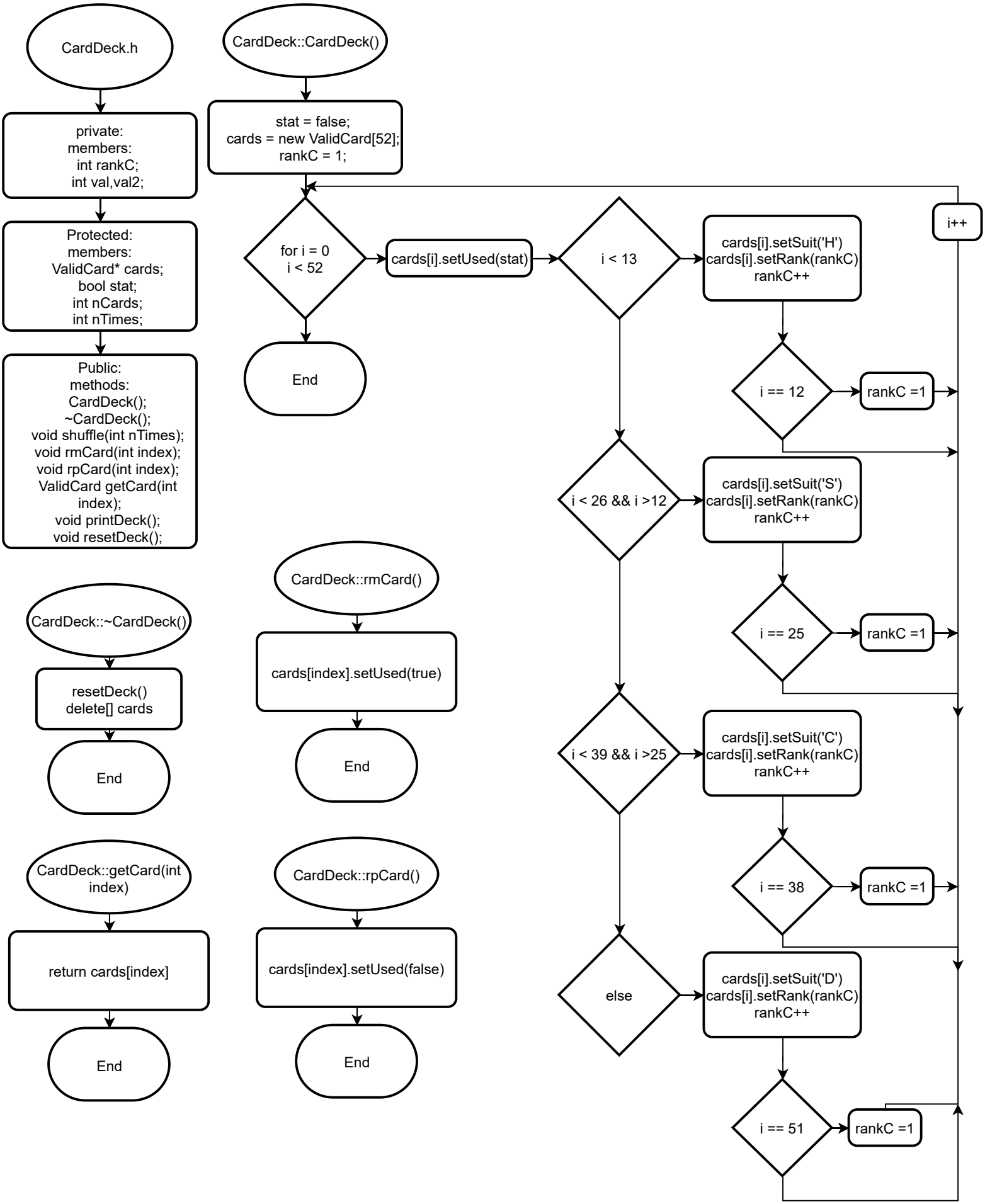

BlackJack::lastChecks()
Start
i++
for i = 0
i < pCount
if bustIndex.at(i)
!= true
posWinR.push_back(i)
i++
for i = 0
i < posWinR.size()
if
players[posWinR.at(i)].getTotal()
> winnerTotal &&
players[posWinR.at(i)].getTotal()
<= 21
winnerTotal =
players[posWinR.at(i)].getTotal()
winner = posWinR.at(i)
if
winner == 0
output
The dealer has won the game with " << winnerTotal
<< "!!!"
else
output
"Player " << winner << " has won the game with " <<
winnerTotal << "!!!"
End
BlackJack::printHand(int
index)
j++
for j = 0
j < players[index]
.deckSize()
players[index].printCardP(j)
End

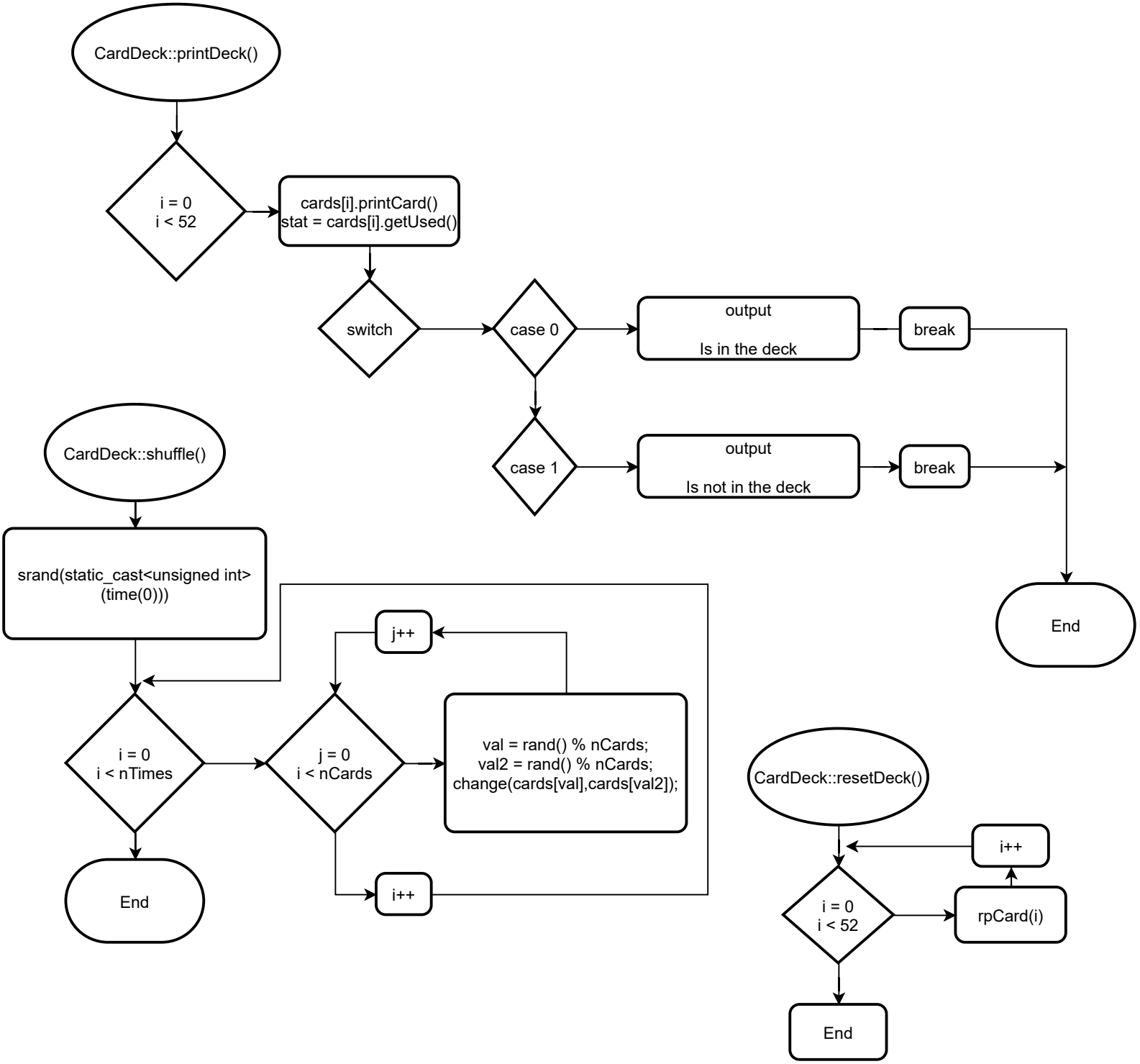

BlackJack::check4bj()
for i = 0
i < pCount
tempCard =
players[i].getCard(0)
tempCard2 =
players[i].getCard(1)
End
if
tempCard.getRank() ==
1 &&
tempCard2.getRank()
== 11
if
i = 0
output
Dealer Won the
game with
blackjack
BJ = true
return
else
output
Player i has won
the game with
blackjack!!!
if
tempCard.getRank() ==
11 &&
tempCard2.getRank()
== 1
if
i = 0
output
Dealer Won the
game with
blackjack
BJ = true
return
else
output
Player i has won
the game with
blackjack!!!
else
if
i = 0
output
No blackjacks
were found in the
dealers hand
else
output
No blackjacks
were found in
player i's hand
i++


BlackJack::addTotal(int i)
players[i].setTotal(0)
for j = 0
j <
players[i].deckSize()
if
tempCard.getRank() <
11 &&
tempCard.getRank() >
1
players[i].addTotal(tempCard.getRank())
else if
tempCard.getRank() ==
1
if i == 0
switch
rand() % 2
case 1
players[i].addTotal(1)
break
case 0
players[i].addTotal(11)
break
default
output
"Bugged"
break
else
output
Player i do you want your ace
to count as 11 or 1
Enter 1) for 11 and 2) for 1:
Input
aceDet
switch(aceDet)
case 1
players[i].addTotal(11)
break
case 2
players[i].addTotal(1)
break
default
output
"bugged"
break
else
players[i].addTotal(10)
End


BlackJack::printTotals(int index)
start
tempTotal =
players[index].getTotal()
if index == 0
Dealer has temptotal
else
Player i has temptotal
End
ValidCard::getUsed
return used
End
ValidCard::ValidCard
ValidCard::setRank(1)
ValidCard::setSuit('S')
ValidCard::setused(false)
End
ValidCard::setUsed
used = initUsed
End
"ValidCard.h"
private:
members:
bool used
public:
methods:
ValidCard()
bool getUsed
void setUsed
operator overloads
operator<
operator>
operator==
End
ValidCard operator <
return this->rank < card.rank
End
ValidCard operator >
return this->rank > card.rank
End
ValidCard operator ==
return this->rank == card.rank
End


PlayerDeck
private:
members:
ValidCard startCard
int DeckValue
vector<ValidCard> PlayerCards
int val,val2
public:
methods:
PlayerDeck()
void printCardP(int index)
void addTotal(int value)
void addCard(ValidCard card)
void setCard(ValidCard card,int index)
void setTotal(int value)
int deckSize()
ValidCard getCard(int index)
int getTotal()
End
PlayerDeck::PlayerDeck
DeckValue = 0
i++
for i =0
i < 2
PlayerCards.push_back(startCard)
End
PlayerDeck::setCard(ValidCard card,int index)
PlayerCards.at(index) = card
End
PlayerDeck::getCard(int index)
return PlayerCards.at(index)
End
PlayerDeck::addCard(ValidCard card)
PlayerCards.push_back(card)
End
PlayerDeck::getTotal()
return DeckValue
End
PlayerDeck::addTotal(int value)
DeckValue += value
End
PlayerDeck::setTotal(int value)
DeckValue = value
End
PlayerDeck::printCardP(int index)
startCard = PlayerCards.at(index);
startCard.printCard();
End
PlayerDeck::DeckSize()
return PlayerCards.size()
End


CardDeck.h
private:
members:
int rankC;
int val,val2;
Protected:
members:
ValidCard* cards;
bool stat;
int nCards;
int nTimes;
Public:
methods:
CardDeck();
~CardDeck();
void shuffle(int nTimes);
void rmCard(int index);
void rpCard(int index);
ValidCard getCard(int index);
void printDeck();
void resetDeck();
CardDeck::CardDeck()
stat = false;
cards = new ValidCard[52];
rankC = 1;
for i = 0
i < 52
cards[i].setUsed(stat)
End
i < 13
cards[i].setSuit('H')
cards[i].setRank(rankC)
rankC++
i++
i == 12
rankC =1
i < 26 && i >12
cards[i].setSuit('S')
cards[i].setRank(rankC)
rankC++
i == 25
rankC =1
i < 39 && i >25
cards[i].setSuit('C')
cards[i].setRank(rankC)
rankC++
i == 38
rankC =1
else
cards[i].setSuit('D')
cards[i].setRank(rankC)
rankC++
i == 51
rankC =1
CardDeck::~CardDeck()
resetDeck()
delete[] cards
End
CardDeck::rmCard()
cards[index].setUsed(true)
End
CardDeck::getCard(int index)
return cards[index]
End
CardDeck::rpCard()
cards[index].setUsed(false)
End


CardDeck::printDeck()
i = 0
i < 52
cards[i].printCard()
stat = cards[i].getUsed()
switch
case 0
output
Is in the deck
break
case 1
output
Is not in the deck
break
End
CardDeck::shuffle()
srand(static_cast<unsigned int>
(time(0)))
i = 0
i < nTimes
j = 0
i < nCards
val = rand() % nCards;
val2 = rand() % nCards;
change(cards[val],cards[val2]);
j++
i++
End
CardDeck::resetDeck()
i = 0
i < 52
rpCard(i)
i++
End