**Project – 설명**

**20182931 신석경**

Project: 블랙잭 게임을 구현하라

1. 기본 사항 및 규칙

먼저, 블랙잭 게임은 카드를 open하며 21에 가장 가까운 사람이 이기는 게임이다.

블랙잭은 3인(dealer, player1, player2)로 진행된다. 이때, dealer는 컴퓨터이고 나머지는 사람이 진행한다.

블랙잭 게임의 규칙은 다음과 같다.

1. 플레이어는 금액을 베팅하고 게임이 시작되면 3인 모두 2장의 카드를 받는다.
2. Player1 -> Player2 -> Dealer 순으로 각각 카드를 open 혹은 stop 할 것인지를 정한다. 이때, 현재 차례의 사람은 open을 계속할 수도 있고, stop할 수도 있다.
3. 위의 상황에서 open을 하던 중, 카드 합의 총합이 21이 넘는 경우 BUST로 게임에서 지게 되고, 21이 되는 경우 BLACKJACK으로 베팅 금액의 2.5배에 해당하는 금액을 받는다. 그 이외의 경우에는 딜러의 카드 합과 비교하여 더 큰 경우 2배의 배당, 작은 경우 돈을 읽고, 같으면 베팅 금액을 다시 돌려받는다.
4. 위의 사항을 반복한다.

위와 같은 규칙으로 게임이 진행된다. 세부적인 규칙은 아래와 같다.

1. A의 경우 1 혹은 11로 카드 숫자를 결정하고 11로 계산하여 21이 넘는다면 무조건 1로 계산한다.
2. 딜러의 경우, 현재 카드 합이 17 보다 작으면 무조건 카드를 open해야 한다.

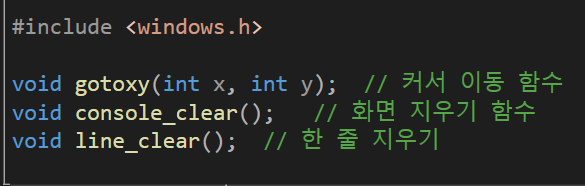
위의 규칙만 알면 블랙잭을 시작할 수 있다.

1. 게임을 구현하기 위한 코드

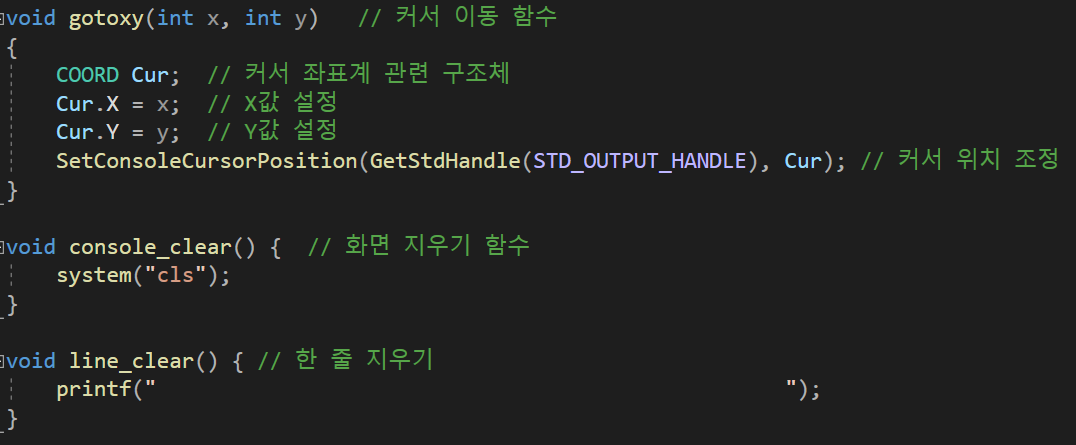
이 게임을 구현하기 위해서는 다양한 코드가 필요하다. 많은 부분에 각자의 책임이 있기 때문에 모듈화를 진행하였다.

1. cursor.h && cursor.c

cursor 모듈은 콘솔 출력을 위한 커서 이동 및 클리어를 담당하는 모듈이다.



cursor.h에 정의된 것들이다. 먼저 콘솔을 조작하기 때문에 windows.h를 include하였다. 나머지 함수에 대해서는 cursor.c에서 의미와 내용을 살펴보겠다.



각각의 함수이다. 먼저 gotoxy() 함수는 커서를 x, y 위치로 이동하는 함수이다. COORD 구조체 변수를 선언하여 Cur의 X와 Y에 입력된 x, y를 할당하고, SetConsoleCursorPosition()함수를 사용하여 현재 커서 위치를 조정한다.

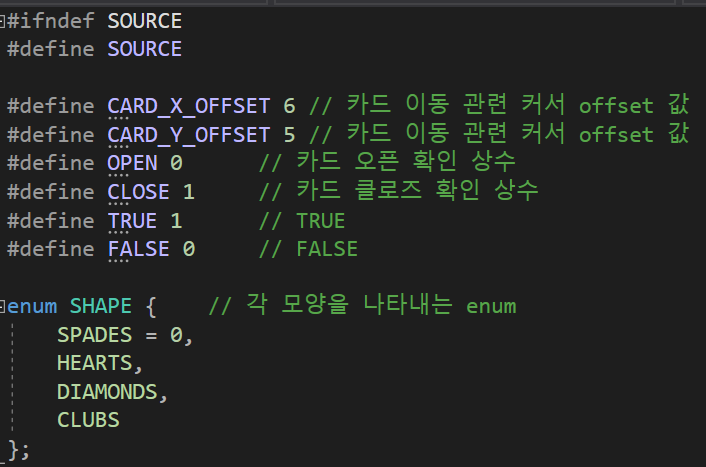
console\_clear() 함수는 system(“cls”)를 사용하여 콘솔창 전체를 삭제한다.

line\_clear()의 경우 현재 줄에 공백을 출력하여 콘솔창 한 줄을 삭제한다.

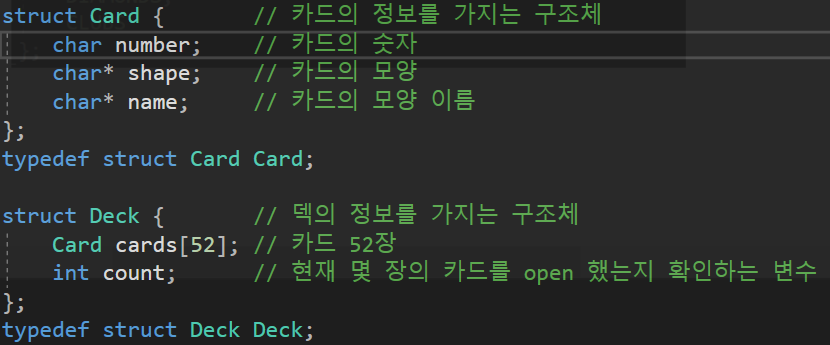
보통 gotoxy()로 해당 장소로 이동한 후 line\_clear()를 실행하여 그 줄을 삭제한다.

1. source.h && source.c

source 모듈은 게임을 진행하기 위한 자료형과, 그 자료형의 정보와 내부계산과 관련된 모듈이다. 먼저 source.h부터 살펴보자.



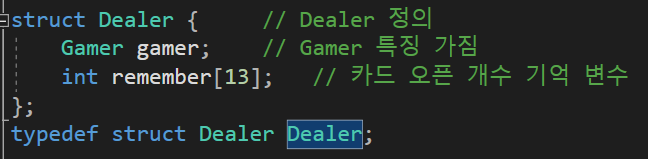
먼저 필요한 상수 정의부터 한다. CARD\_X\_OFFSET은 Card를 그리는 과정에서 x축으로의 이동 offset이고, CARD\_Y\_OFFSET은 y축으로의 이동 offset이다. OPEN과 CLOSE는 카드를 그리는 과정에서 앞면으로 그릴 지, 뒷면으로 그릴 지 정하는 상수이다. 그리고 기본적인 TRUE와 FALSE가 있다. 그리고 각 모양을 나타내는 열거형이 있다.



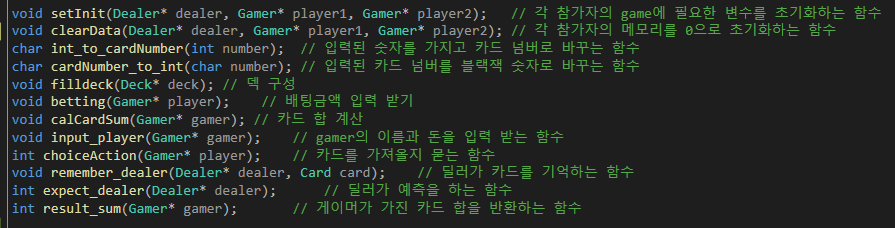
카드와 덱을 나타내는 구조체이다. Card 구조체의 경우, 카드의 숫자를 기억하는 number char 변수가 있고, 카드의 모양과 이름을 기억하는 shape, name char\* 변수가 있다. Deck 구조체 경우, 52길이인 Card 구조체 배열을 가지고, 몇 개의 카드를 open했는지 기억하는 count 변수를 선언한다.



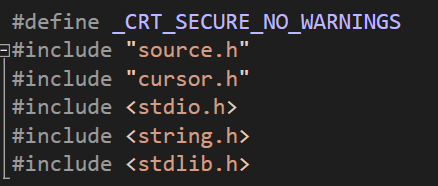
사람과 게이머를 나타내는 Person과 Gamer 구조체이다. 먼저 Person의 경우 이름을 가지는 name char 배열과 보유 금액을 나타내는 money 변수를 가지고 있다. Gamer의 경우, 먼저 Person 변수 person을 가지고, 가지고 있는 카드를 나타내는 Card 구조체 배열 cards를 가지고 있다. bet\_money는 베팅 금액을 저장하는 변수이고, num은 현재 카드 개수를 나타낸다. 카드 합을 저장하는 sum 변수를 배열로 선언한 이유는 A가 1 혹은 11로 계산되기 때문에 각각의 상황별로 계산하기 위함이다. 그리고 A를 가졌는지 판단하는 hasA, 진 경우를 판단하는 isLose, 블랙잭 인지 판단하는 isBlackJack, 멈추는 것을 판단하는 isStop 변수를 가진다.



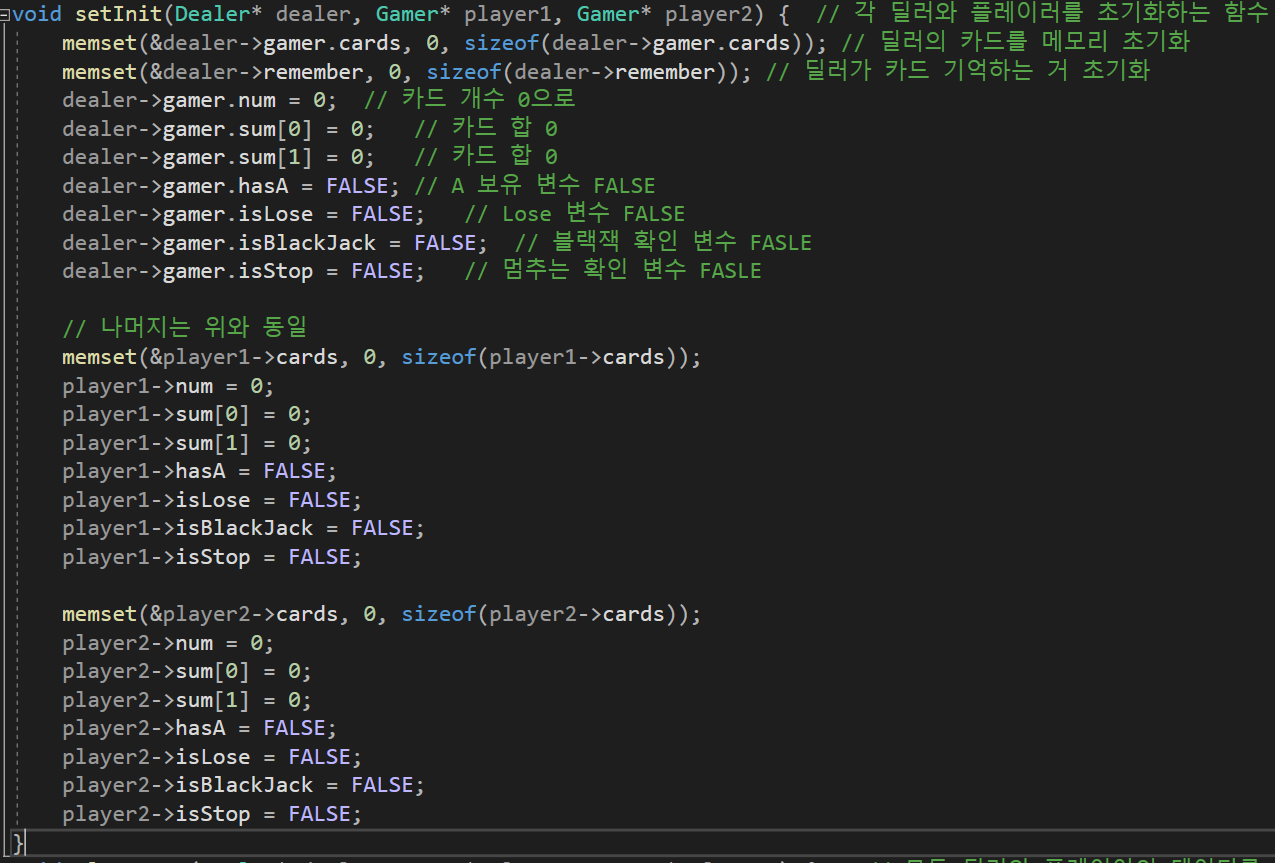
Dealer 구조체이다. Dealer의 경우 나머지는 Gamer와 똑같아 Gamer 변수를 가진다. 추가된 것은 현재 오픈된 카드를 기억하는 remember int 배열이 추가된 것이다.



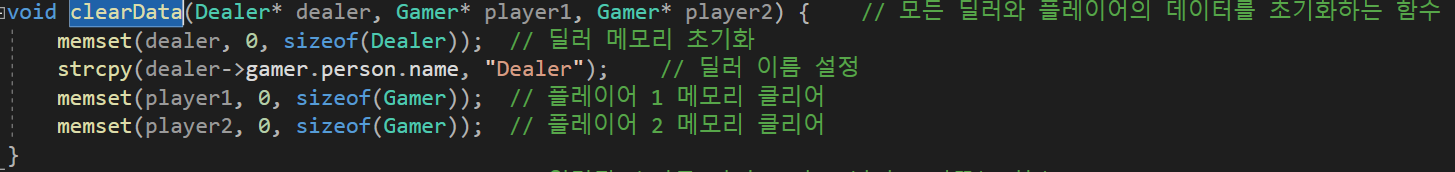
source 모듈에서 사용하는 함수이다. 각각의 의미와 내용은 source.c에서 살펴보겠다.



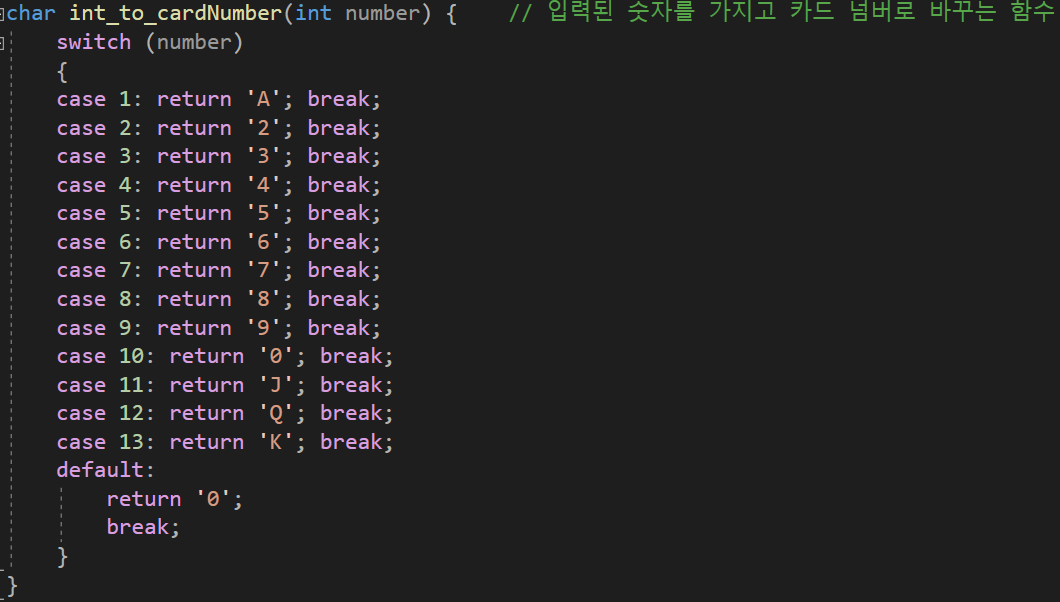
source.c의 include이다. source.h와 cursor.h를 include하고, string 연산을 위한 string.h, 동적 할당을 사용하기 위한 stdlib.h를 include한다.



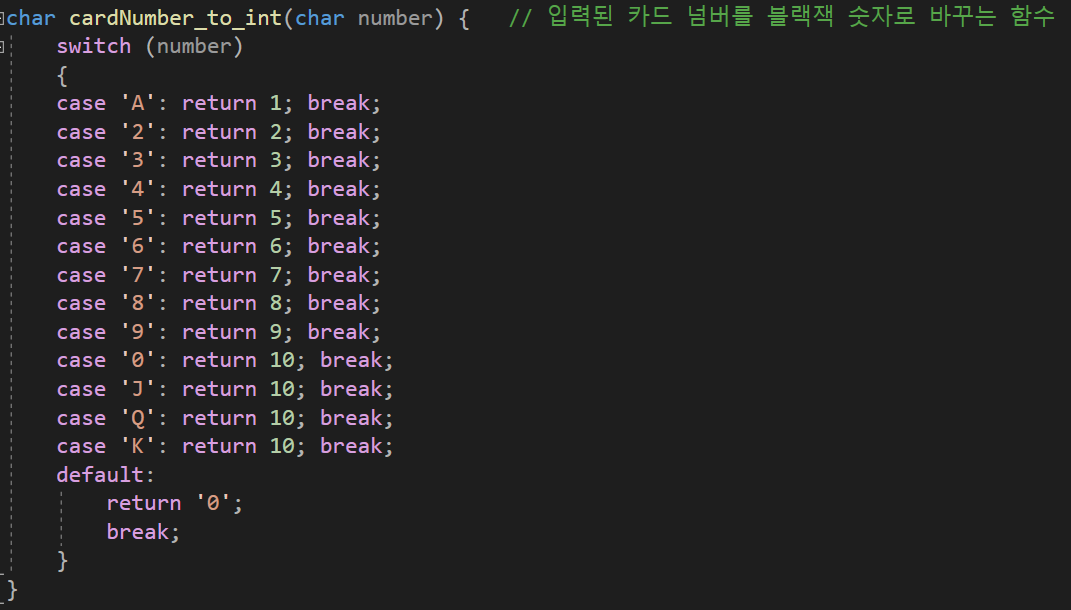
setInit함수이다. 이 함수는 각 딜러와 플레이어의 게임과 관련된 변수를 초기화하는 함수이다. 즉, 플레이어의 정보를 초기화하는 것이 아니라 게임과 관련된 정보를 초기화하는 것이다. 따라서 각 딜러와 플레이어가 게임과 관련된 변수를 초기화한다. 보통, 3인 모두 num, sum, hasA, isLose, isBlackJack, isStop을 초기화 하지만, dealer의 경우 카드를 기억하는 변수 remember도 존재하지 때문에 그것도 memset을 통해 메모리를 초기화해준다.



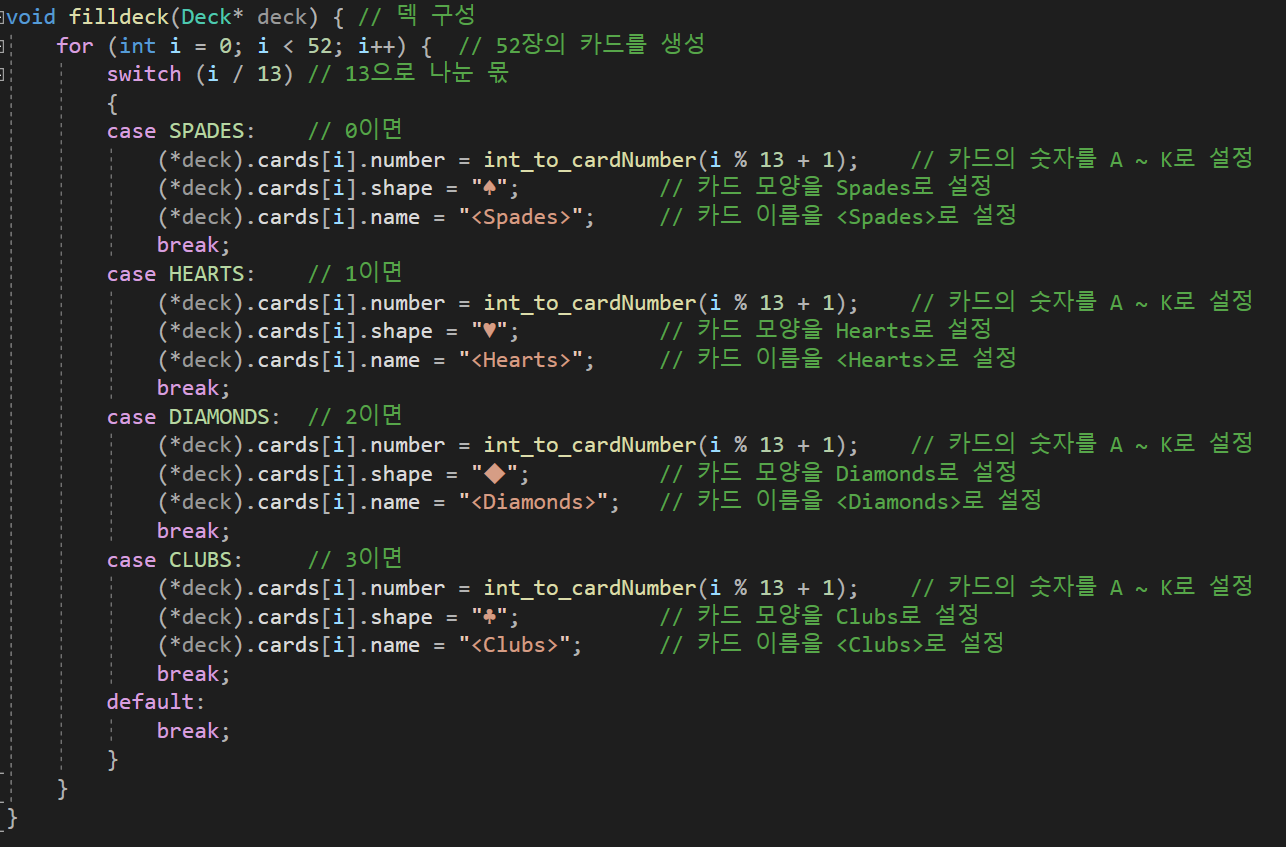
clearData() 함수이다. 이 함수는 3명의 모든 정보를 초기화하는 함수이다. 이것은 보통 한 참가자를 바꾸기 전에 실행하는 함수이다. 3명의 모든 메모리를 0으로 초기화하고 Dealer의 경우에만 이름을 Dealer로 설정하여 보내준다.



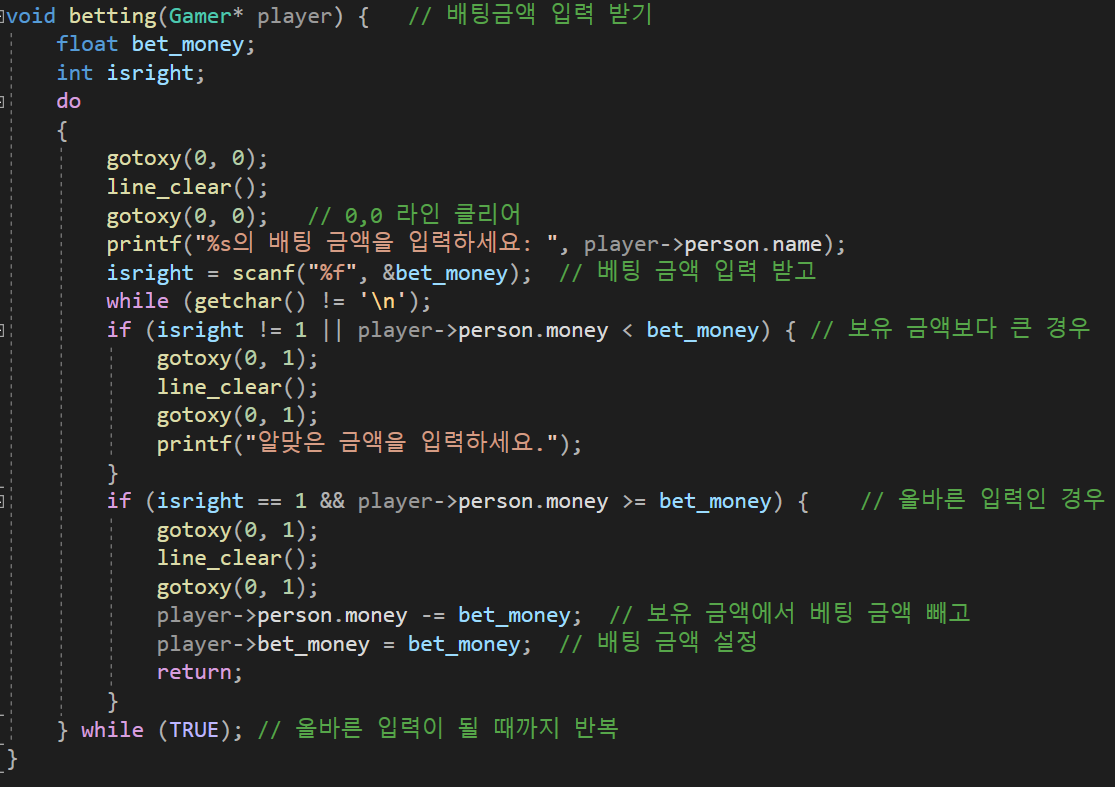
Int\_to\_cardNumber() 함수이다. 입력된 number를 가지고 char로 변환하여 반환한다. 1의 경우 A, 11의 경우 J, 12의 경우 Q, 13의 경우 K로 출력한다. 10의 경우 0으로 출력하는데 그 이유는 char이기 때문에 한 문자만 저장가능하기 때문이다. 따라서 앞으로 ‘0’은 10이라고 생각하자.



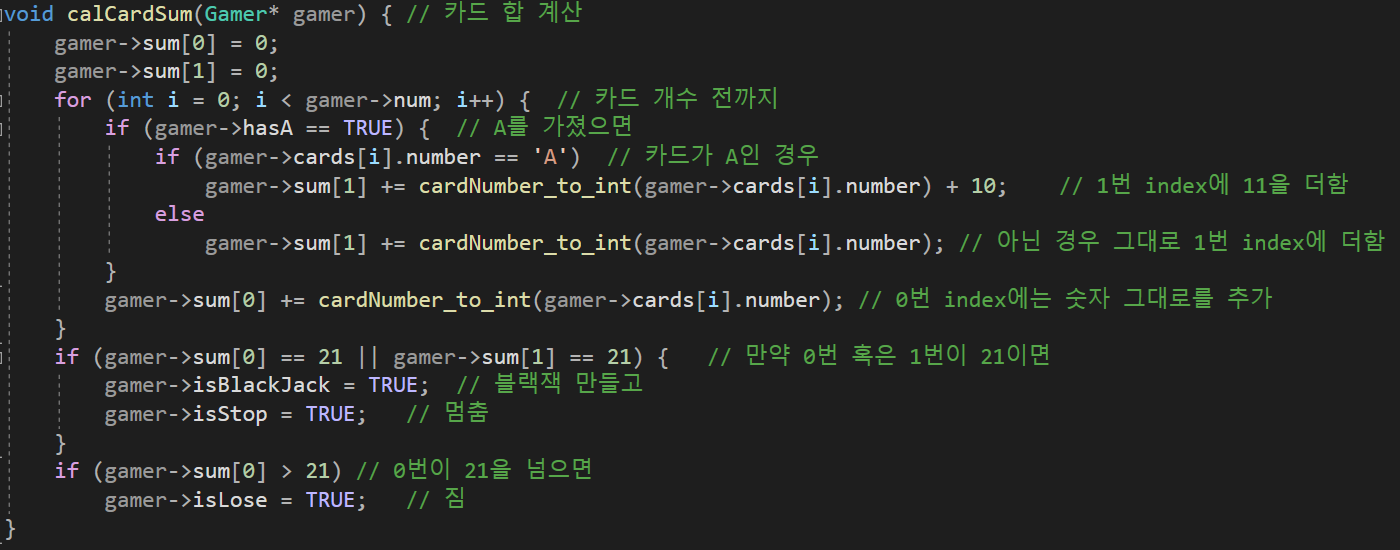
cardNumber\_to\_int이다. 이것은 보통 카드 합을 계산하기 위함으로 사용되는데, 그래서 잘 보면 J, Q, K인 경우 모두 10으로 반환하는 것을 알 수 있다. 그리고 ‘0’인 경우도 10으로 반환하는 것을 알 수 있다.



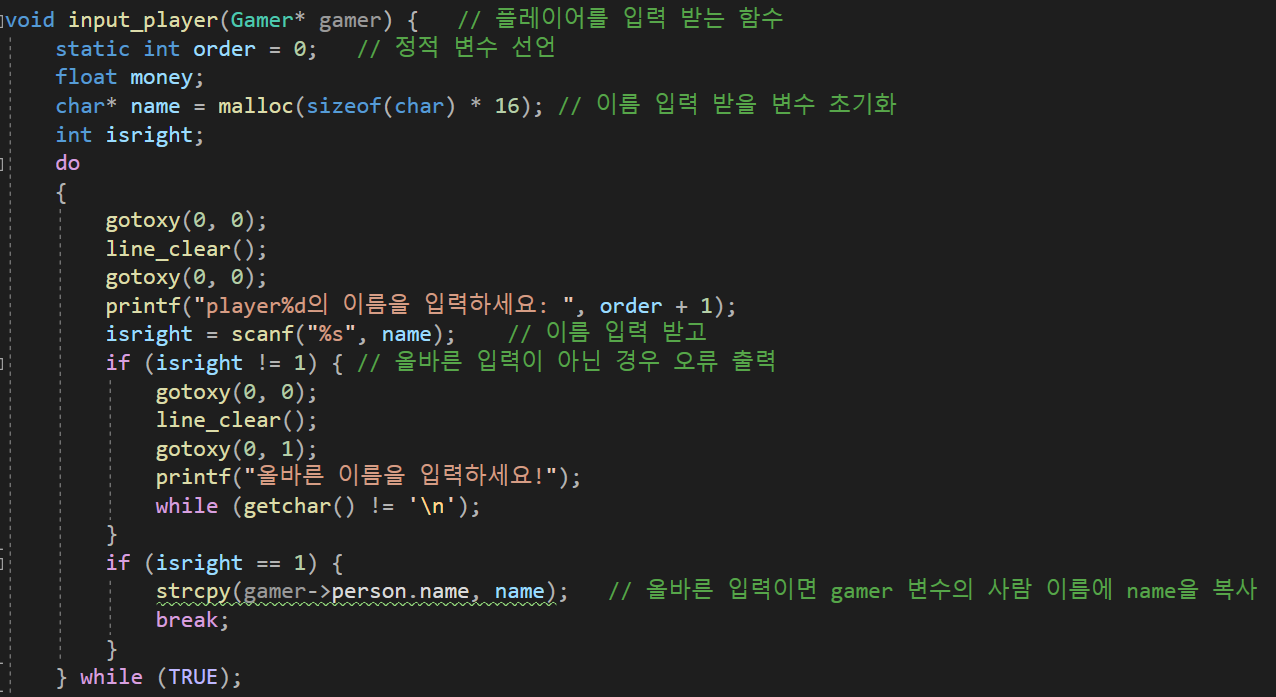
덱을 구성하는 filldeck() 함수이다. 이 함수의 경우 이전 과제에서 설명하였지만 다시 설명하면 , 52까지의 숫자를 가지고 13으로 나누었을 때, 그것이 SPADES이면, 13으로 나머지 연산한 값으로 카드를 할당하고, 그에 해당하는 모양과 이름을 할당한다. 이와 같이 반복하여 덱을 구성할 수 있다.



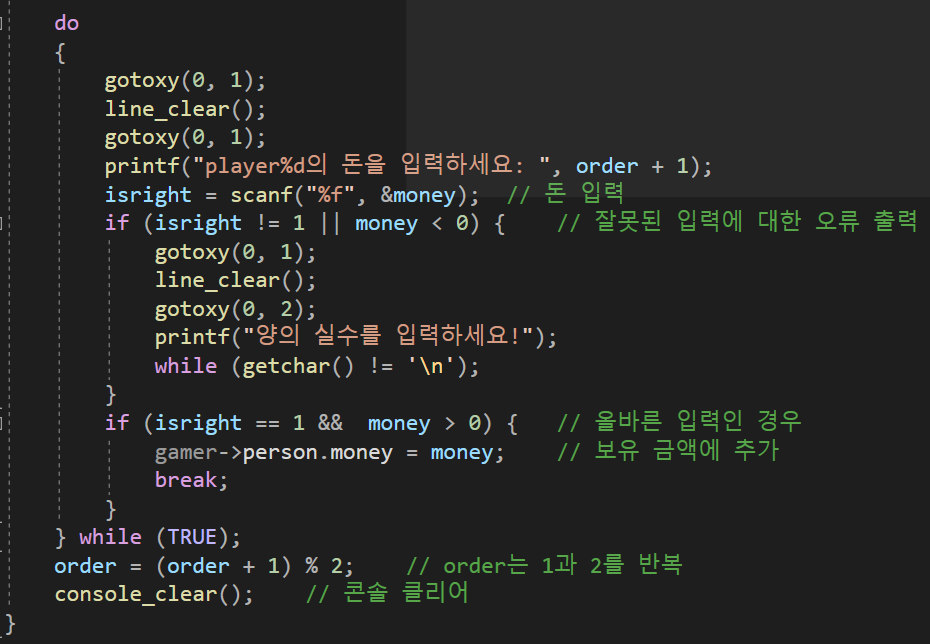
Player의 베팅 금액을 입력 받는 betting() 함수이다. 먼저 베팅을 받기 위한 질문을 출력하기 위해 0,0으로 이동하여 line\_clear()를 하고, 다시 0,0으로 이동한다. 그리고 그 자리에 player의 이름과 함께 베팅할 금액을 입력 받을 질문을 출력한다. 그리고 베팅 금액을 입력 받아 bet\_money에 저장한다. 이때, 베팅 금액이 올바르지 못하거나, 현재 보유 금액보다 작다면 0,1 라인에 오류를 출력하고, 다시 while문을 실행하여 다시 입력 받는다. 올바른 입력인 경우, 0,1에 있을 수도 있는 오류 메시지를 삭제하고 보유 금액에서 베팅 금액을 빼고, 베팅 금액을 설정하고 반환한다. 이 과정을 올바른 입력이 될 때까지 반복한다.



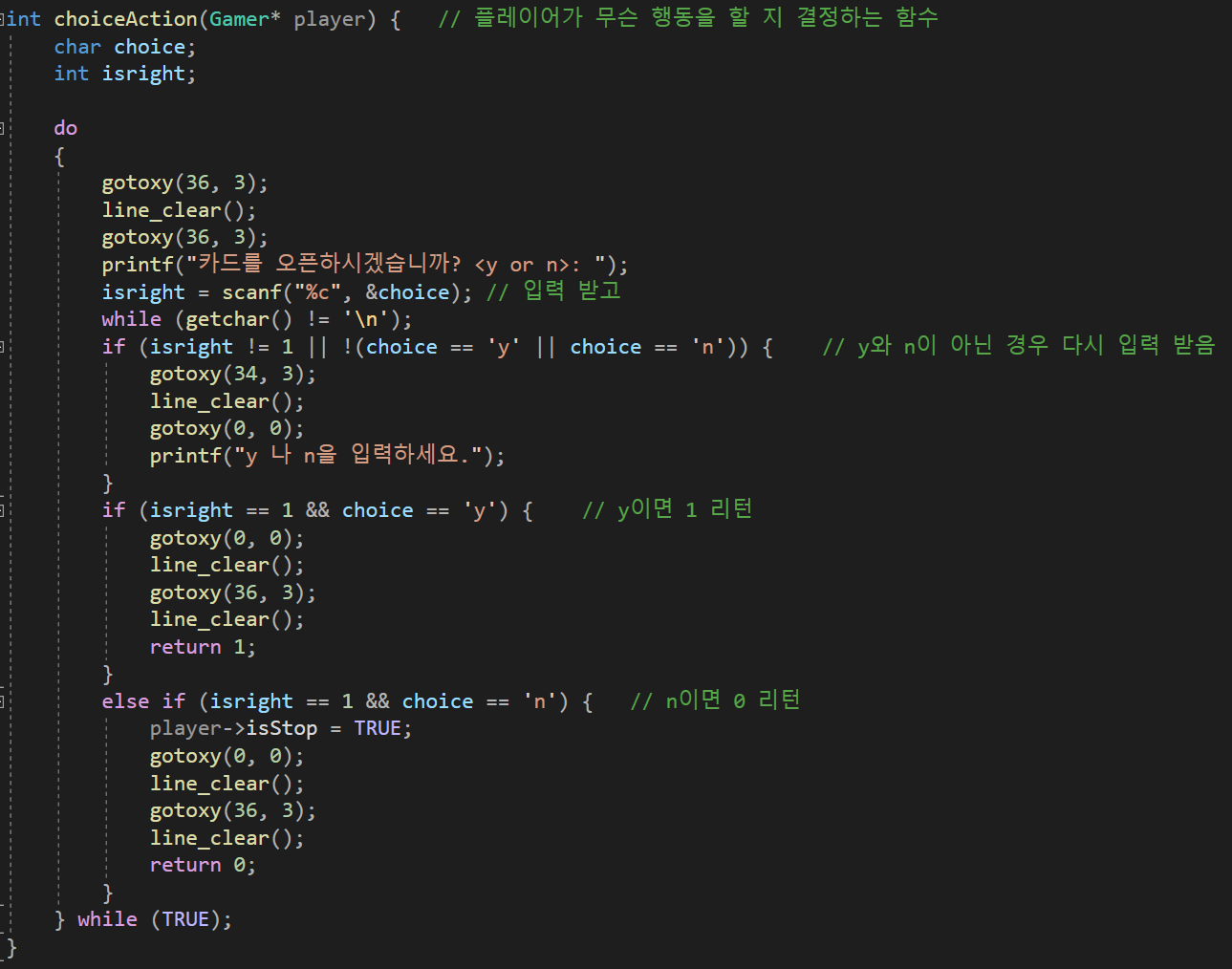
Gamer의 카드 합을 계산하는 함수 calCardSum()이다. 먼저 입력된 gamer의 sum[0], sum[1]을 0으로 초기화하고, gamer가 가진 카드 개수 전까지 카드를 검사한다. 만약 카드의 숫자가 ‘A’이면, sum[1]에 11을 더하고, 아니면 그냥 sum[1]에 해당하는 숫자를 더한다. 그리고 sum[0]에는 항상 카드의 숫자를 더한다. 만약 계산을 끝낸 다음 sum[0] 혹은 sum[1]이 21인 경우 isBlackJack과 isStop을 TRUE로 만들어 블랙잭으로 설정한다. 만약 sum[0]이 21을 넘으면 isLose를 TRUE로 하여 진 것을 확인한다. 마지막에 sum[1]을 확인하지 않는 이유는 sum[1]은 항상 sum[0]보다 크기 때문에(‘A’를 11로 더하기 때문) sum[0]이 21보다 크면 항상 sum[1]도 21보다 크기 때문이다.



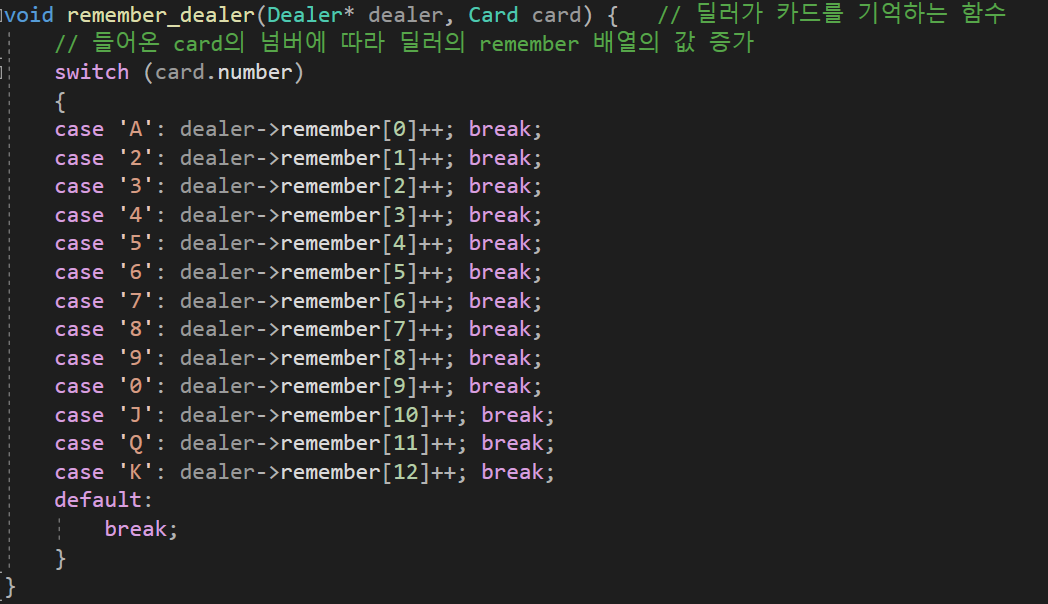
플레이어의 정보를 입력 받는 input\_player함수이다. 이 함수는 길어서 두 부분으로 살펴보겠다. 먼저 각 정보를 입력 받을 변수를 할당해준다. order의 경우 static 변수로 선언하여 항상 일정한 값을 순회하도록 할 것이다. 먼저 이름을 입력 받는다. 올바른 입력일 경우에만 gamer->person.name으로 복사하고 아닌 경우 계속 입력을 새로 받는다.



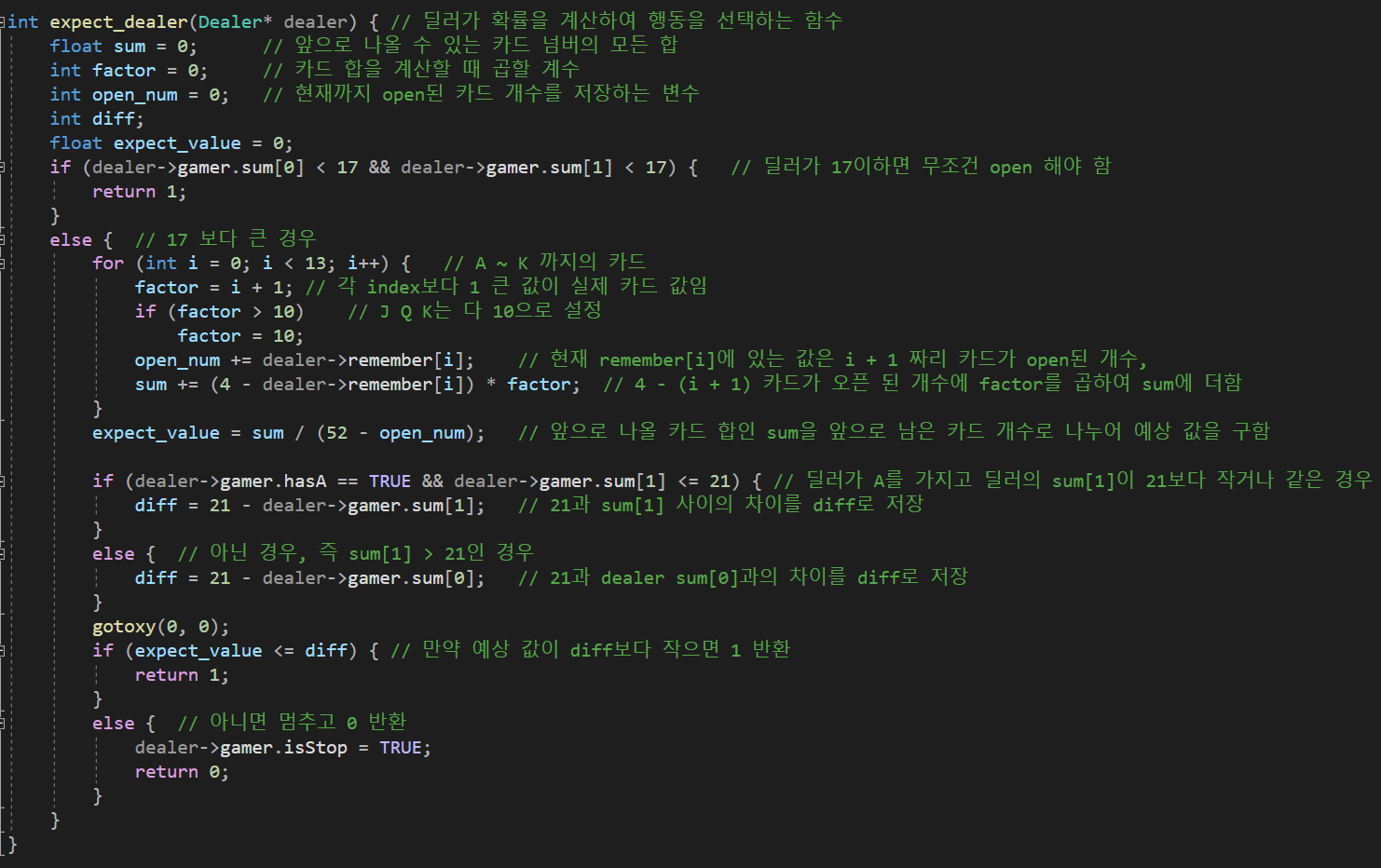
이름을 입력 받으면 보유 금액을 입력 받는다. 이 경우에도 입력을 받고 올바른 입력인 경우에만 gamer->person.money에 그 값을 할당하고 아닌 경우 다시 입력을 받는다. 모든 입력이 끝나면 order를 1과 2를 반복하여 순회하도록 설정하고, 콘솔을 클리어 하여 입력을 종료한다.



플레이어가 어떤 행동을 할 지 입력 받는 choiceAction 함수이다. 함수의 내부 구조를 보면 알겠지만, 어떤 입력을 받는 부분은 대부분 비슷하게 흘러간다. 이때에도 y 혹은 n을 입력 받아, y인 경우에는 1을 반환, n인 경우에는 0을 반환하고 나머지의 경우에는 오류를 출력하며 다시 입력을 받도록 한다.

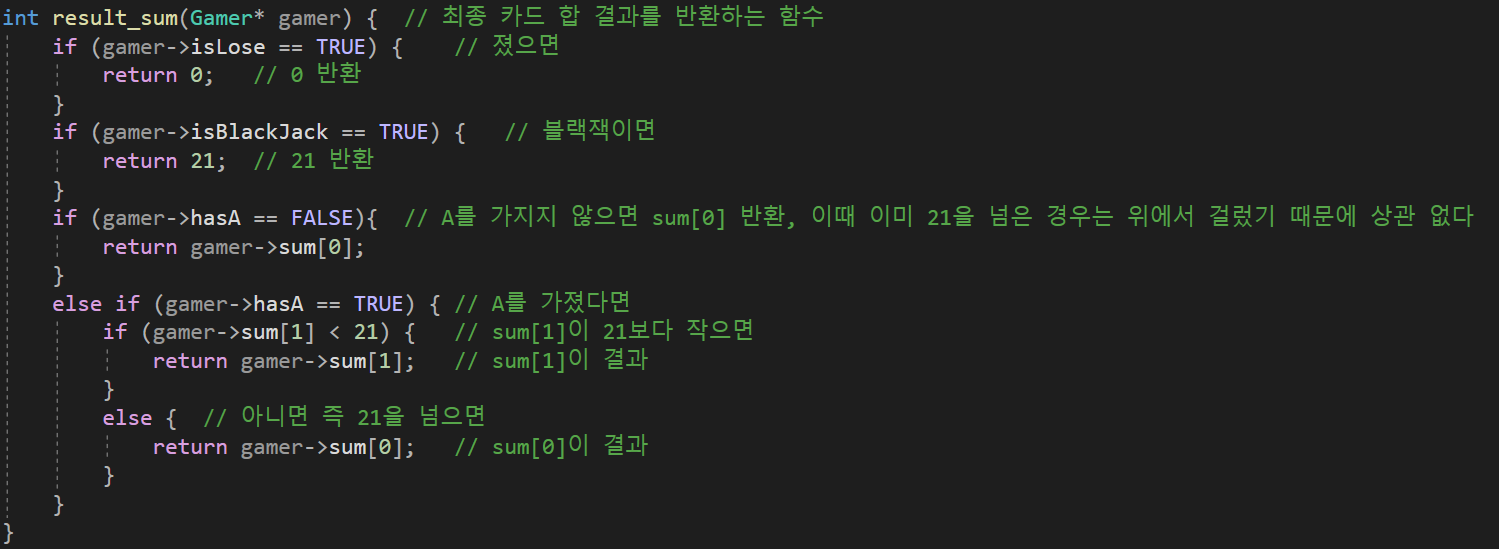


Dealer가 오픈된 card를 기억하도록 하는 함수 remember\_dealer이다. 들어온 card의 number를 확인하여 각 경우에 맞게 dealer->remember 배열의 특정 index의 값을 1 증가시킨다. 0번 index에는 ‘A’의 개수가, 1번 index에는 ‘2’의 개수가 들어가는 식이다.



이제 기억된 카드를 가지고 dealer의 행동을 결정하는 expect\_dealer 함수이다. 먼저 sum 변수는 앞으로 나올 수 있는 카드 합이고, factor는 카드 총합을 계산할 때 곱할 계수이다. open\_num은 현재 open된 카드의 개수를 저장하는 변수이다. diff는 21과 dealer 카드 합의 차이를 저장하는 변수이고, expect\_value는 예상되는 카드 넘버 기대값이다.

먼저 앞서 규칙에서 설명했듯이, dealer의 카드 합이 17보다 이하면 무조건 카드를 open해야 하기 때문에 1을 반환한다. 17보다 큰 경우, 0 ~ 12index의 remember 변수를 검사한다. 이때, factor에는 실제 카드 값인 i + 1이 들어가고 J, Q, K의 경우 다 10으로 설정해준다. 이후 open\_num에 현재 remember[i]의 값을 더하고, (4-remember[i])를 하여 i+1 카드가 open되지 않는 개수에 factor를 곱하여 예상되는 기대값을 sum에 더한다. 이 for문이 끝나면 sum에는 앞으로 나올 카드의 모든 합이 저장되고, open\_num에는 현재까지 open된 카드 개수가 저장된다. 그러면 sum/(52-open\_num)을 하므로 앞으로 카드를 open할 때 나올 수 있는 기대값을 구할 수 있는 것이다. 그리고 diff를 구하는데, dealer가 A를 가지고 sum[i]이 21보다 작거나 같은 경우 diff는 21 – sum[1]이고 다른 경우 diff는 21 – sum[0]이 된다. 그 이유는 앞서 설명했듯이 sum[1]이 항상 sum[0]보다 크기 때문에 sum[1]이 21보다 작으면 sum[0]보다 훨씬 21에 가깝기 때문이다. 그런 다음 expect\_value가 diff 보다 작거나 같은 경우 카드를 open하는 것이 유리하기 때문에 1을 반환하고, 아니면 isStop을 TRUE로 하고 0을 반환한다.

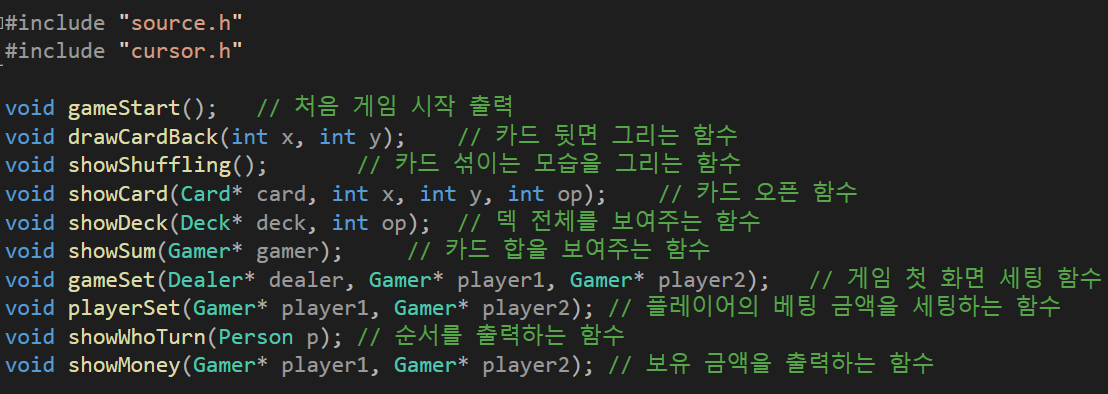


gamer의 최종 카드 합을 계산하는 함수 result\_sum이다. 먼저 isLose가 TRUE이면 무조건 진 것이기 때문에 0을 반환한다. 그리고 isBlackJack인 경우 21을 반환한다. 그 이후에 hasA가 FALSE이면 sum[0]을 반환한다. 이때, 21을 초과하는 경우는 이미 앞서 isLose 검사에서 처리하기 때문에 그냥 반환해도 상관없다. 그리고 hasA가 TRUE이면, sum[1]이 21보다 작으면 sum[1]을, 아닌 경우 sum[0]을 반환한다.

지금까지 source 모듈에 대해 살펴보았다. 이제는 이것을 이용하여 다른 곳에서 사용할 수 있다.

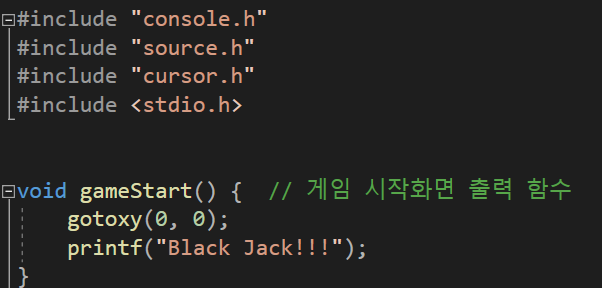
1. console.h && console.c

console 모듈의 경우, console 창에 출력하는 것들을 모아 놓은 모듈이다. 즉, 그리는 이미지들을 모아 놓았다고 생각하면 된다.

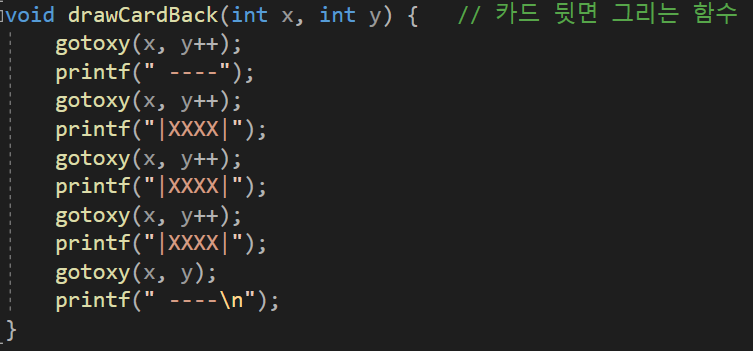


먼저 console.h에 정의된 것들이다. source에 정의된 것들과 cursor에 정의된 것들을 사용하기 때문에 두 헤더를 include 한다. 나머지 함수는 하나하나 살펴보겠다.

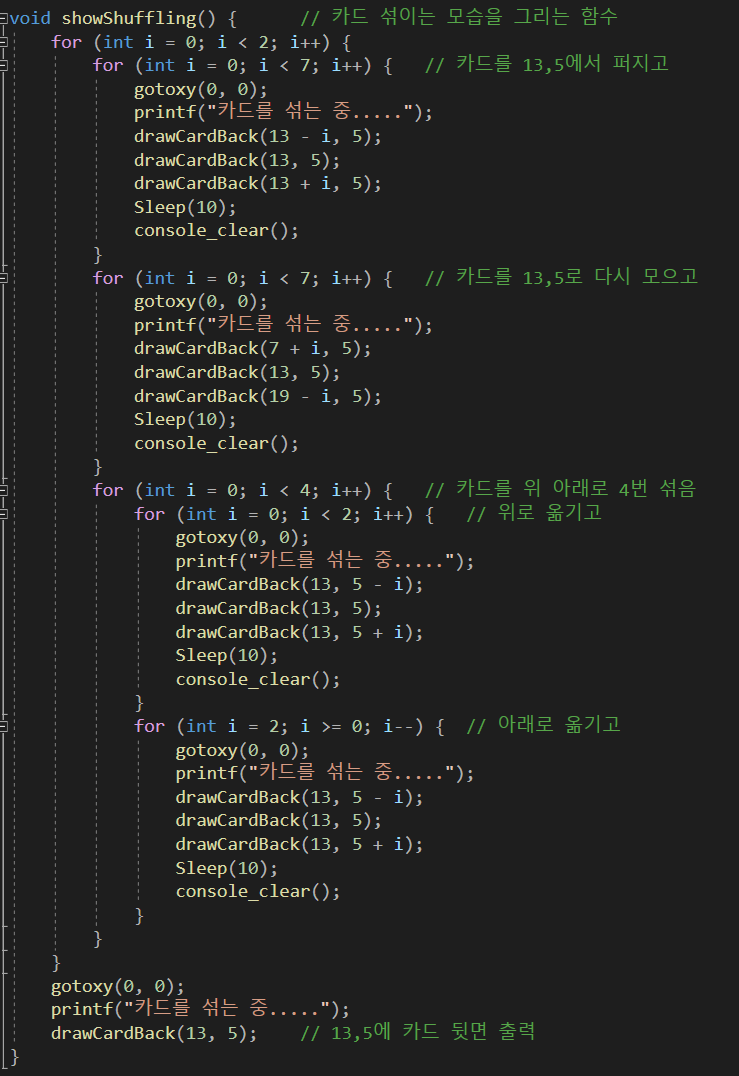
console.c이다.



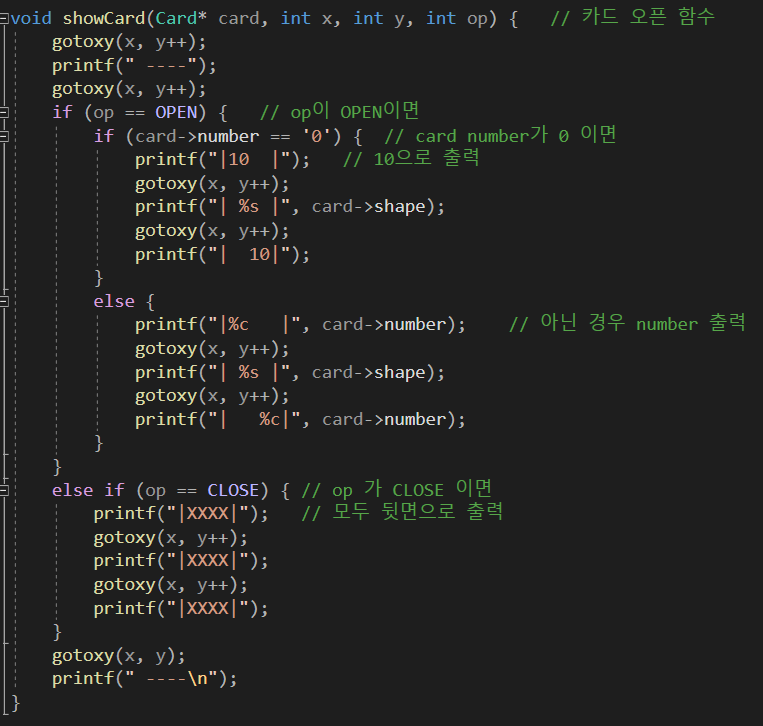
먼저 필요한 헤더를 include한다. gameStart() 함수는 첫 화면에 게임 시작을 알리는 문구를 출력하는 함수이다. 0,0으로 이동하여 문구를 출력한다.



카드 뒷면을 그리는 함수 drawCardBack()함수이다. x, y를 입력 받아 그 곳으로 gotoxy()를 이용하여 이동하며 카드를 그린다.



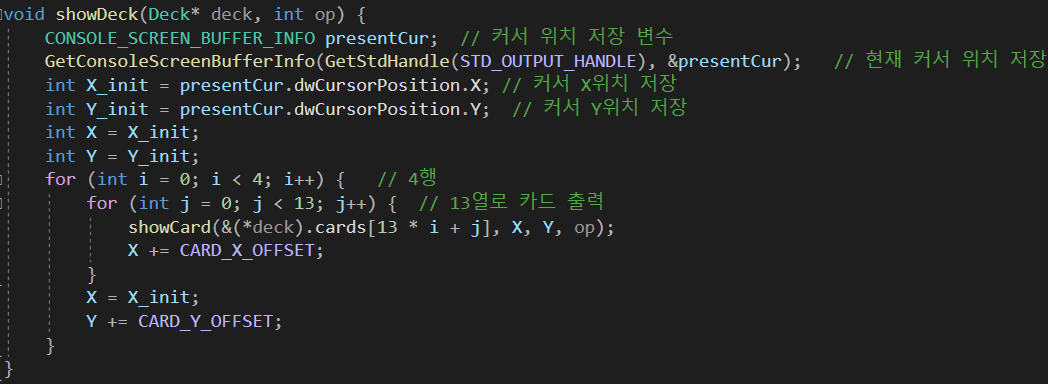
카드를 섞는 모습을 출력하는 함수 showShuffling이다. 총 2세트로 움직이는데, 먼저 13,5에 그려진 카드 뒷면을 기반으로 양 옆으로 카드 2개가 움직이고, 다시 모여서, 위 아래로 4번씩 움직이는 것이 한 세트이다. 이때, drawCardBack()함수를 사용하여, 양 옆으로 이동할 때는 x값을 순차적으로 바꾸고, 위 아래로 이동할 때는 y 값을 순차적으로 바꾸어 카드가 움직이는 것을 확인할 수 있다. 자세한 것은 동영상을 첨부하도록 하겠다.



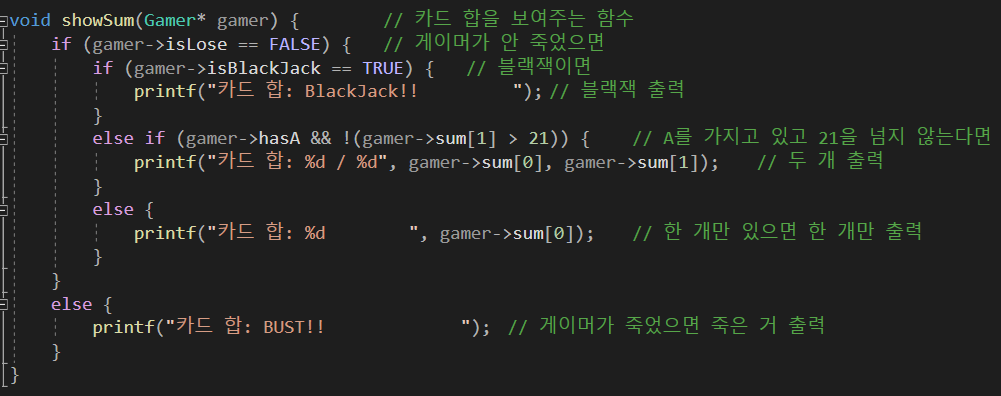
카드 한 장을 그리는 showCard() 함수이다. 입력 받은 x, y에 카드를 그리고 op의 종류에 따라 앞 면을 그릴 지 뒷면을 그릴 지 정하게 된다.

먼저 앞면인 경우 op == OPEN인데, 각 카드의 숫자와 shape을 카드 형식에 맞춰 출력한다. 카드 숫자가 0인 경우 10을 의미하기 때문에 그 부분은 따로 처리했다.

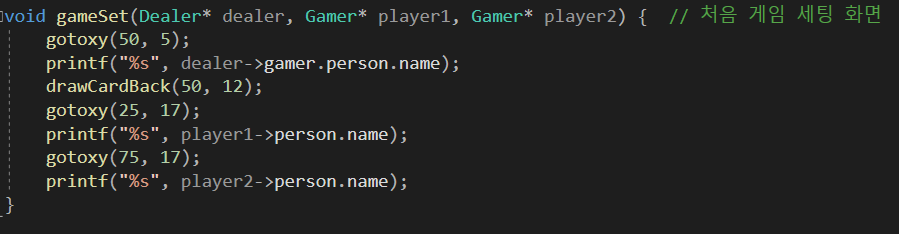
op == CLOSE인 경우는 뒷면이기 때문에 x를 이용하여 뒷면을 나타내었다.



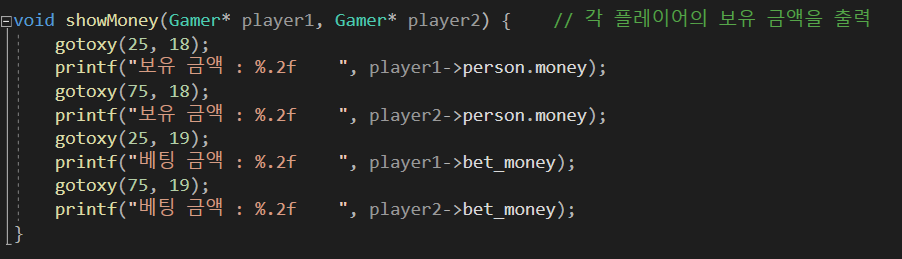
덱 전체를 보여주는 showDeck()함수이다. op에는 OPEN 혹은 CLOSE가 들어온다. 현재 커서를 저장하는 구조체를 가지고 커서 위치를 저장한다. 그리고 2중 for문을 돌면서 13열, 4행으로 카드를 그린다. 이때, source에서 정의한 CARD\_X\_OFFSET, CARD\_Y\_OFFSET을 이용하여 카드가 겹치지 않도록 그린다. 이 또한 자세한 것은 사진을 첨부하겠다.

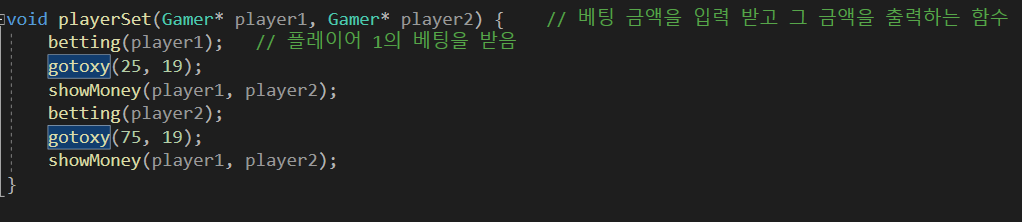


카드 합을 보여주는 함수 showSum()이다. 먼저 게이머가 isLose == FALSE 인 경우에, isBlackJack == TRUE 이면, 블랙잭을 출력, 아닌 경우 hasA == TRUE이고 sum[1]이 21보다 작은 경우 sum[0]과 sum[1]을 모두 출력하고, 아닌 경우 sum[0]만 출력한다. 만약 isLose == TRUE이면, BUST를 출력한다.

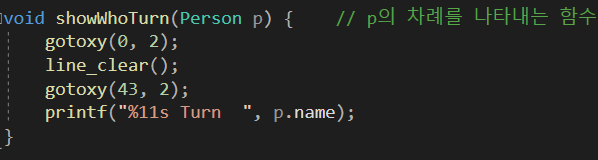


게임이 진행되는 화면을 출력하는 gameSet() 함수이다. 각각의 dealer, player1, player2의 위치에 맞게 그 이름을 출력한다.



보유 금액과 베팅 금액을 보여주는 함수 showMoney()이다. 정해진 위치에 각 player의 보유 금액과 베팅 금액을 보여준다. 

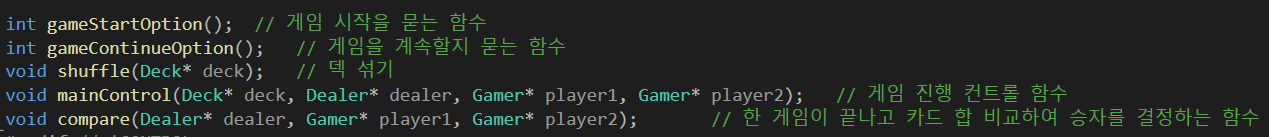
베팅 금액을 입력 받고 베팅 금액을 출력하는 playerSet() 함수이다. source.c에 정의된 betting 함수를 통해 베팅을 받고, showMoney() 를 호출하여 해당 금액과 보유 금액을 출력한다.



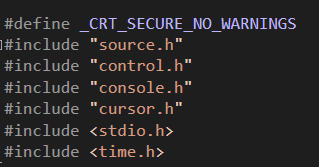
차례를 출력한 showWhoTurn함수이다. 들어온 person의 이름을 정해진 위치에 출력한다.

1. control.h && control.c

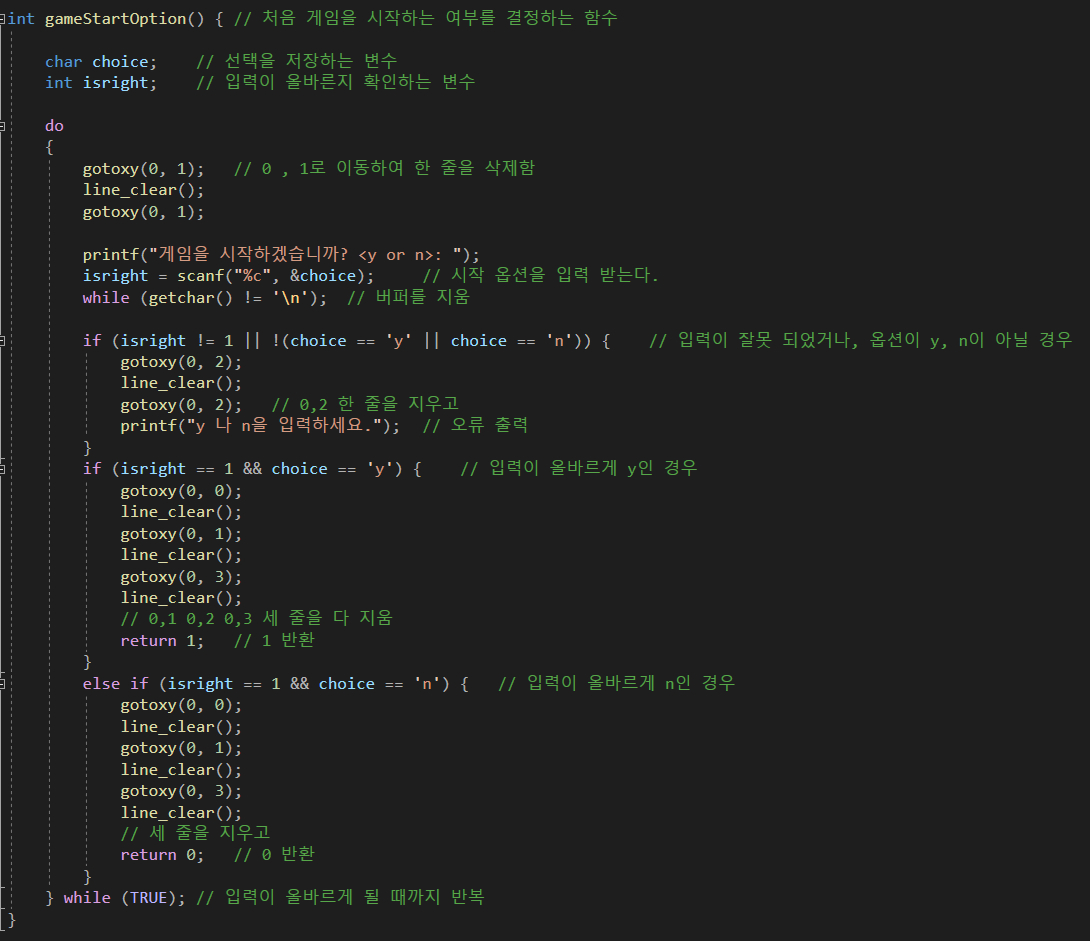
control 모듈은 게임의 전반적인 흐름을 모두 구현한 모듈이다. 앞서 구현한 모든 모듈을 사용한다.



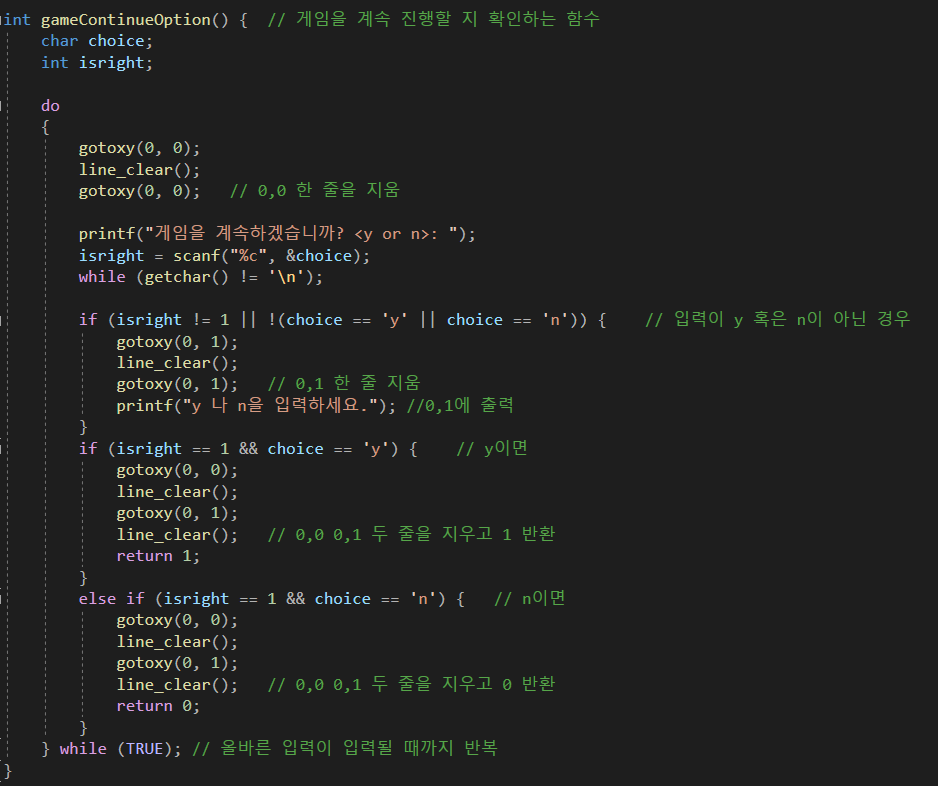
control.h에 정의된 함수들이다. 자세하게는 .c 파일에서 살펴보겠다.



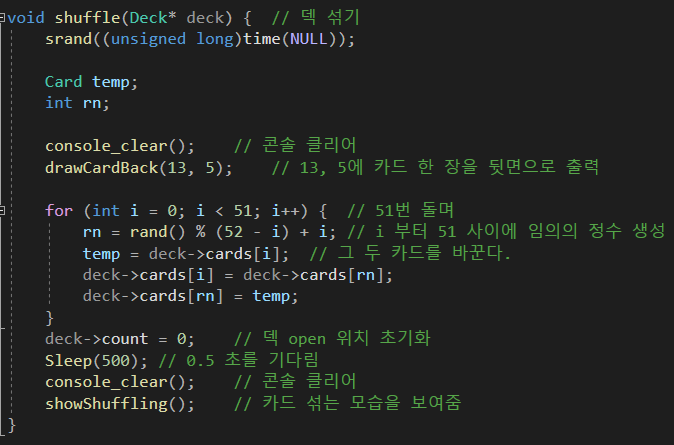
먼저 필요한 헤더를 include 한다.



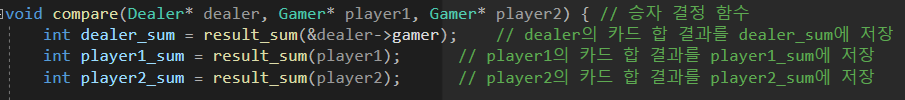
먼저 게임 시작 여부를 선택하는 gameStartOption()함수이다. 이 구조는 입력을 받는 구조이기 때문에 기존에 입력을 받던 코드와 동일하다.



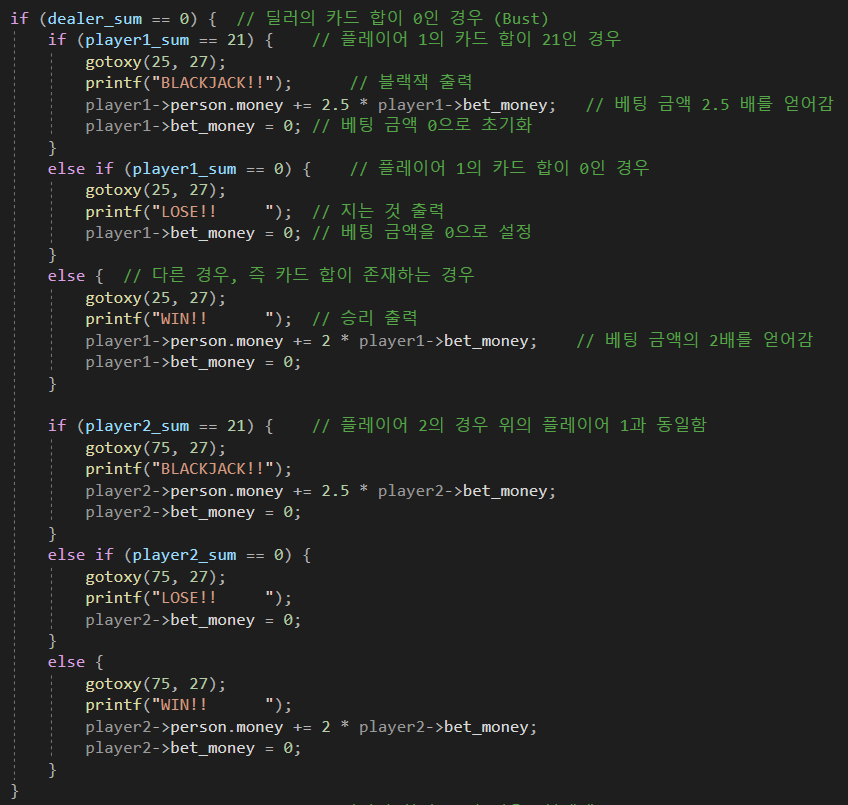
게임을 계속 진행할 지 확인하는 함수 gameContinueOption()이다. 이 또한 입력을 받는 함수이기 때문에 기존에 입력을 받는 코드와 동일하다.



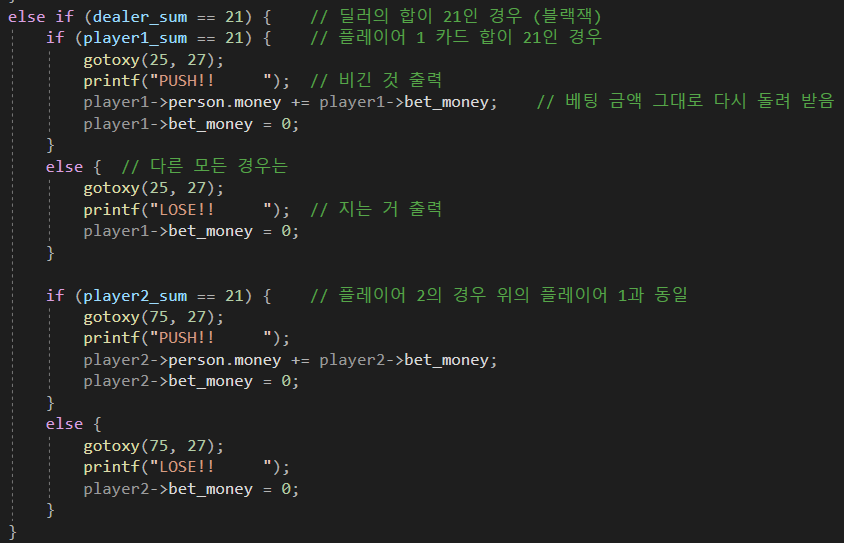
카드를 섞는 shuffle() 함수이다. 먼저 콘솔을 초기화하고, 13, 5에 카드 한 장을 뒷면으로 그린다. 그리고 51번 동안 for문을 실행하며 i ~ 51 임의의 정수를 생성하여 서로 카드를 바꾸어 순서를 바꾼다. 그리고 콘솔을 클리어하고, showShuffling()을 호출하여 섞이는 모습을 출력한다.



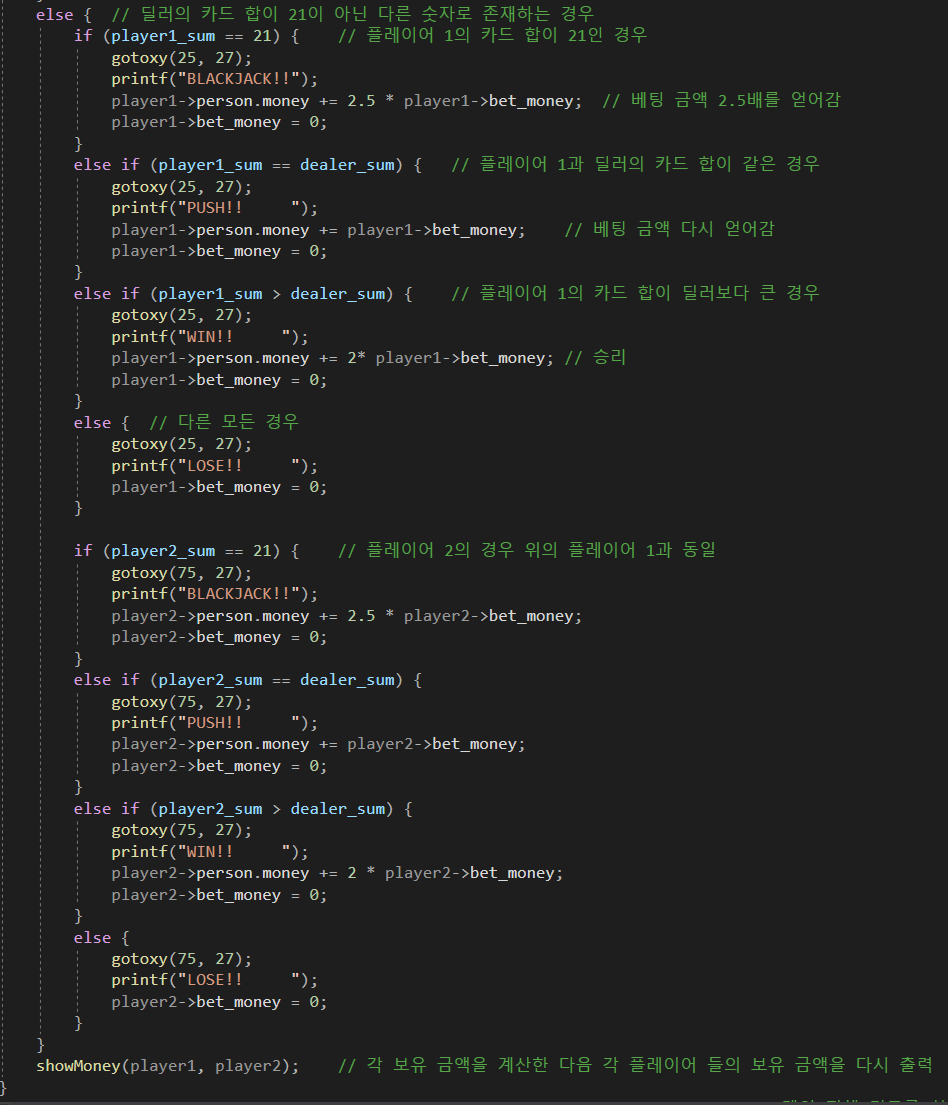
승자를 결정하는 함수 compare()이다. 이것은 길어서 나누어서 보겠다. 먼저 3인의 최종 카드 합 결과를 각각 dealer\_sum, player1\_sum, player2\_sum에 저장한다.



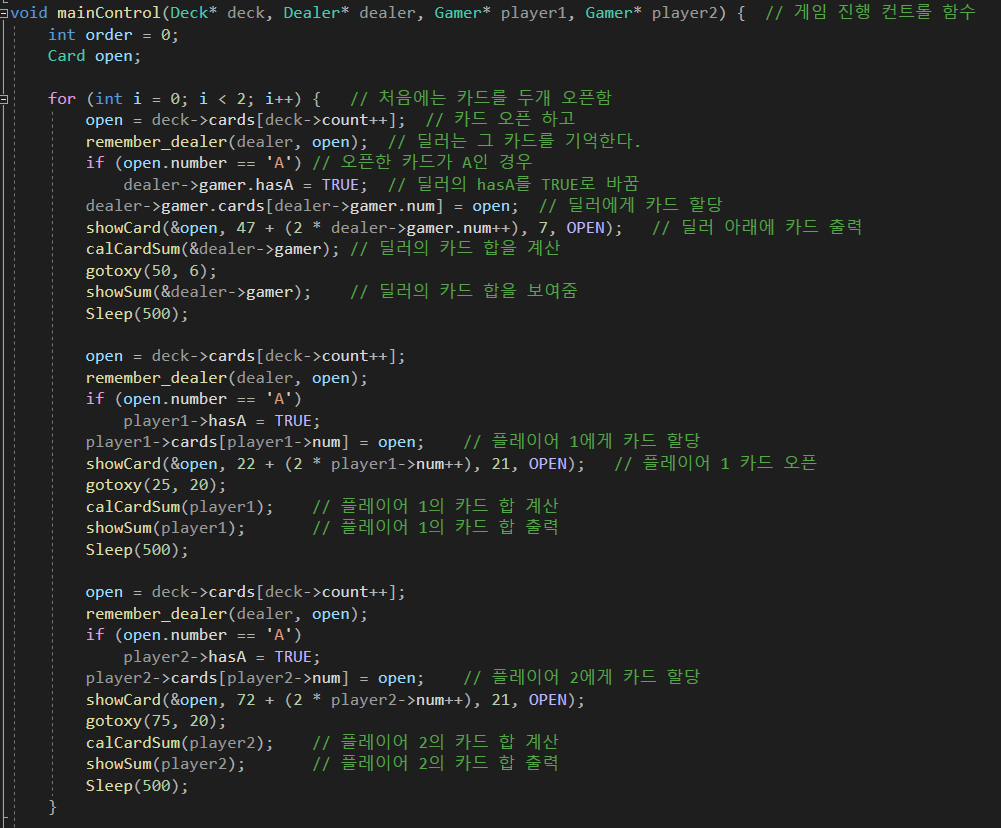
먼저 dealer\_sum == 0 인 경우, 즉, dealer 가 BUST인 경우이다. 이 경우 player1\_sum이 21인 경우 BLACKJACK 이므로, 베팅 금액의 2.5 배를 보유 금액에 추가하고 베팅 금액을 0으로 초기화한다. 만약 player1\_sum이 0 인 경우 player1도 BUST이기 때문에 베팅 금액만 초기화한다. 다른 경우는 어찌됐던 sum이 존재하는 경우이기 때문에 이긴 것으로 베팅 금액의 2배를 돌려받는다. player2의 경우 player1의 경우와 동일하게 흘러간다.



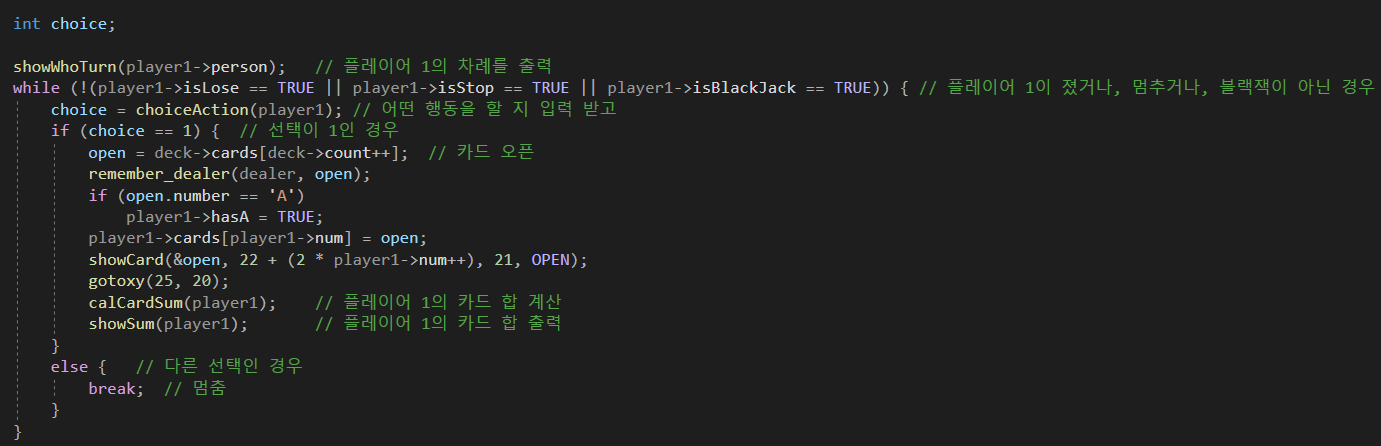
만약 딜러가 21, 즉 BLACKJACK인 경우, 두 플레이어 모두 sum이 21인 경우에만 비겨서 베팅 금액 그대로 돌려받고, 다른 경우 지는 경우로 돌려 받지 못한다.



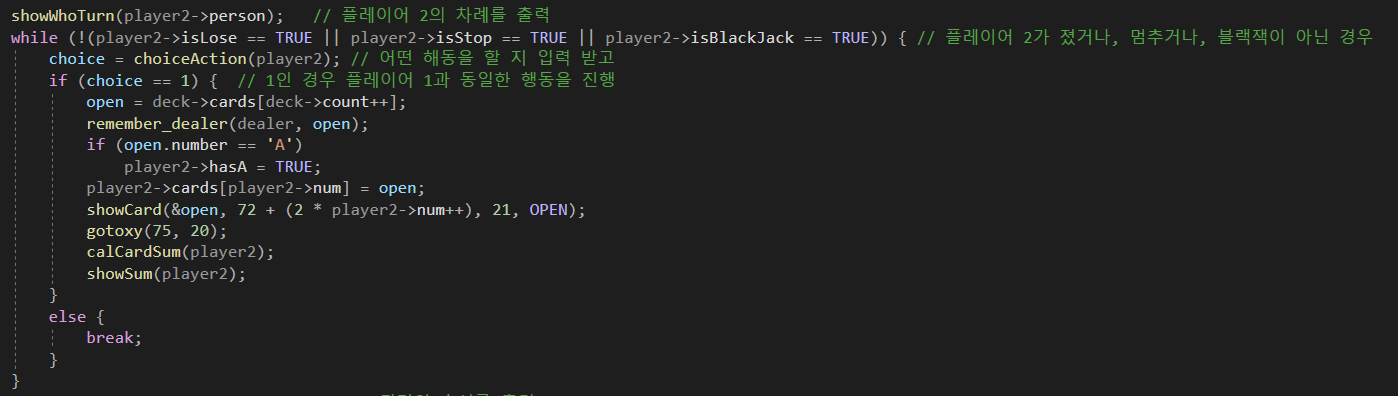
다른 경우는 딜러의 카드 합이 0 ~ 21 사이의 값으로 BUST는 되지 않고 BLACKJACK도 아닌 상황이다. 이 경우 플레이어의 카드 합이 21이면 블랙잭으로 베팅 금액의 2.5배를 받는다. 만약 딜러와 값이 같으면 비겨서 베팅 금액만 받고, 플레이어의 카드 합이 더 크면 이겨 2배를 돌려받는다. 다른 경우는 지는 경우로 아무 금액도 돌려받지 못한다. 모든 계산이 끝나면 showMoney()를 호출하여 최종 금액을 출력한다.



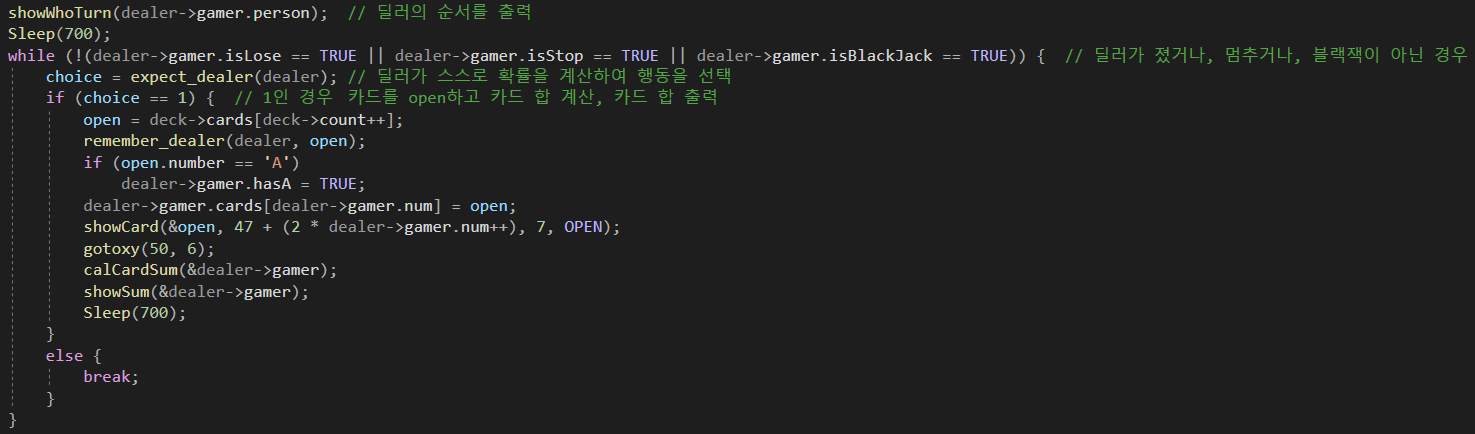
게임이 흘러가는 진행 사항을 구현한 mainControl 함수이다. 이 경우도 나눠서 살펴보겠다. 먼저 이미 정의된 deck, dealer, player1, player2를 가져온다. 그리고 먼저 for문을 통해 3인 모두에게 카드 2장씩 분배한다. 분배하는 과정은 카드를 open하여 임시로 저장하고, open할 때마다 dealer는 카드 넘버를 기억한다. 그리고 오픈한 card가 ‘A’라면 해당하는 순서의 hasA를 TRUE로 바꾼다. 그리고 해당 차례의 cards에 추가한다. 이후, 카드를 출력하고, calCardSum()으로 카드 합을 계산, showSum()으로 카드 합을 출력한다. 이 과정을 각 사람 당 2번씩 진행한다.



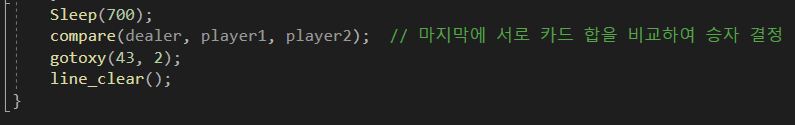
이후 player1부터 카드를 open할 지, stop할 지 정한다. player1의 차례를 출력하고, 아직playe1이 지거나, 멈추거나, 블랙잭이 아닌 경우 choiceAction을 통해 카드를 open할 지, stop할 지 정한다. 선택이 1인 경우 카드를 오픈하고, 카드 합을 계산하고, 출력한다. 만약 0인 경우는 while을 벗어나서 카드 오픈을 멈춘다.



player2의 경우에도 player1과 동일하게 진행한다.



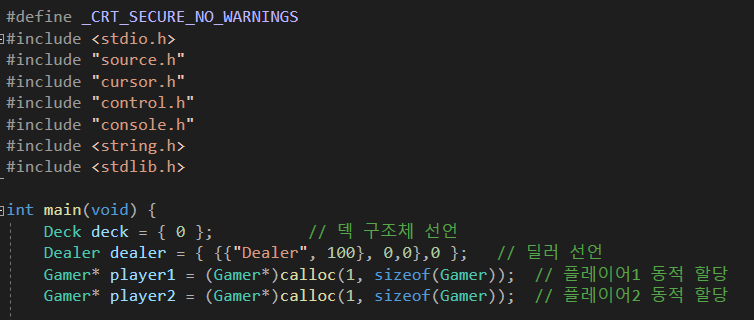
딜러의 경우 choiceAction이 아닌 expect\_dealer를 통해 카드를 open할 지 stop할 지 결정하게 된다. 나머지는 위와 동일하다.



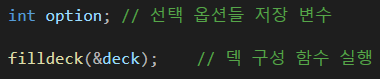
마지막으로 결과를 비교하여 승자를 결정하며 함수를 종료한다.

1. BlackJack.c

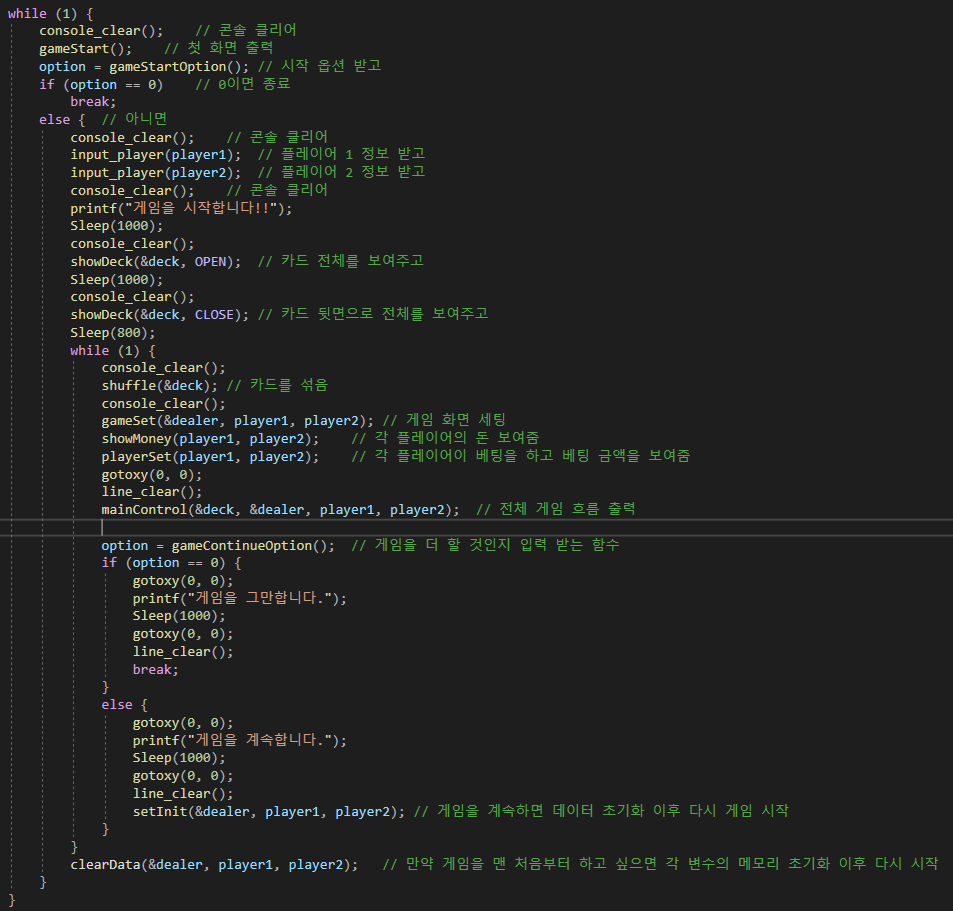
이제 앞서 정의한 모듈을 모두 사용하여 하나의 게임을 실행하는 실행파일 BlackJack.c 파일이다.

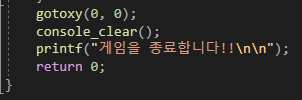


먼저 앞서 정의한 모듈들을 include한다. main에서는 먼저 Deck과 Dealer, player1, player2를 할당해준다.



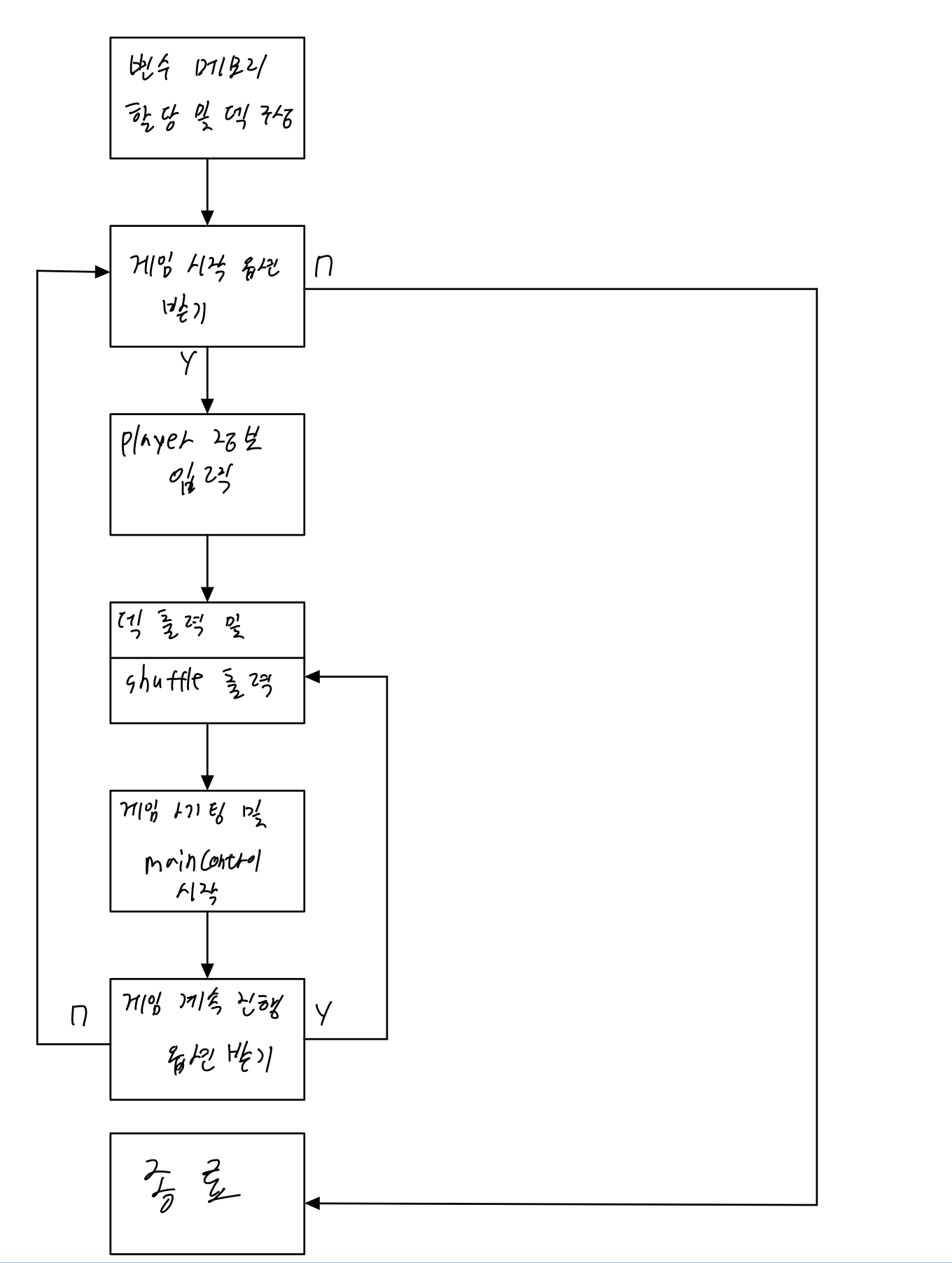
그 다음 덱을 카드로 채운다.



게임이 돌아가는 총 상황이다. 차근차근 살펴보자. 먼저 콘솔을 클리어하고, 첫 화면을 출력한다. 그 다음 시작을 할 것인지 옵션을 받아 0이면 전체 while문을 빠져나가 종료, 아니면 입력으로 들어간다. input\_player()함수를 통해 player1, player2의 정보를 입력 받고, 게임을 시작한다는 문구를 출력한다. 먼저 showDeck()을 처음에는 OPEN, 다음에는 CLOSE로 출력하여 카드가 올바르게 셋팅 됐는지를 확인한다. 새로운 while문으로 들어가, shuffle을 통해, 카드를 섞고 섞는 모습을 보여준다. 이후 gameSet을 통해 게임 화면을 세팅, showMoney()를 통해 기본 금액 출력, playerSet()을 통해 베팅을 받는다. 베팅을 받고 나면 mainControl을 실행하여 게임을 진행한다. 끝난 이후, gameContinueOption()을 통해 계속할 지 입력 받고, 계속할 경우는 게임 관련 데이터를 초기화하는 setInit를 실행하고 shuffle 전의 while로 돌아가고, 그만 할 경우, 전체 데이터를 초기화 하는 clearData를 호출하고 밖의 while문으로 돌아가게 된다. 

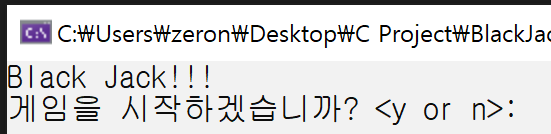
다 끝나고 게임 종료를 출력하고 프로그램이 종료한다.

이 과정을 그림으로 살펴보면,



이런 식으로 동작한다.

이제 출력을 살펴보겠다.

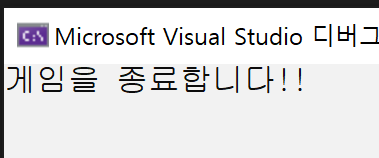


첫 화면이다.

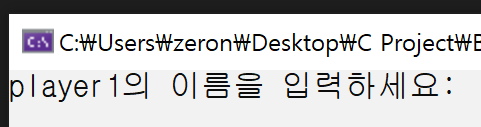
 이 부분이 실행되는 중이다.



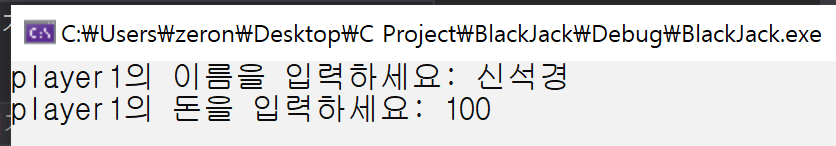
잘못 입력한 경우 이렇게 출력된다.



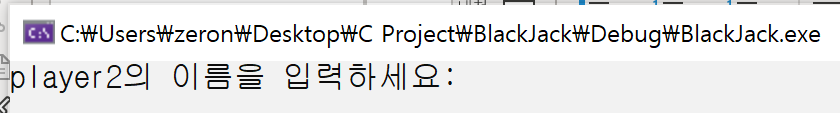
n을 입력하고 게임이 종료된 상황이다.



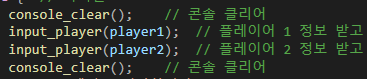
y를 입력하면 player1의 이름을 받는 곳으로 돌아온다.



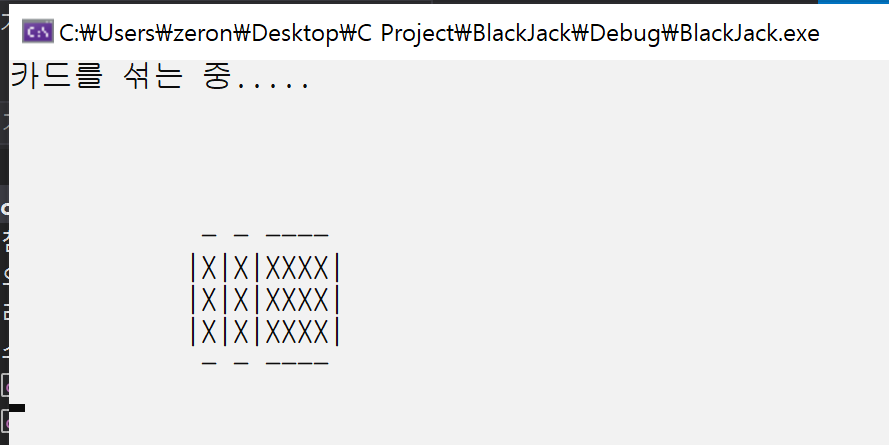
이런 식으로 정보를 입력한다.



player2도 입력 받는다.



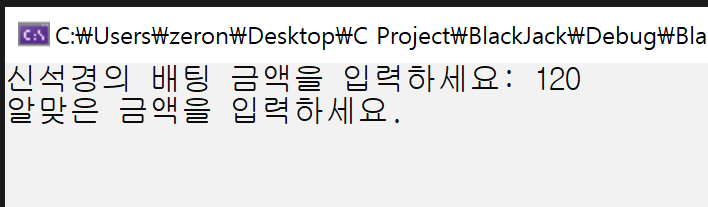
이 부분이 실행된 것이다.



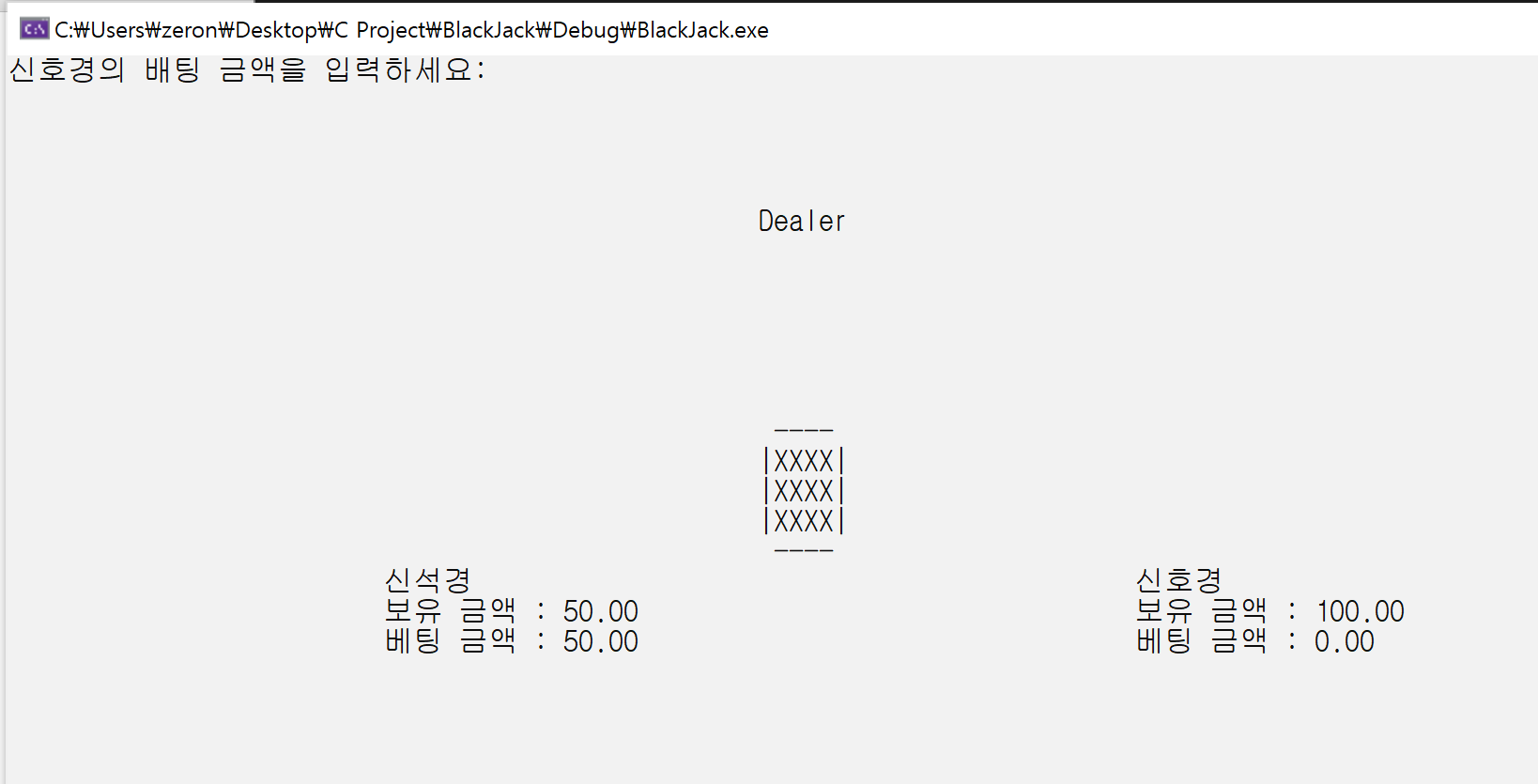
그 다음 카드를 섞는 부분이다. 이 부분은 동적으로 움직이니 동영상으로 대체하겠다.



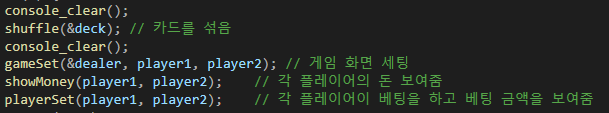
이제 game에 들어온 상황이다. 각각 위치에 맞게 출력하고, player1부터 베팅 금액을 입력 받는다.



이때에도 보유 금액을 넘어가는 금액을 입력하면 오류를 출력한다.



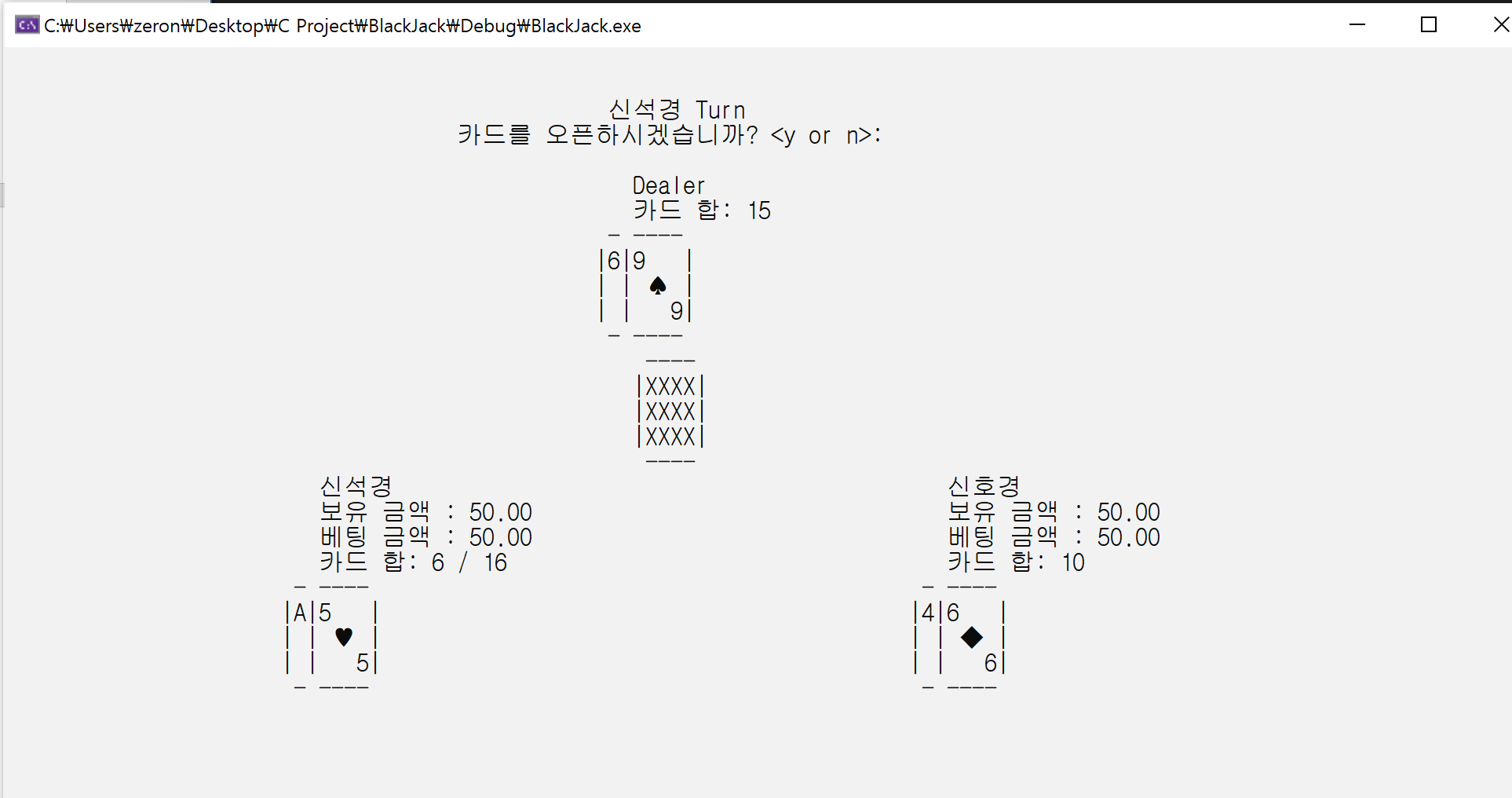
player2도 베팅 금액을 입력 받는다. 둘 다 50씩 입력을 받겠다.



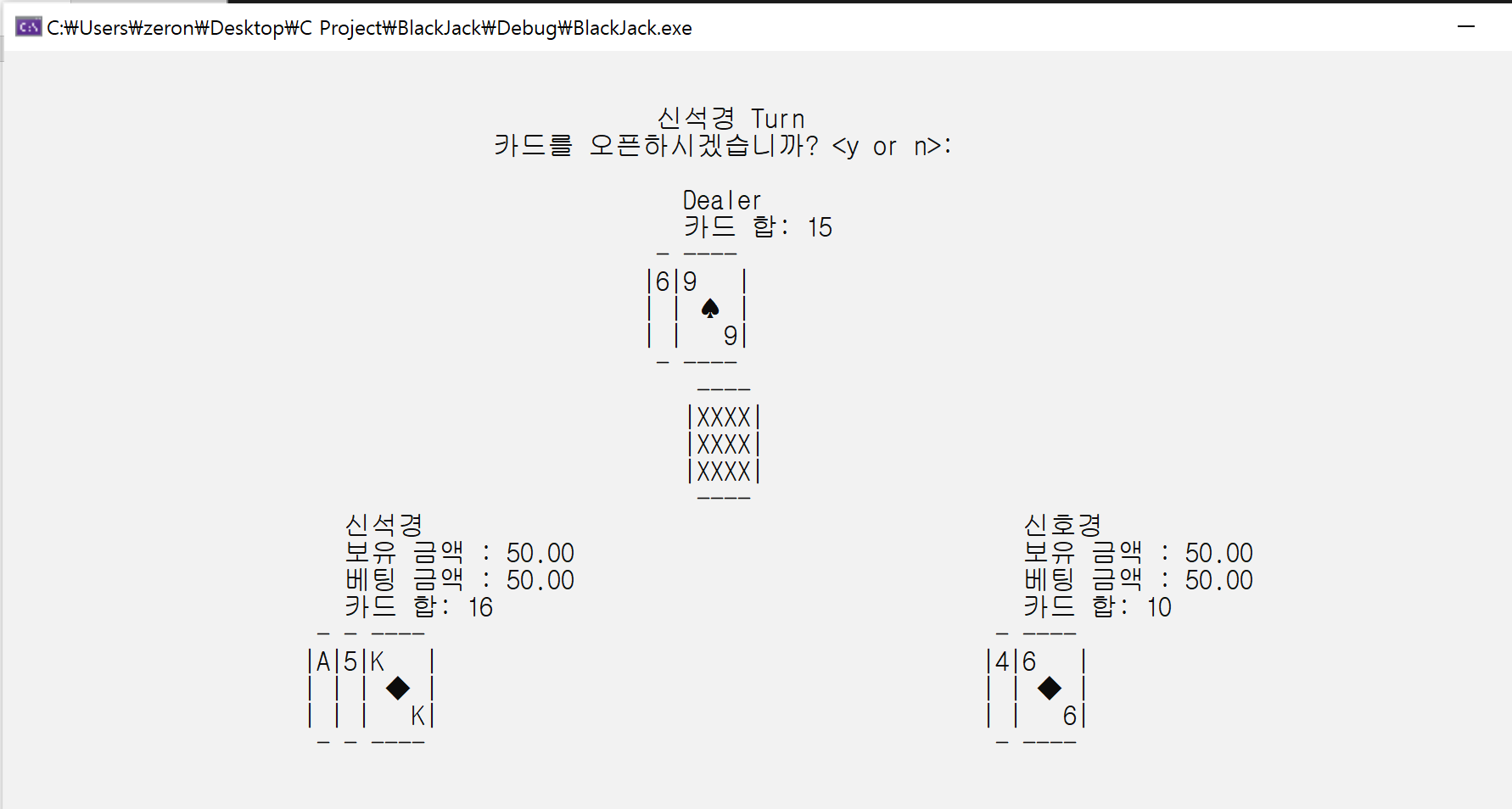
여기까지의 실행 결과이다.



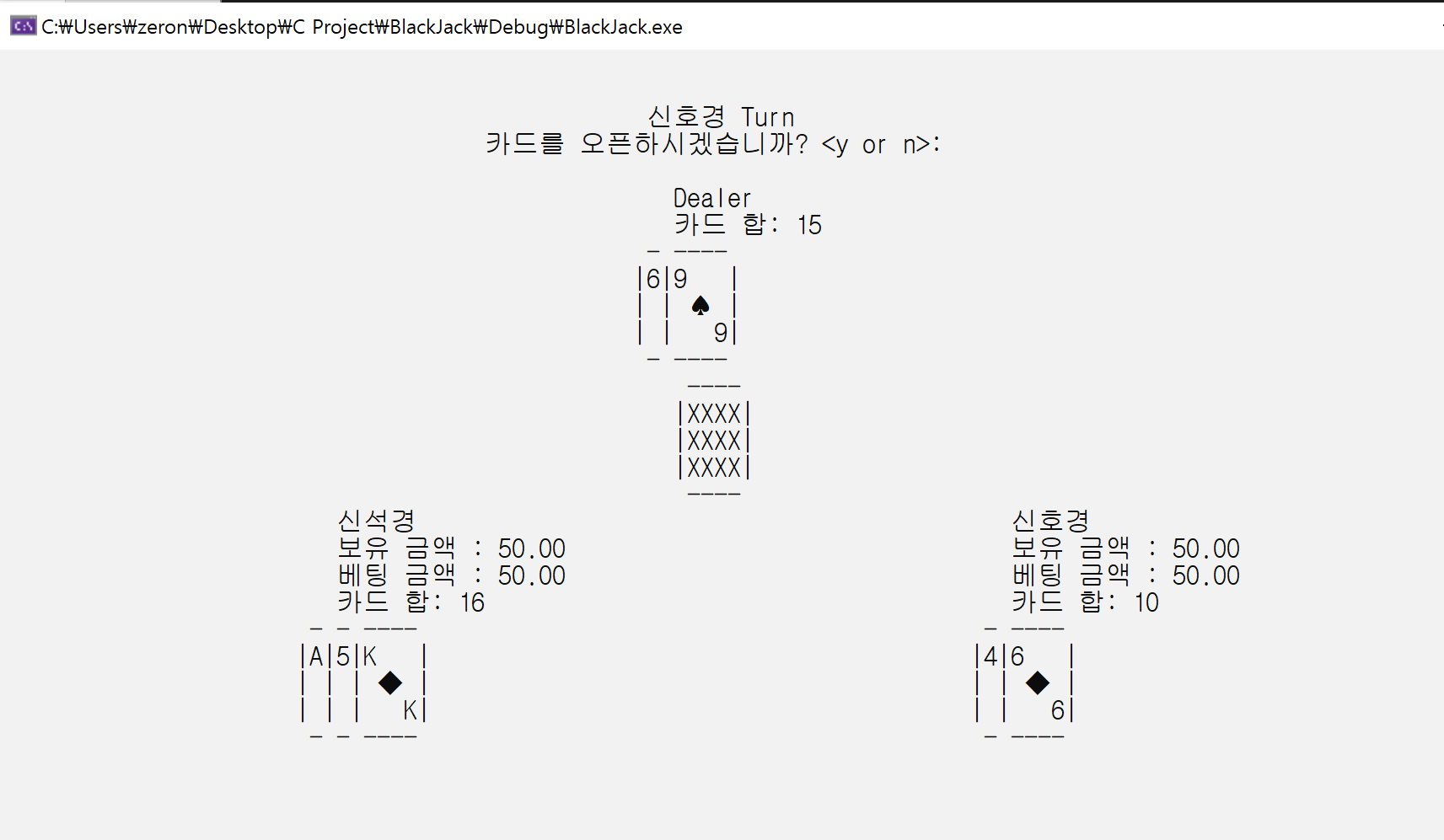
이제 mainControl이 돌아가는 모습을 살펴보자.



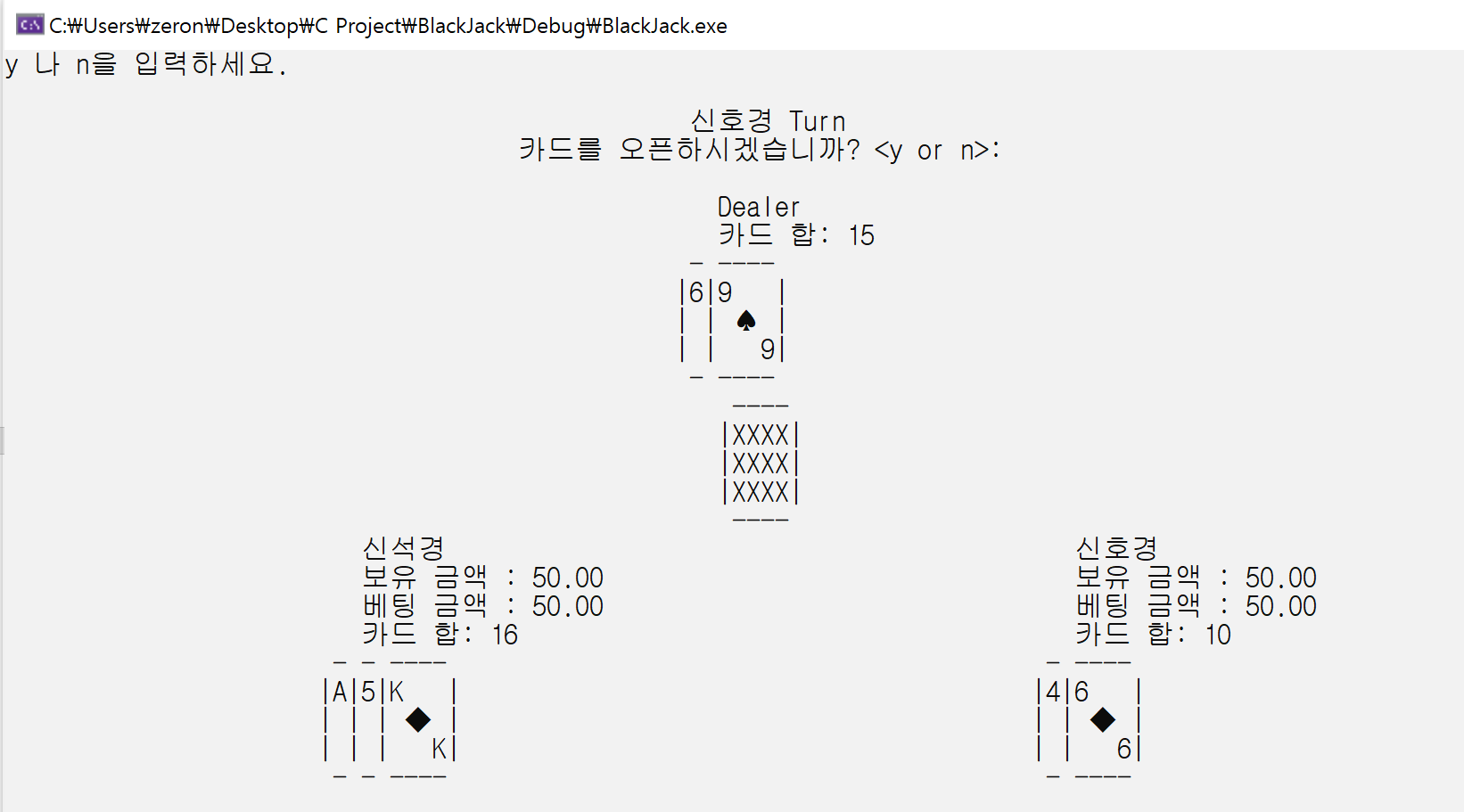
먼저 각각 2장의 카드를 open한다. 그리고 그에 대한 카드 합도 출력한다. 중앙을 보면 “신석경”이 카드를 오픈할 것인지 묻는 문구가 있다. 여기서 y를 입력하면,



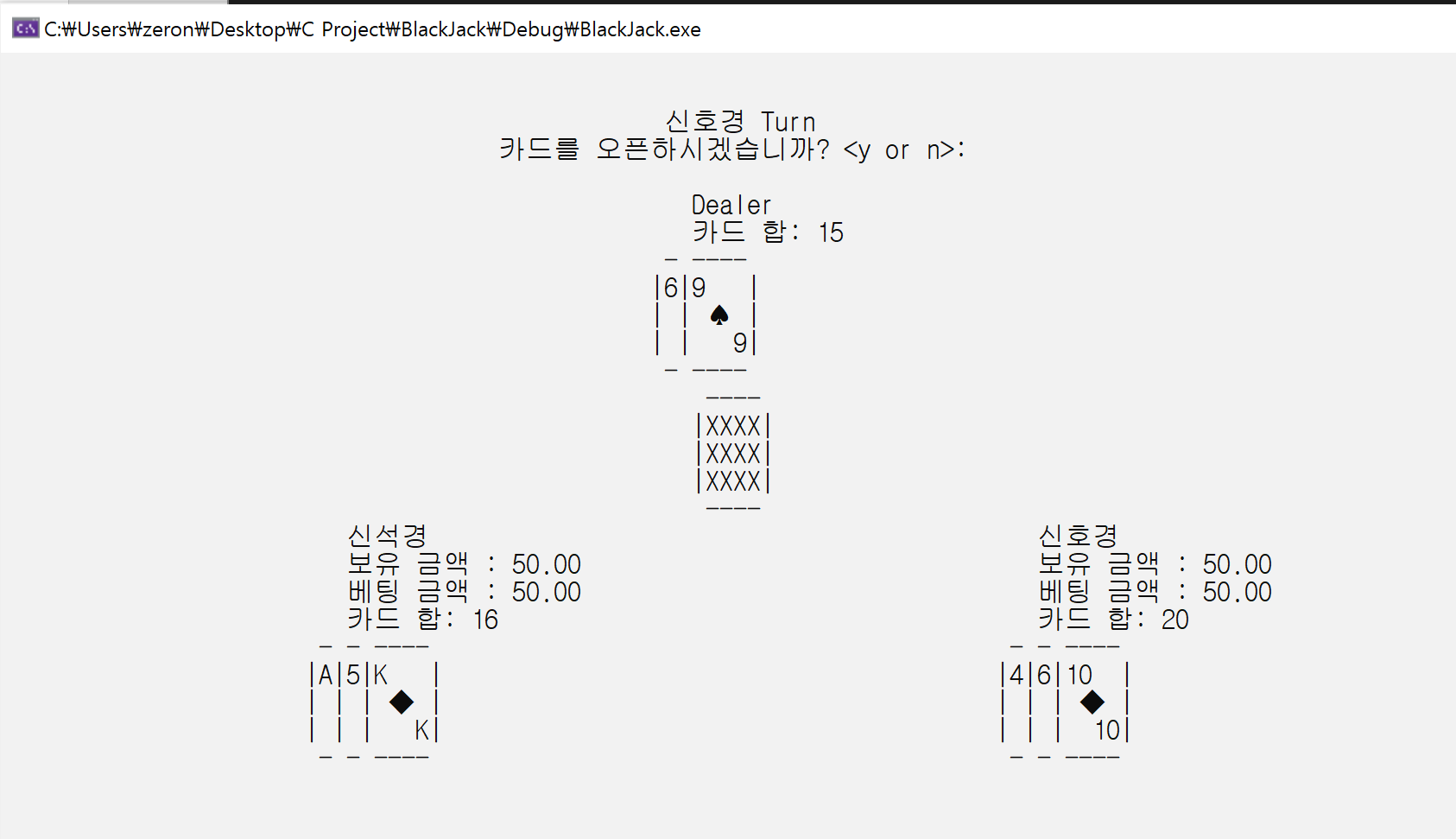
k가 나와 카드합이 16이 되고, 다시 “신석경”에게 묻고 있다. 이제 n을 누르겠다.



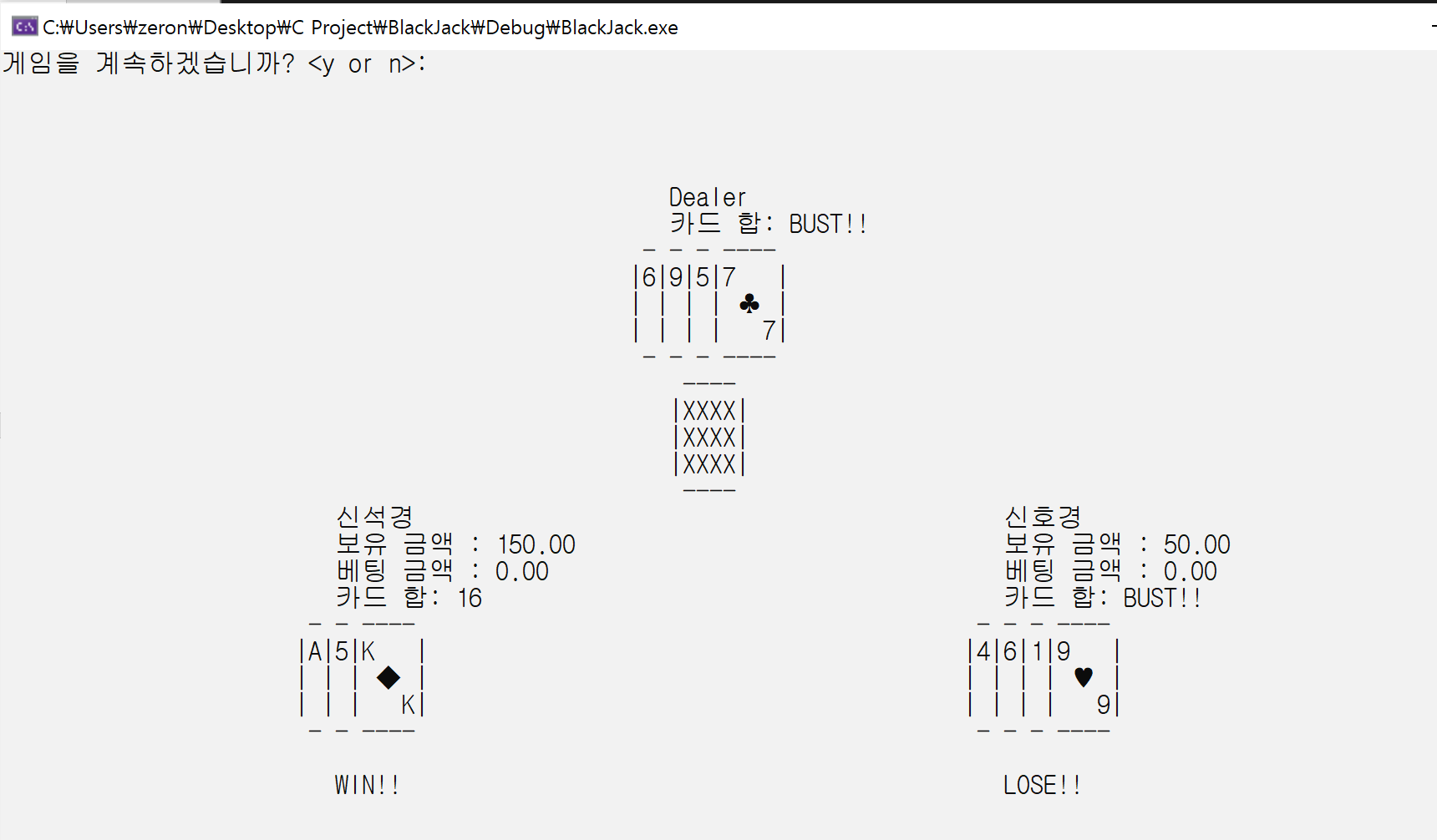
그러면 “신호경”의 차례로 넘어간 것을 확인할 수 있다. 여기서 y, n이 아닌 다른 입력을 넣으면,



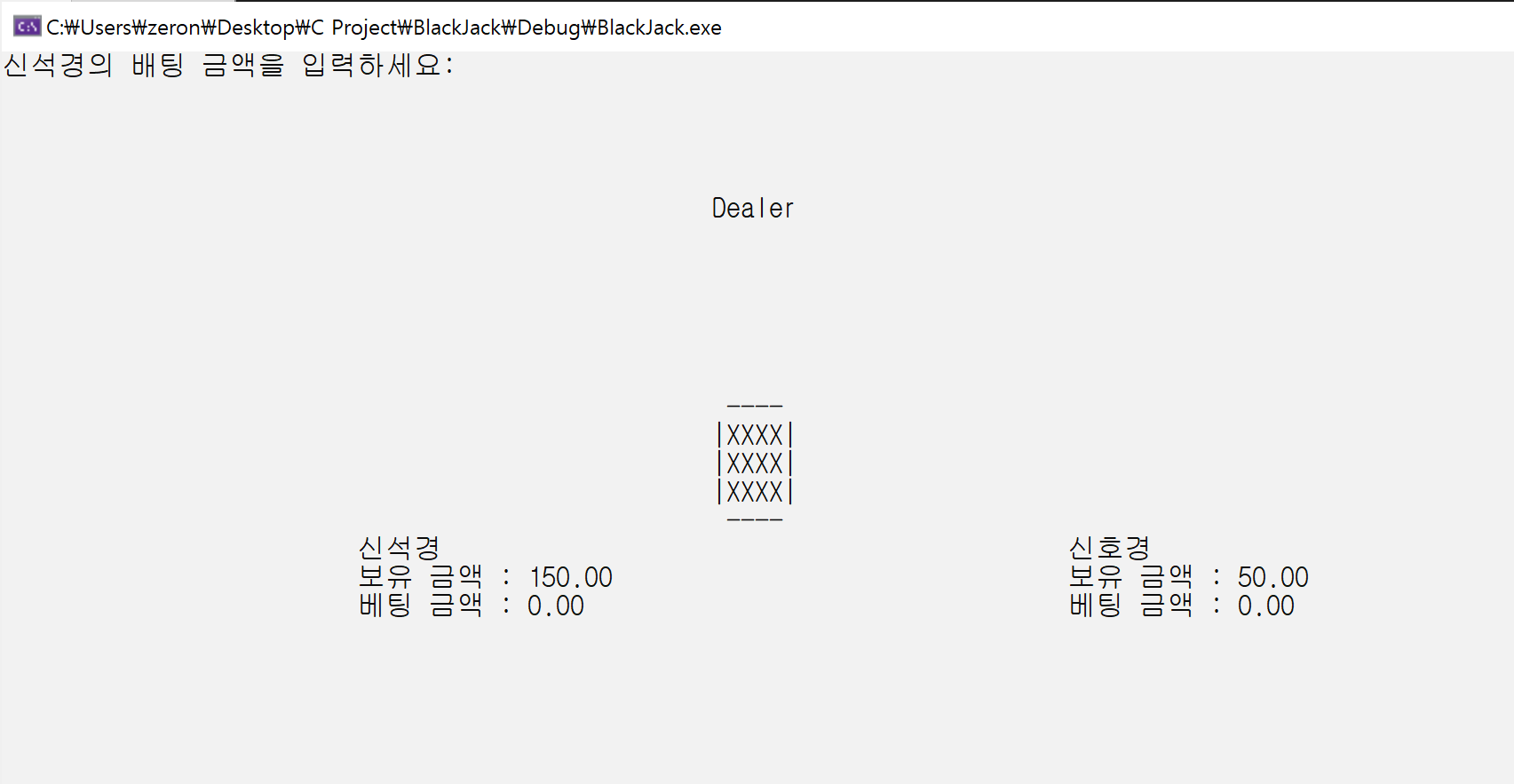
왼쪽 위에 오류가 뜬다. 카드를 open하기 위해 y를 입력하겠다.



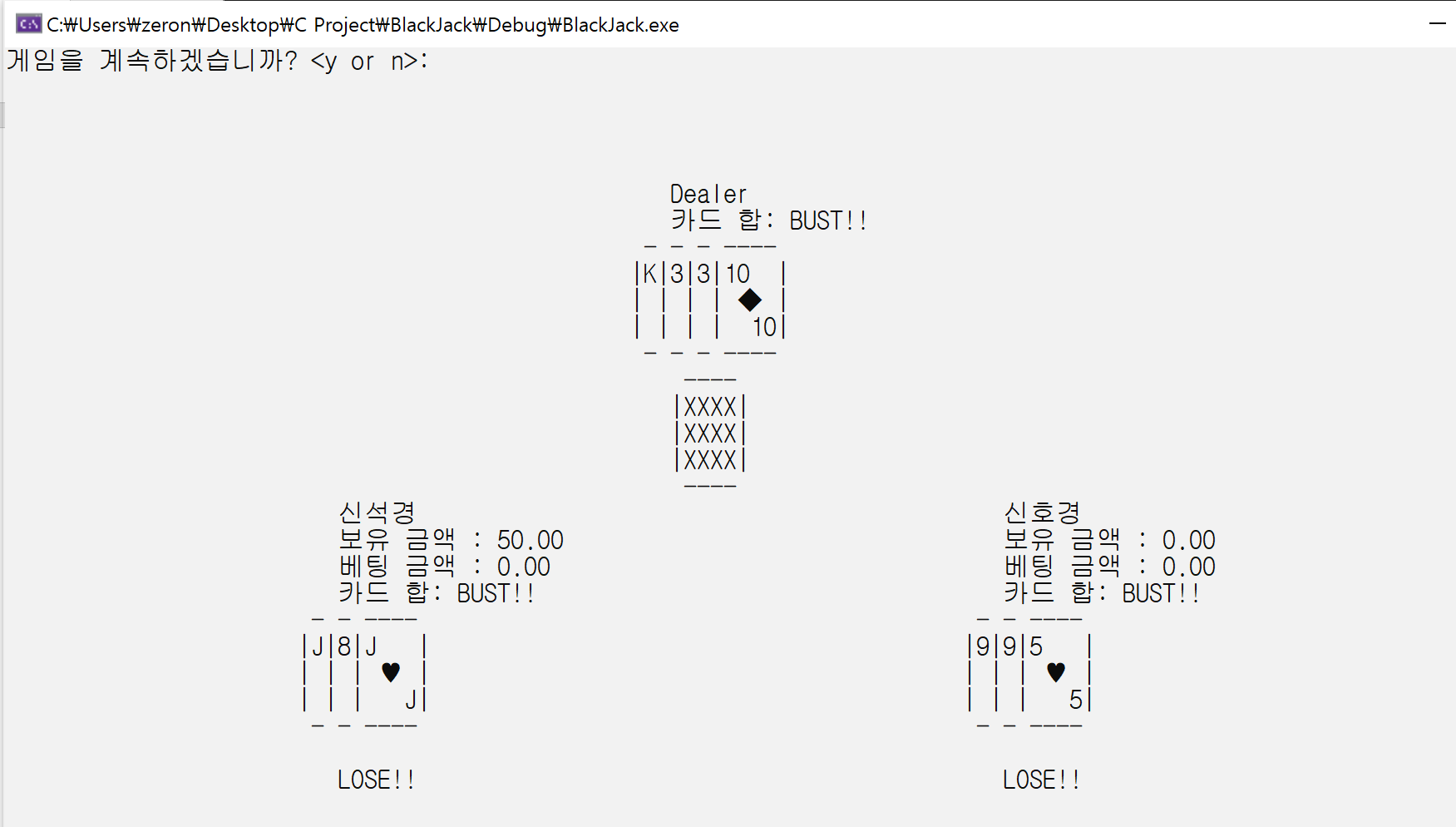
10이 open되어 20이 되었다. 한 번 더 open하겠다.



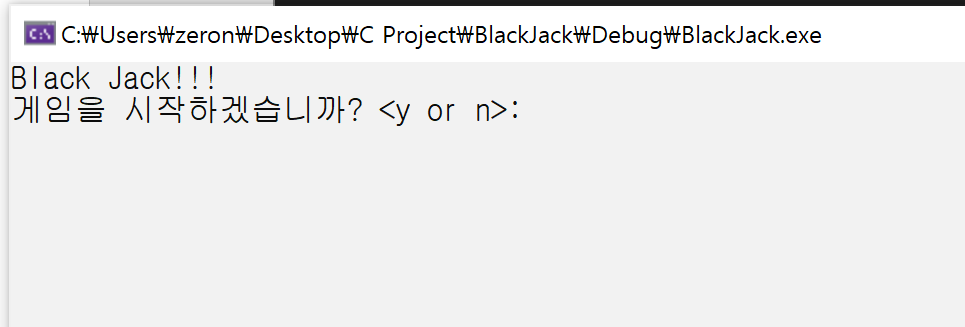
“신호경”은 BUST되었다. “신호경” 차례가 끝난 후 딜러는 스스로 카드를 open하였는데, BUST되었다. 따라서 “신석경”은 이겨서 베팅 금액의 2배를 얻어 150원이 되었고, “신호경”은 져서 베팅 금액을 날리게 되었다. 여기서 왼쪽 위를 살펴보면 계속하겠냐는 문구가 나오는데, y를 누르고 다시 shuffle로 돌아가게 된다.



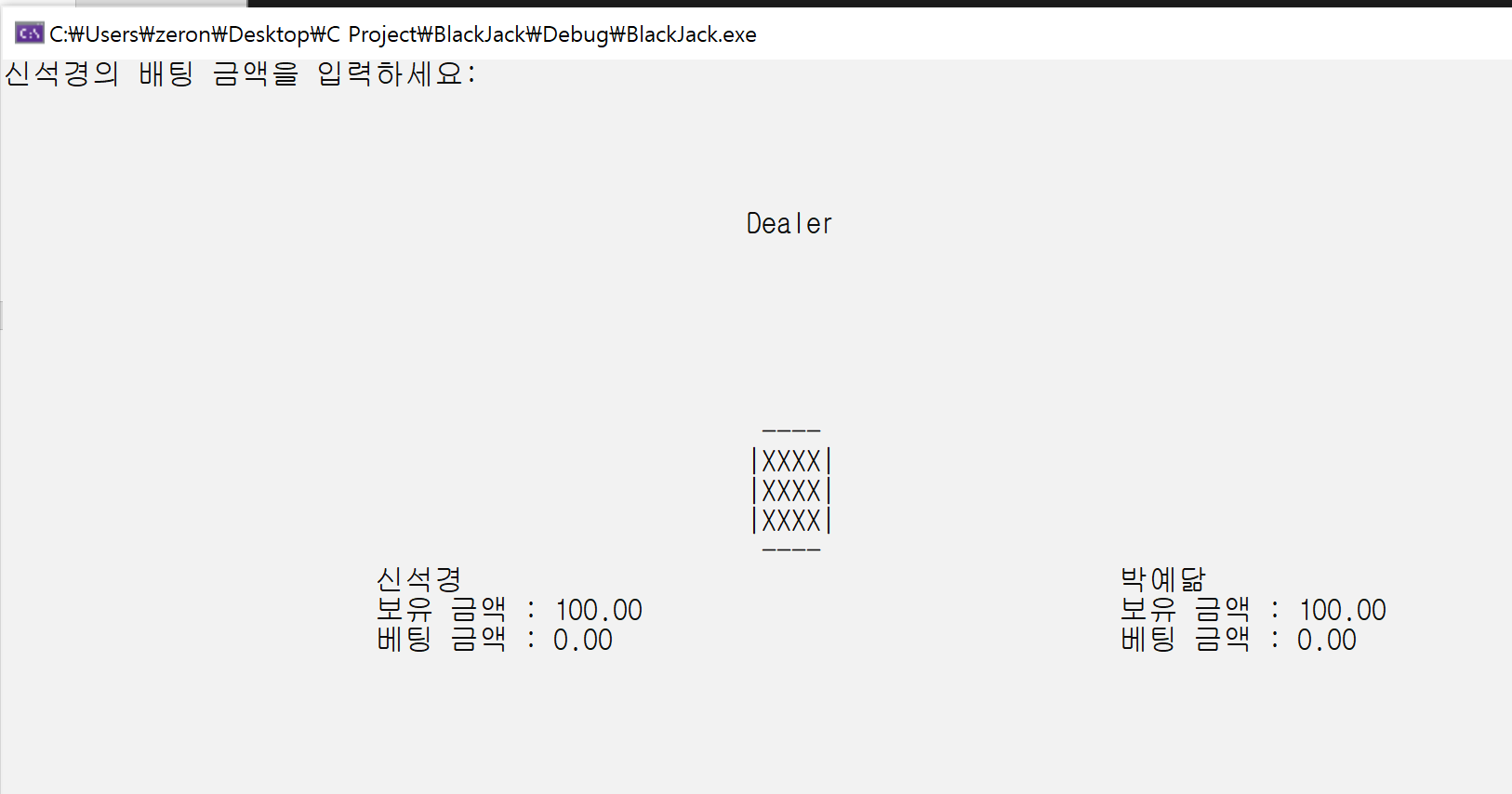
보면, 보유 금액이 유지되는 것을 확인할 수 있다. 이제 다시 베팅을 받고 게임을 진행할 수 있다.

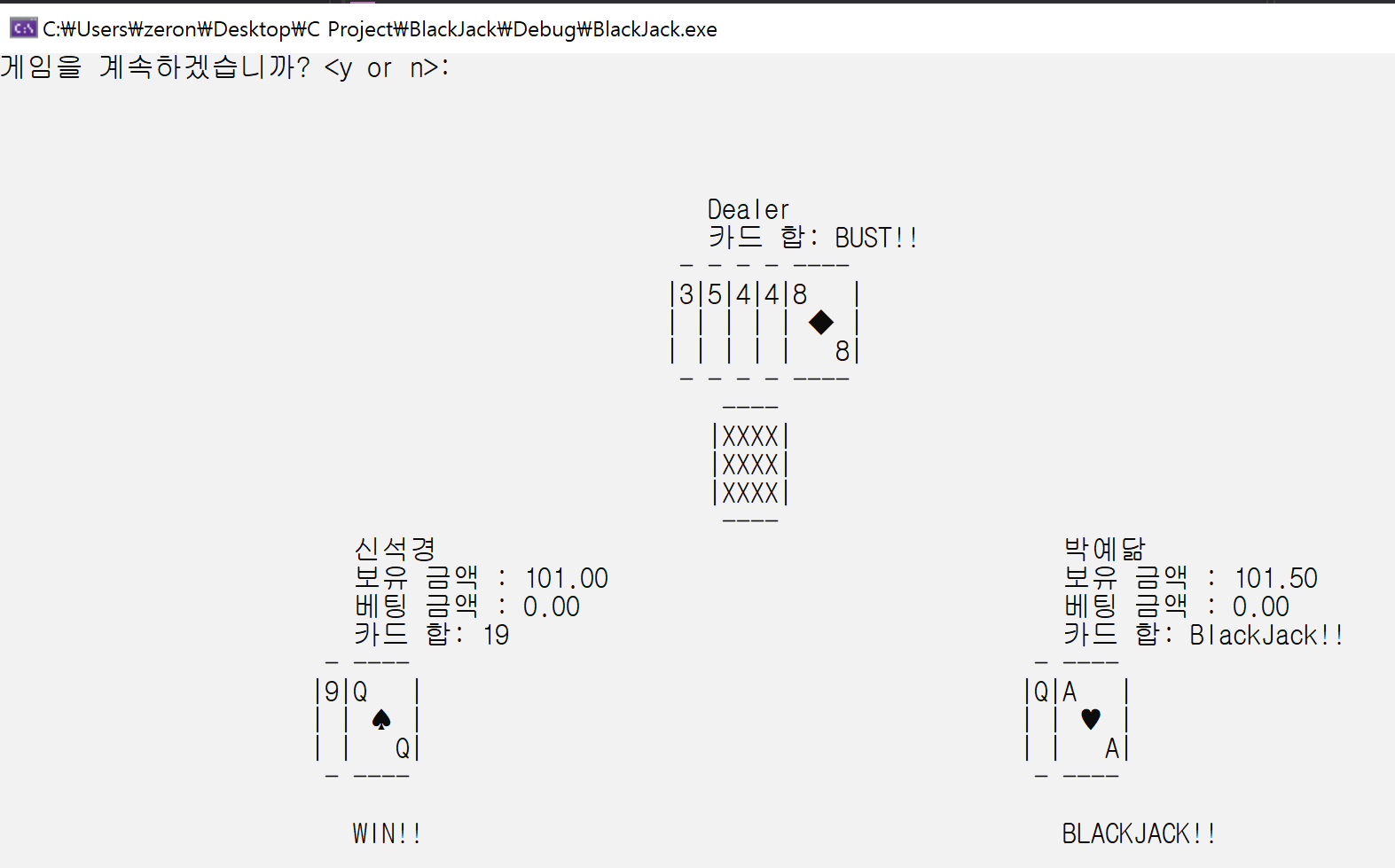


모두 BUST 한 경우가 나왔다. 여기서 n을 입력하여 처음으로 다시 돌아가면,

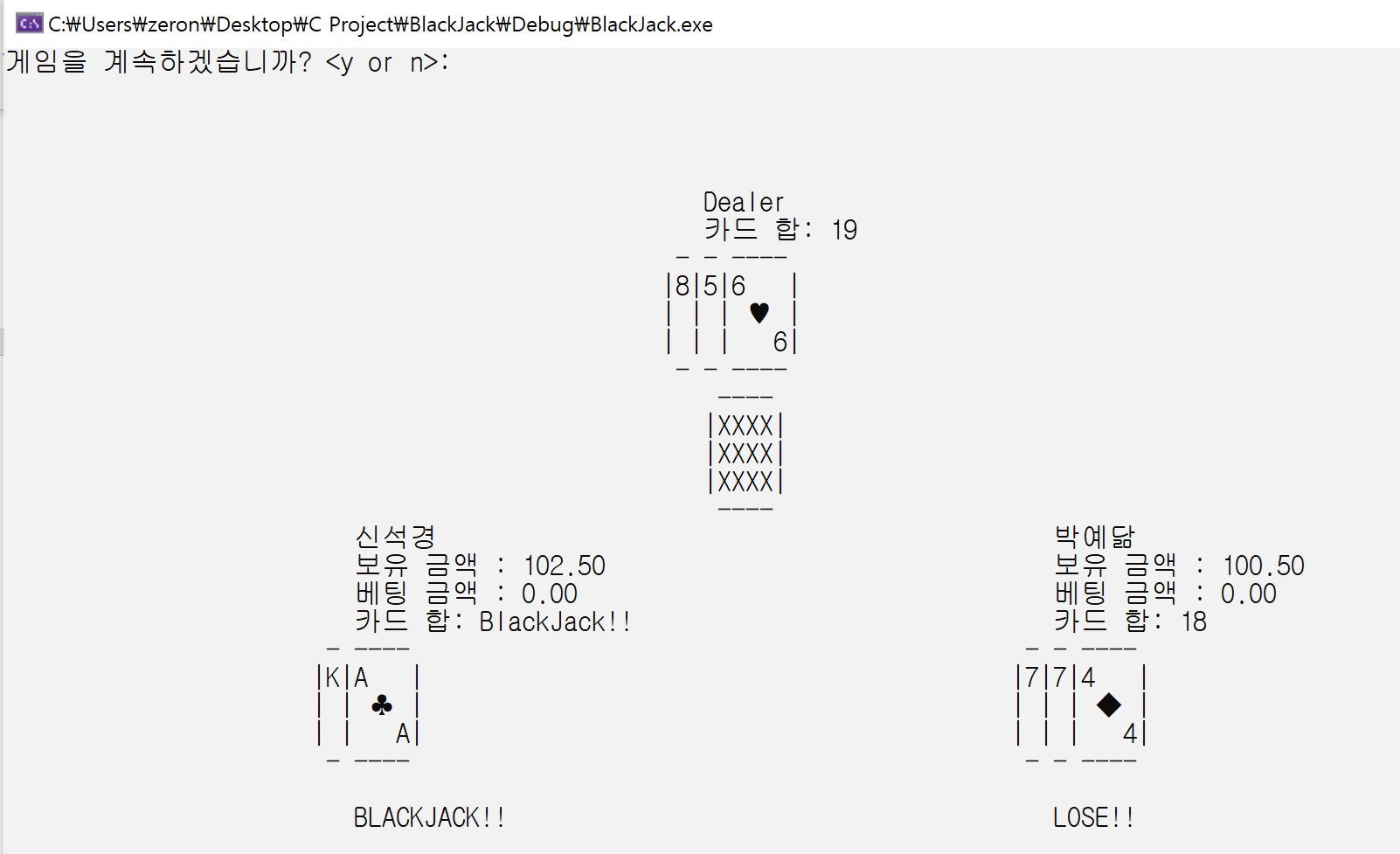
다시 이 화면으로 돌아오게 된다. 이제 여러가지 결과를 살펴보겠다.

다시 y를 눌러 새로운 플레이어를 입력 받은 상황이다.





1원을 베팅하고 게임을 진행한 결과, “박예닮”이 BLACKJACK이 나와 2.5배인 2.5원을 돌려받은 결과이다.



이 경우에는 “신석경”은 블랙잭으로 2.5원을 획득하였고, “박예닮”은 카드 합이 18이었지만, 딜러가 19에서 open을 멈춰 진 경우이다.

지금까지 블랙잭 프로젝트를 설명하였다. 부족한 부분을 영상으로 살펴보겠다.