과제 2

현재 시각 계산하기

빙고 게임

20232907 정현승

1. 문제 1: 현재 시각 계산하기

time.h에 정의된 time 함수는 매개변수로 NULL을 넘겨주면 그 결과로 세계 표준 시각 기준 1970년 1월 1일 0시 0분 0초 이후로 현재까지 몇 초가 지났는지를 알려준다(그 값을 반환한다). 이를 분석해 오늘 날짜와 시각을 표시하고, 이를 1초 단위로 반복한다. 단 반환된 값을 분석하는 기능을 사용할 수 없고 직접 만들어야 한다.

2. 문제 1 - time함수의 반환값 분석 방안

우선 년도, 월, 일, 시, 분, 초를 저장할 수 있는 구조체를 만들고, 이 구조체를 자료형으로 하는, 분석 결과를 저장할 변수와 우리나라의 시간대를 저장할 상수 각각 1개씩을 선언한다. 코드에는 분석 부분을 함수로 빼서, 분석 결과를 저장할 변수는 함수 내에서 선언하고 시간대를 저장할 상수는 매개변수로 받아왔다. 또한 확장성을 고려해 구조체에 요일도 저장할 수 있게 했지만, 이 문제에서는 계산은 하나 활용하지 않는다. 먼저 time 함수의 반환 값(코드에서 timeint)을 60으로 나눈 나머지가 현재 초(second)이므로, 시간대에 저장된 초를 더한 뒤 전체 반환 값을 60으로 나눈 나머지를 분석 결과의 초(second)에 저장한다. (time 함수의 반환 값에는 윤초가 고려되지 않으므로, 윤초 보정은 하지 않아도 된다.) 그리고 다음 계산의 편의를 위해 전체 반환 값을 60으로 나누고 이를 timeint에 저장한다. 이 과정을 한 번 반복해 현재 분을 얻고, 60이 아닌 24로 나눈 나머지를 이용해 또 한 번 더 반복하면서 현재 시를 구한다. (분과 시의 경우 시간대에 저장된 상수가 영향을 미쳐, 미리 저장된 시간대에 맞는 시간으로 계산될 것이다. 또한 시의 경우, 오전과 오후를 구분하지 않고 24시간 형식을 사용하며, 이에 따라 오후 1시는 13시, 오후 2시는 14시, ..., 오후 11시는 23시가 된다.)

현재 시를 얻고 timeint에 이를 24로 나눈 값을 저장하고 나면, timeint에는 1970년 1월 1일 이후 며칠이 지났는지를 저장하고 있게 된다. 시간대에 일과 월이 설정되어 있다면 이것까지 계산한 뒤(코드가 정상 작동하는지 확인할 때 활용할 수 있다), 우선 윤년을 고려하지 않고, 즉 1년을 365일로 보고 기준 년도(1970년)로부터 몇 년이 지났는지를 계산한다. 이는 timeint을 365로 나눈 나머지로 구할 수 있으며, 구한 후 timeint에 이를 365로 나눈 나머지를 저장하면 이 변수에는 년초부터 며칠이 지났는지만 남게 된다. 이후 앞에서 무시한 윤년 보정을 위해 기준 년도부터 (기준 년도 + 계산된 년도) 1년 전까지의 기간 사이 윤년의 수를 계산해 날짜(timeint)를 하루씩 뺀다. 이때 날짜(timeint)가 음수가 되면 1년을 빼고 timeint에 1년 만큼의 날짜를 보충해 준다(보통은 timeint에 365일을 더하지만, 이번 년도가 윤년이면 1년의 366번째 날이 존재하게 되므로 년도를 빼면서 이번에 윤년 체크한 년도와 (기준 년도 + 계산된 년도)가 같아지게 되면 366일을 더한다. 이는 이후 월과 일을 계산하는 과정에서 추가 처리를 해 줄 필요가 없어지는 부가효과도 존재한다.) 이제 계산된 년도에 기준 년도를 더해주면(코드에서는 시간대를 저장하는 상수에 가본값이 저장되어 있다.) 년도 계산은 끝났고, 몇 월 몇 일인지만 계산하면 된다. (기준 년도부터 1년씩 년

도를 더하면서 윤년이면 366일, 그 외에는 365일을 빼는 방법도 존재하나 이는 뺄셈을 반복하게 되어 시간이 더 소요될 것이라 예상해 사용하지 않았다.)

월 계산도 년도 계산과 같이 1달을 30일로 계산하고 이후 1달이 30일이 아닌 달에 대해 보정하는 방법을 사용하였다. 다만 윤년 보정과 달리 월 보정은 1달이 며칠인지가 불규칙하기 때문에, switch 문을 사용하였고, 다른 달과 달리 1달이 30일이 안 되는 2월을 제외한 내부 계산은 별도 함수로 뺐다. 여기서 이미 특정한 달이 되었으면 그 달 이후의 보정은 필요가 없으니, 그 이후의 코드를 건너뛰는 목적으로만 switch 문을 사용하였다. (switch 문에서 break; 명령어를 사용하면 그 이후부터 switch문이 끝날 때까지의 명령문은 실행되지 않는다는 점을 이용하였다.) month_calculater함수는 일과 월, 1달이 31일인 월 2개(이번에 계산해야 할 월과 다음에 계산해야 할 월)를 매개변수로 받아(이때 일과 월은 그 값을 직접 바꾸어야 하는 관계로 그 주솟값을 받는다), 일(day)을 하루 빼고 일(day)가 음수가 되면 1달을 뺀 뒤 일(day)에 1달만큼의 날짜를 보충해 준다(년도와 마찬가지로 보통은 30일을 더하나, 1달을 빼서 계산된 월과 이번에 계산해야 할 월이 같아지는 경우, 31일을 더한다. 이렇지 않은 경우 30일과 31일 모두 30일로 계산되기 때문이다.) 이후 월이 다음에 계산해야 하는 월 이후가 아니면 switch 문 break를 위해 1(true)을 반환하고, 아니면 0(false)을 반환한다. (이 break 판정마저 반복되는 관계로, 쓸 일 없었던 반환 값을 이용해 판정까지 함수 내에 넣었다.) 2월 보정의 경우, 보정 전 먼저 1월 계산 후 월 값이 2월이면서 날짜가 29일 이내(윤년이 아니면 28일 이내)이면 2월 보정이 필요 없으므로 보정 작업을 종료한다. (이 경우는 그냥 2월이기 때문이다.) 이후 윤년이면 1일, 아니면 2일을 더한 후, 날짜가 30일을 넘어가면 1달을 올려주고 30일을 뺀다. 마지막으로 계산 후 3월이면 3월 보정이 필요 없기에 보정 작업을 종료한다. 예를 들어 1월 보정 직후 윤년이 아니고 월에는 2, 일에는 29가 들어가 있다면 보정 전 보정 작업 종료 여부를 판정할 때 월은 2이지만 29일 이내가 아니므로 보정을 계속하게 된다. (일에는 29가 들어가 있지만 이는 일을 1이 아닌 0부터 세서 그런 것으로, 일에 0이 들어가 있으면 실제로는 1일, 일에 29가 들어가 있으면 실제로는 30일이다. 따라서 29일 이내가 아니다.) 윤년이 아니므로 일에는 2를 더해 31이 되고(실제로는 32일), 30일을 넘겼으므로 30일을 빼고 1달을 더한다. 이 과정 이후에는 월에 3, 일에 1이 저장되어 있을 것이다. 마지막으로 3월이므로 보정작업을 종료하게 된다. 이 예시의 경우 계산된 날짜는 3월 2일이 된다.

위의 월 보정이 끝나면 지금까지 일(day)을 0부터 셌으니, 일(day)에 1을 더해주고, 이를 분석 결과에 저장하면 분석이 끝났다. 마지막으로 이렇게 구한 분석값을 반환한다.

main 함수에서는 이 분석 결과를 1초 단위로 반복해 화면에 출력해야 하나 Sleep(1000)을 사용하는 경우 분석 시간을 셀 수 없으므로 이를 사용할 수 없다. 따라서 time(NULL)이 1초마다 바뀌는 점을 이용했다. 이전 분석 시에 함수로부터 반환받은 값을 계속 저장하고 있다가, 함수의 반환값과 저장한 값이 일치하지 않으면 저장한 값을 새로운 반환 값으로 업데이트하고, 이를 날짜, 시간을 분석하는 함수에 매개변수로 넣어 분석 결과를 반환받는다. 그리고 이 분석값을 화면에 출력한다. 별도의 종료 조건이 없으므로 이를 무한 반복한다. 단 이 경우 첫 번째 출력과 두 번째 출력은 1초보다 짧은 시간 간격을 두고 일어난다. 프로그램의 동작 시간을 구하는 함수를 사용하

면 첫 번째 출력과 두 번째 출력도 1초보다 짧은 시간 내에 일어나게 할 수 있지만, 그 1초를 지키는 것 보다 초가 바뀌는 순간 출력을 하는 것이 더 중요하다고 생각해 그렇게 하지는 않았다.

이를 적용하여 문제 1을 해결한 결과는 다음과 같다.

3. 문제 1의 결과

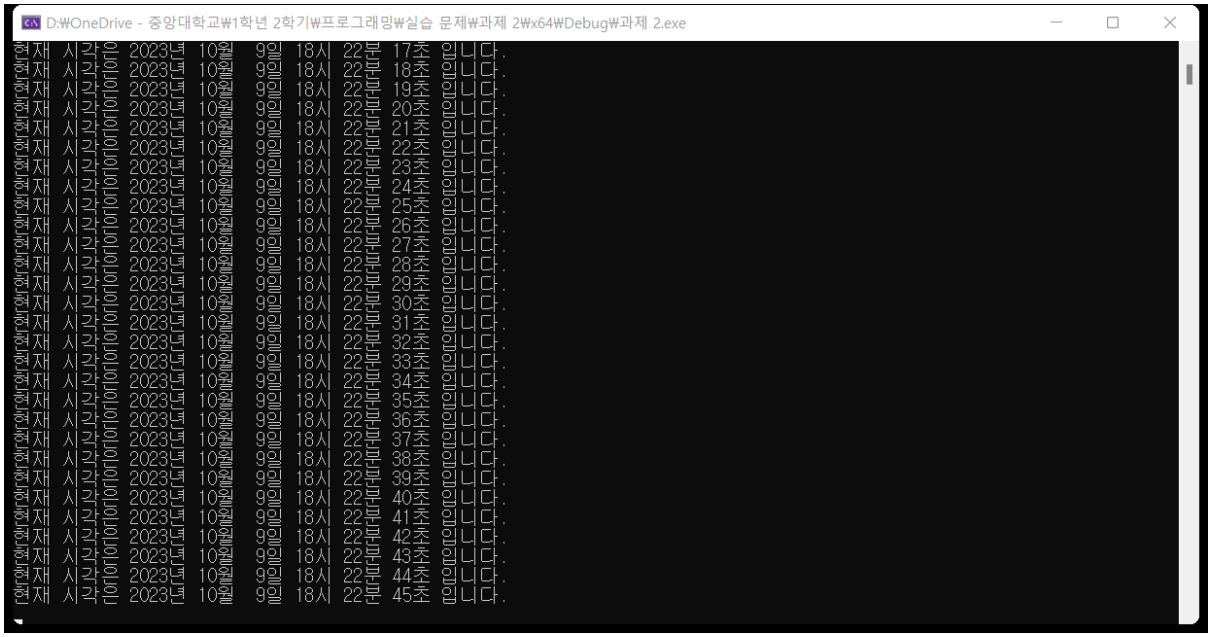

그림 1

31일 계산을 정확하게 할 수 있는지 알아보기 위해 시간대를 저장하는 상수의 값을 바꾸어, 오늘에 22일을 더하도록 해 현재 날짜가 10월 31일을 가리키도록 하고 실행하였다. 아래 그림 2는 그 실행 결과이다.

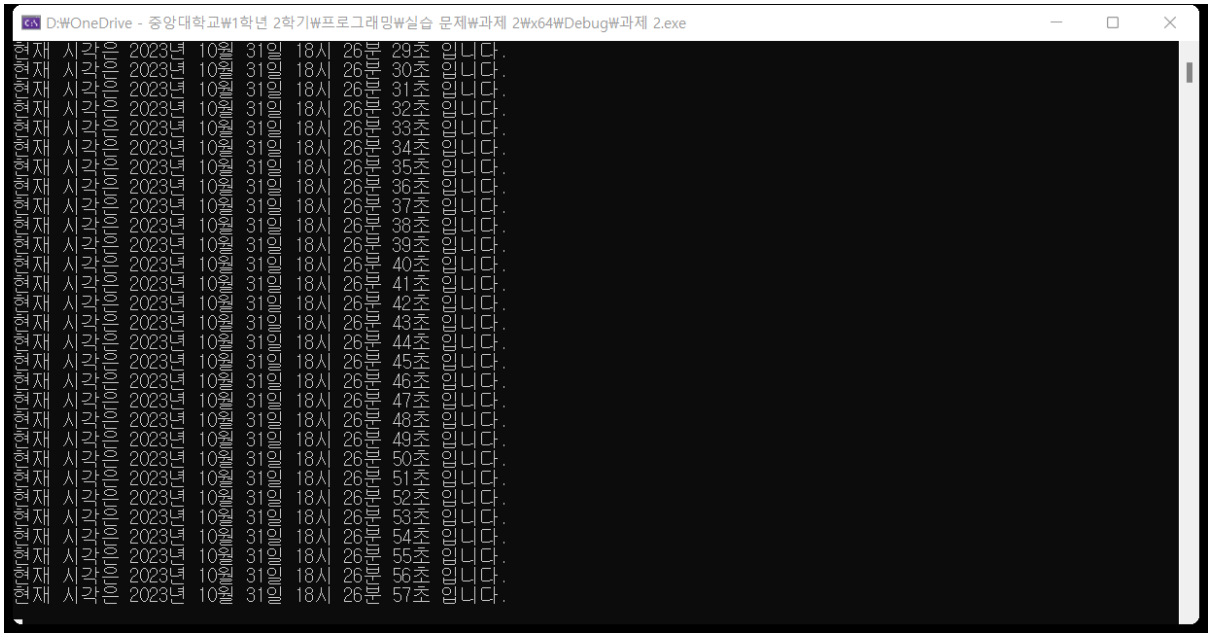

그림 2

위 그림 2의 상태에서 60일을 더 더한 후 실행해 보았다. 그림 3은 그 결과이다. 결과가 12월 31일이 아닌 30일로, 또 2024년이 아닌 2023년으로 제대로 나온 점을 볼 수 있다.

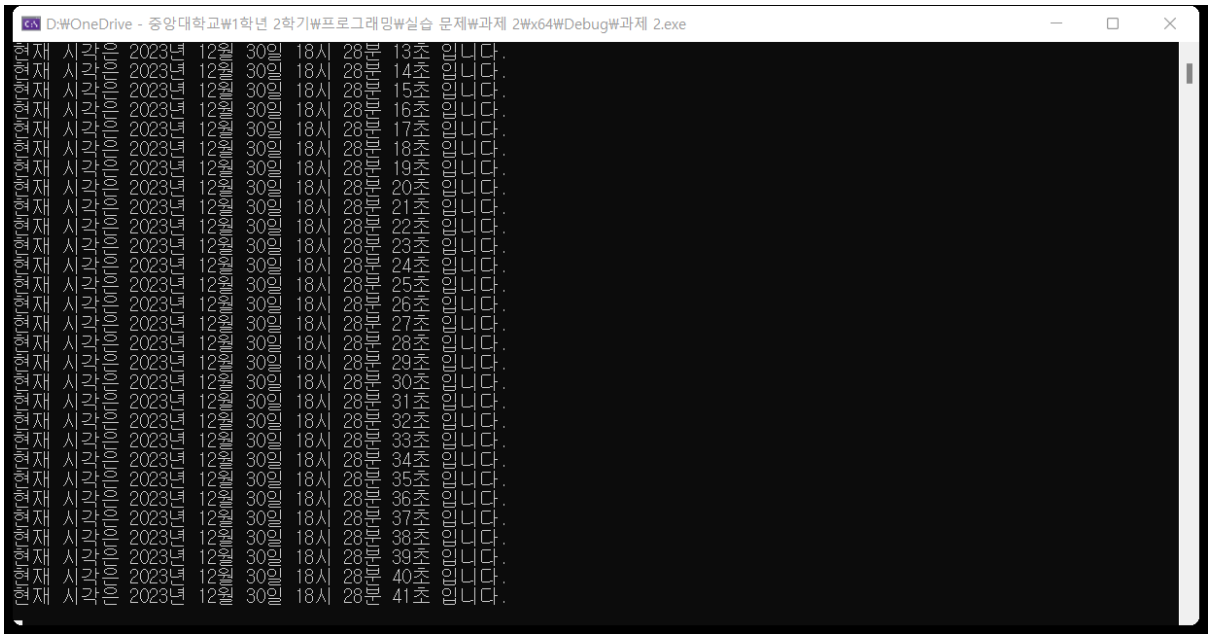

그림 3

위 그림 3의 상태에서 1일을 더하면 12월 31일이 정상적으로 출력된다. 그림 4는 그 결과이다.

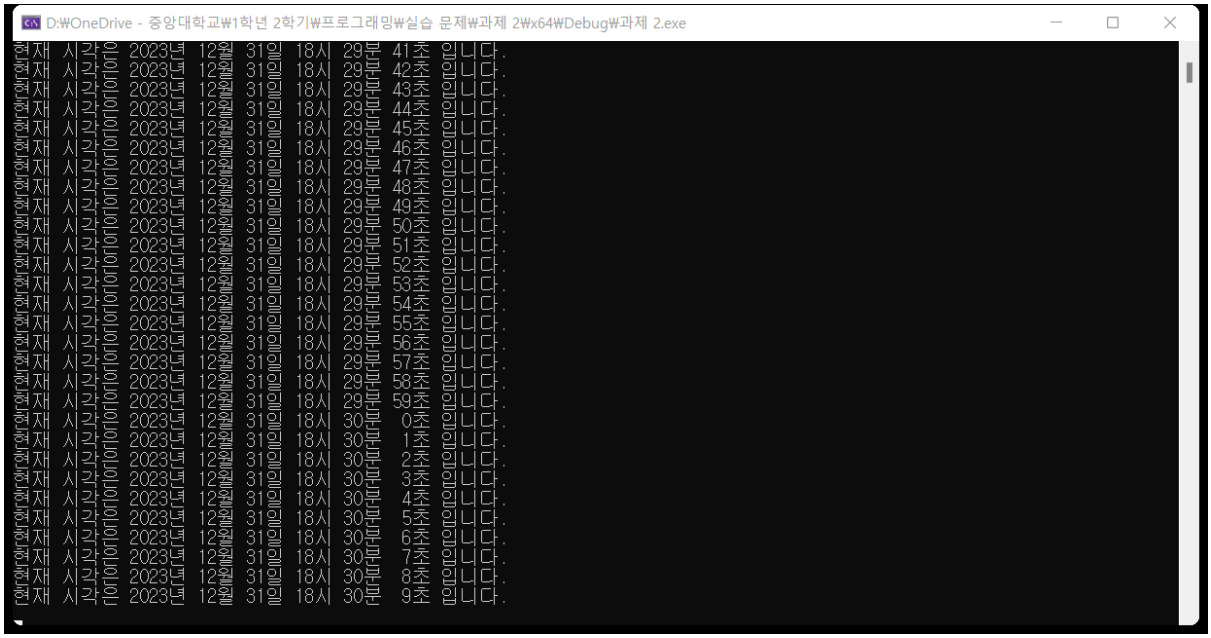

그림 4

위 그림 4의 상태에서 하루를 더 더하면 2024년으로 넘어가며, 이 또한 정상 동작한다. 그림 5는 그 결과이다.

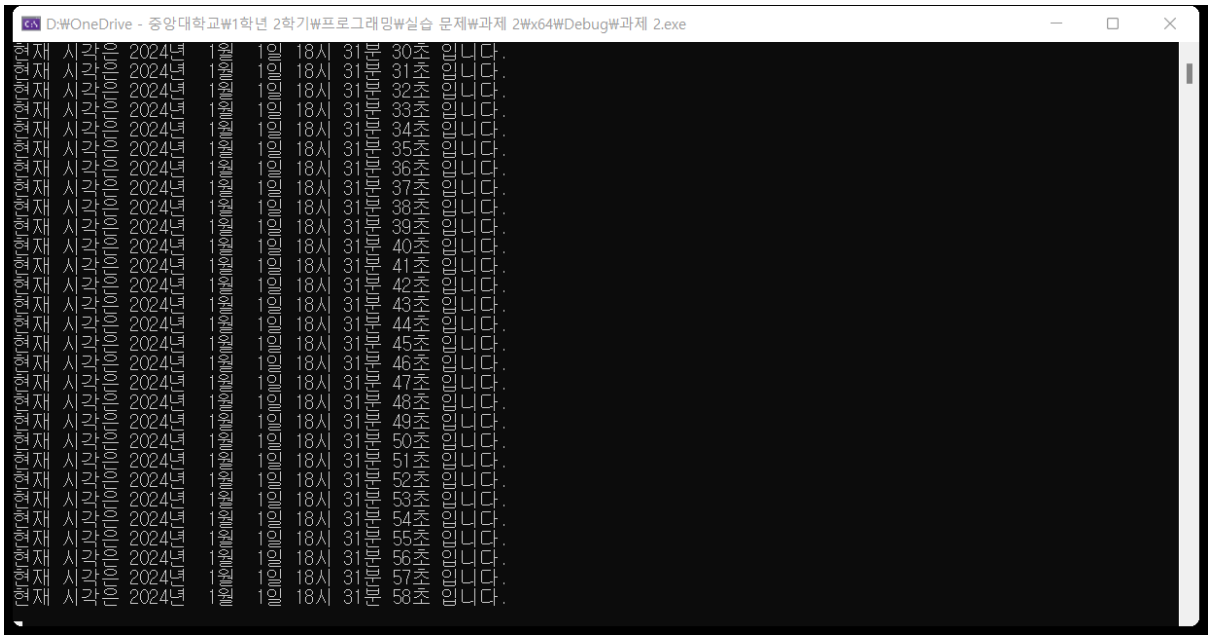

그림 5

윤년 계산이 정확하게 되는지 알아보기 위해 2023년 12월 31일(그림 4)에서 60일을 더했다. 2024년은 윤년이고 1월은 총 31일이므로 결과는 2024년 2월 29일이 나오게 된다. 프로그램 실행 결과는 그림 6으로, 정상 동작한다는 점을 볼 수 있다.

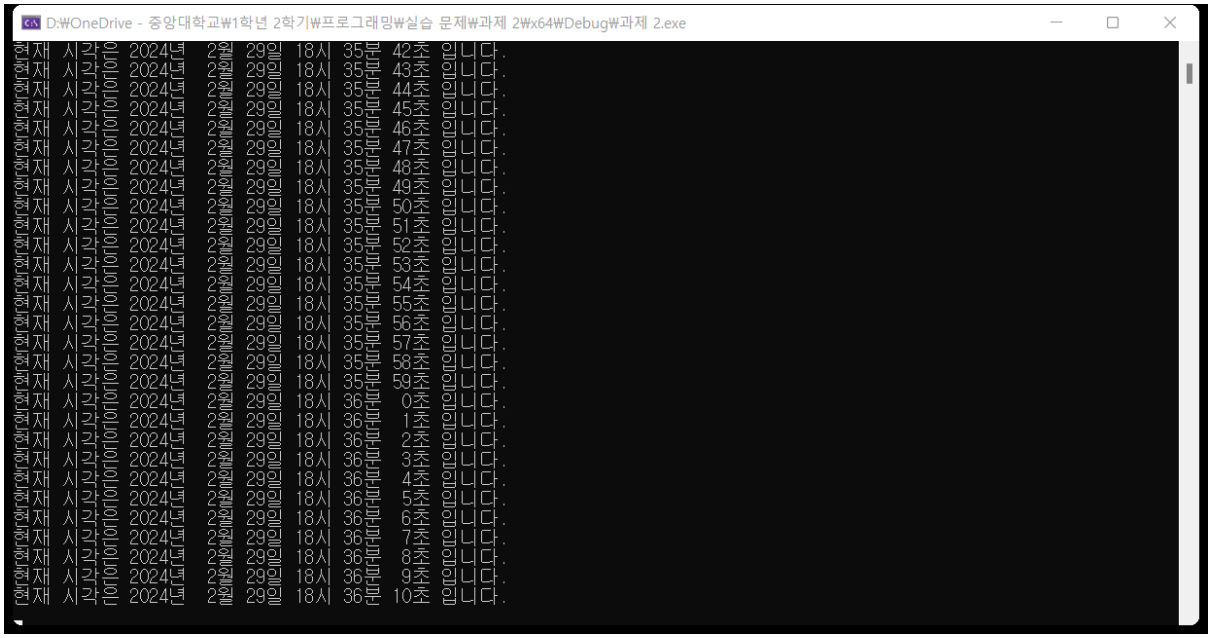

그림 6

2월 보정 과정 중, 보정 전 1월 계산 후 월 값이 2월이지만 날짜가 29일 이내가 아니면 2월 보정이 들어가서 3월이 되는지 확인하기 위해 위(그림 6)에서 하루를 더했다. 그림 7은 그 결과로, 제대로 3월 1일이 되는 점을 확인할 수 있다.

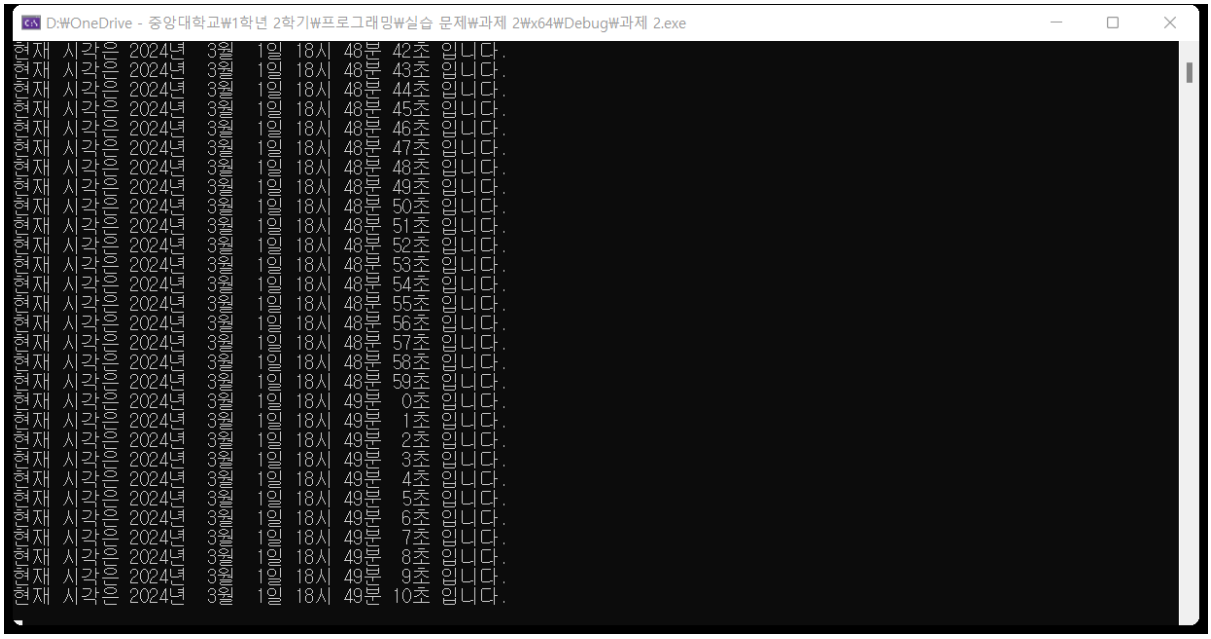

그림 7

년도 윤년 보정에서, 년도를 빼면서 이번에 윤년 체크한 년도와 (기준 년도 + 계산된 년도)가 같아지게 되면 366일을 더한 점도 정상적으로 처리되었는지 확인하기 위해 프로그램이 2024년 12월 31일을 향하도록 했다. 기준 년도인 1970년이 윤년이 아니라는 점을 고려해서, 시간대를 저장하는 상수가 2024년 1월 1일을 가리키는 상태(그림 5)에서 기준 년도만 1971년으로 바꾸어 1년을 추가하면 구현할 수 있다. 이렇게 실행한 결과 그림 8을 보면 결과가 제대로 나오는 것을

확인할 수 있다.

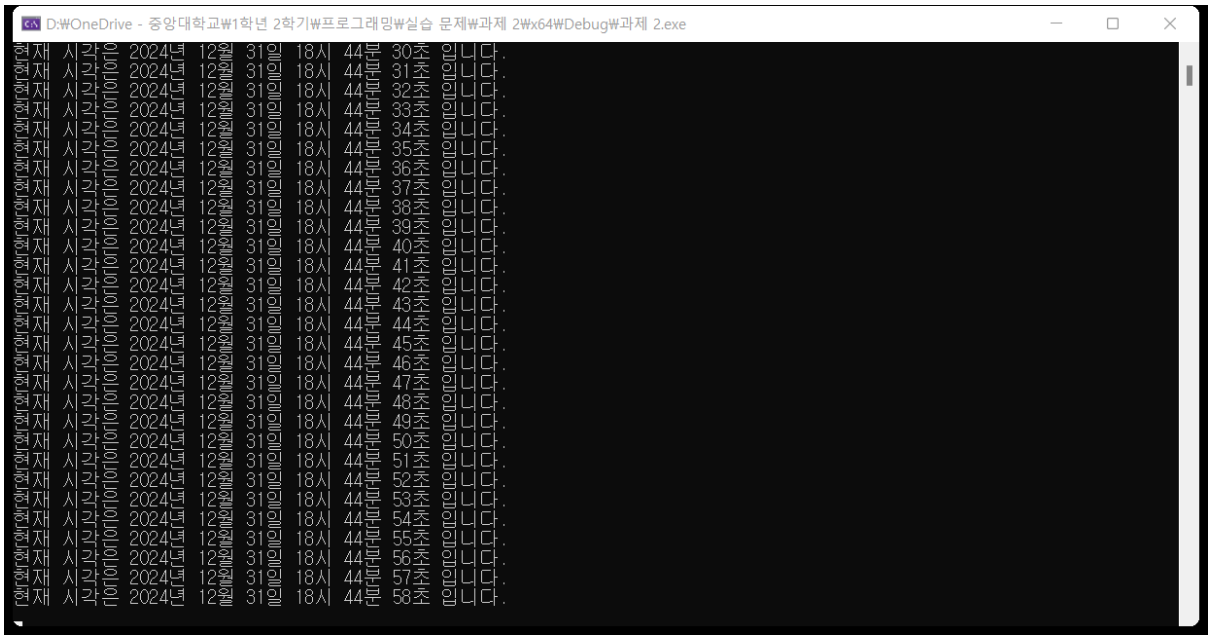

그림 8

이제는 윤년 확인, 보정 코드가 제대로 작동하는지 확인하기 위해, 다시 시간대 상수를 원래대로 돌려놓고, 기준 년도를 1960년으로 10년 전으로 바꾸었다. 1960년~1969년에는 윤년이 1960년, 1964년, 1968년 총 3개 년도가 있지만 2013~2023년에는 윤년이 2016년, 2019년 총 2개 년도밖에 없어 년도만 10년 전으로 돌아가는 게 아니라 날짜도 하루 전을 가리킬 것이다. 실행 결과는 그림 9로, 정상적으로 2013년 10월 8일을 가리키는 것을 알 수 있다.

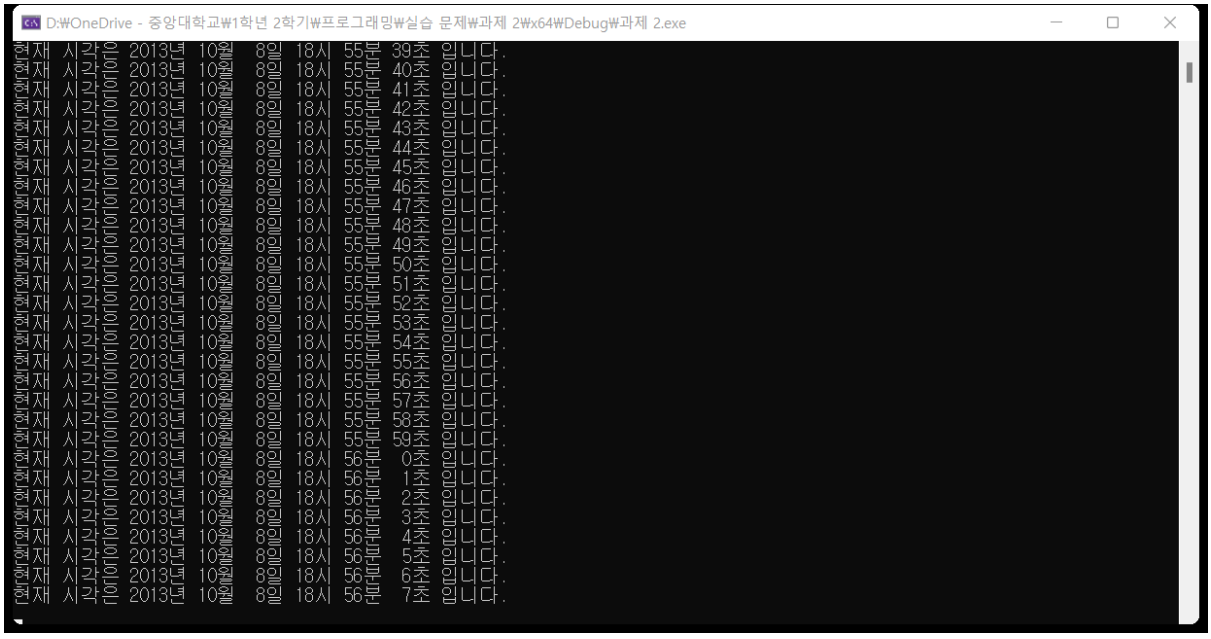

그림 9

윤년 확인 시 100으로 나누어떨어지는 년도는 윤년이 아니지만, 100으로 나누어떨어지면서 400으로 나누어떨어지는 년도는 윤년인데, 이 부분이 정상 동작하는지 알아보기 위해 이번에는 기준

년도에서 100을 빼 1870년으로 바꾼 뒤 실행했다. 1970년~2023년 중 다른 년도는 100년을 빼도 윤년 여부가 바뀌지 않지만 2000년은 1900년과 달리 윤년이어서, 기준 년도에서 100을 빼면 윤년의 수가 하나 줄어 날짜도 다음 날을 가리키게 된다. 실행 결과인 그림 10을 보면 이 또한 정상적으로 1913년 10월 10일을 가리키는 것을 볼 수 있다.

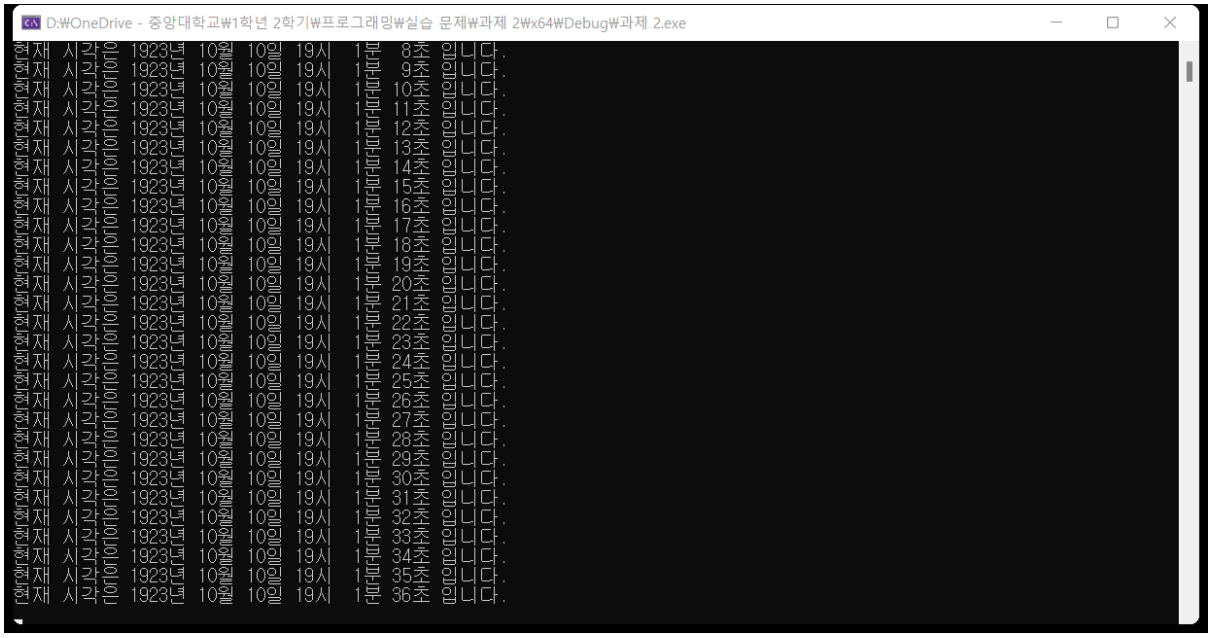

그림 10

위와 같이 모든 상황에 대해 정상적으로 작동하는 것을 볼 수 있다.

4. 문제 2: 빙고 게임

사람과 컴퓨터가 하는 5*5 빙고 프로그램을 만든다. 사람과 컴퓨터는 각각의 빙고 판을 가지며, 자신의 것만 볼 수 있고 타인의 것은 볼 수 없다. 빙고 판에 들어가는 숫자는 1~25이며 한쪽의 빙고 판에 있는 숫자가 다른 쪽에 없는 일은 없다. 게임이 시작되면 사람과 컴퓨터는 번갈아 가며 숫자를 하나씩 부르고, 가로, 세로, 대각선 중 총 5개의 줄이 완성되면 게임은 종료되며 이를 완성한 사람이 이긴다. 이때 컴퓨터가 좀 더 똑똑하게 숫자를 부르도록 몇 가지 규칙을 만들어 적용해 본다.

5. 문제 2 – 빙고 판 제작 및 게임 진행 방법

먼저 사람과 컴퓨터의 보드를 저장하기 위해 가로, 세로가 5인 2차원 배열 2개를 저장할 수 있는 3차원 배열 1개를 선언한다. 원래 보드 안의 숫자 자료형은 char로 하려 했으나 프로그램 실행 중 프로그램이 문자열로 인식해 오류가 자주 나는 바람에 int를 사용하였다. (다만 오류를 무시

하고 프로그램을 계속 돌리면 별다른 문제가 보이지 않고 잘 실행되기는 한다.) 또한 보드에서 완성된 줄을 세서 저장할 1차원 배열을 선언했고, 확장성을 고려해 각 보드의 이름, 번호를 고르는 방식을 저장하는 1차원 배열도 선언하였다. 사람과 컴퓨터의 보드를 설정하기 전 랜덤시드를 초기화하고, 사람과 컴퓨터의 보드를 각각 랜덤하게 숫자를 배치하였다. 랜덤하게 숫자를 배치하는 방법은, 보드의 앞부분에 넣을 숫자부터 랜덤하게 뽑으나 뽑을 때 0부터 (최대 숫자 – 이미 뽑은 숫자의 수 - 1) 사이의 무작위 값을 뽑고 여기에 1을 더한다. 이렇게 숫자를 뽑게 되면, 적어도 앞에서 뽑은 숫자와 중복되어서 숫자를 다시 뽑는 일은 없게 된다. 다만 이렇게 뽑은 수를 바로 보드에 넣을 수는 없고 약간의 보정 작업이 필요하다. 위에서 뽑은 값은 이번에 뽑힌 수가 아직 안 뽑힌 수 중 작은 것 순서대로 몇 번째인지를 알려주는 숫자이지 이번에 뽑은 수 그 자체가 아니기 때문이다. 따라서 이번에 뽑힌 수보다 작은 수의 개수를 뽑힌 수에 더해 주는 작업이 필요하다. 이 작업은 보드의 맨 앞부터 숫자를 하나하나 확인하며 이번에 뽑힌 수보다 작은 수가 있으면 이번에 뽑힌 수를 1만큼 더해주는 방식으로 구현할 수 있다. 이때 주의할 점은 보드의 앞부분에서 이번에 뽑힌 수보다 크다고 넘어갔는데 뒤에서 뽑힌 수의 크기가 커지다가 앞에서 그냥 넘어간 숫자보다 커지는 경우도 계산해야 한다는 점이다. 이를 해결하기 위해 위의 보정작업을 여러 번 반복하되, 보정작업 1회 전의 숫자와 2회 전의 숫자를 별도의 변수에 저장해 두고, 보드의 맨 앞부터 숫자를 하나하나 확인하되 작업 2회 전의 숫자를 초과하고 작업 1회 전의 숫자 이하인 숫자가 있으면 뽑은 수를 1만큼 증가시키고, 보정작업 전후 뽑은 숫자가 변하지 않을 때 보정작업을 종료하는 방법을 사용하였다. 또한 2차원 배열의 요소 하나하나를 보는 것은, 원래라면 이중 for 문을 이용하고 배열의 요소에 접근할 때는 board[i][j](이때 5*5 배열에서 i<5, j<5)와 같은 코드를 사용하는 것이 맞으나, 배열의 요소들은 빈틈없이 이어져 있다는 점을 고려하여 단일 for 문을 사용해 ((int*)board)[i](이때 5*5 배열에서 i<25, 다만 형 변환과 괄호 2번을 모두 사용하지 않으면 배열의 요소에 접근할 수 없고 다른 값을 읽게 된다.)와 같은 코드를 사용하였다. 이렇게 무작위 값 25개를 뽑아 배열에 저장하는 작업을 2번 반복하면 사람과 컴퓨터의 보드가 완성된다. 확장성을 고려해 사람이 자신의 보드를 직접 고를 수 있도록 하는 함수도 같이 만들어 두었지만, 이번에는 사용하지 않는다. 또한 보드 제작 코드를 만들 때 코드가 잘 동작하는지 확인하기 위한 확인용 코드와 함수도 만들었지만, 보드를 충분히 많이 만들어 보아 코드가 잘 동작한다는 점을 확인한 이후로 사용하지 않는다.

보드가 완성되었으므로 게임 진행과 게임 종료 여부, 승자를 판정하는 것만 남았다. 먼저 사람이 선이 되어 보드를 보여주고 지울 숫자를 입력받는다. 숫자 입력 전에는 숫자 선택에 도움이 되도록 현재 보드도 같이 보여준다. 보드 출력 방법은 후술한다. 실제 빙고 게임에서 실수로 이미 불렸던 숫자를 다시 불러 자신이 숫자를 부를 기회를 날리는 경우도 존재하므로, 여기서도 이미 불렸던 숫자도 부를 수 있도록 하였다. 숫자를 입력받으면 어떤 숫자를 입력받았는지 확인차 입력받아 저장된 값을 출력하고, 보드의 맨 앞부터 값을 하나하나 확인하여 입력된 숫자와 일치하는 숫자가 있으면 열린 숫자(불린 수)로 바꾼다. 이때 불린 숫자와 그렇지 않은 숫자는 보드의 저장된 수의 부호(양수, 음수)로 판단하며, 음수일 때가 열린(불린) 숫자로 판단한다. 또한 확장성을 고려하여 보드 생성 규칙이 달라 하나의 보드에 같은 숫자가 들어갈 때를 대비해, 이미 입력된

숫자와 일치하는 숫자를 찾아 열린 수로 바꾸어도 보드의 끝 숫자까지 확인을 계속한다. 그리고 지워진 줄의 수를 확인하고(이 과정도 후술한다), 승리 기준인 5개의 줄이 완성되면 게임을 끝내고, 그렇지 않으면 숫자를 지운 후의 보드를 보여준 후, 화면에 출력된 게 누적되면 화면이 더러워지므로 화면의 글을 전부 지운 뒤 컴퓨터의 차례로 넘어간다. 컴퓨터는 이미 불린(열린) 숫자를 포함해 1부터 25까지의 숫자 중 무작위의 숫자를 뽑으며, 이후 동작은 보드를 화면에 출력하는 과정만 생략하고 사람이 부른 이후와 같다. 어떤 참여자가 어떻게 숫자를 부를 지는(직접 입력할지, 아니면 무작위로 결정할지는) 앞에서 언급한 1차원 배열에 저장된 값을 이용한다.

보드 출력은 맨 윗줄에 보드의 이름(여기서는 뭐가 사람(플레이어)의 보드고 뭐가 컴퓨터의 보드인지)을 출력하고, 그 아랫줄부터 보드 출력, 이후 마지막 줄에는 현재 완성된 줄의 수를 보여준다. 또한 보드의 이름과 완성된 줄의 수를 보여줄 때는 string.h에 정의된 strlen 함수를 이용하여 중앙정렬하고(다만 이는 한글의 길이가 2여서 가능했다. 화면 출력 시 한글은 영문자 2개만큼의 크기를 차지하므로 한글의 길이가 3인 경우 중앙정렬 계산이 틀어졌을 것이다), 2개의 보드는 상하가 아닌 좌우로 정렬한다(따라서 보드를 출력할 때 2개 보드의 요소를 왔다 갔다 하면서 출력하게 된다. 첫 번째 보드의 첫 번째 줄을 출력하고 나면 화면상으로는 같은 줄에 두 번째 보드의 첫 번째 줄도 출력하고, 개행 후 첫 번째 보드의 두 번째 줄을 출력하고 화면의 같은 줄에 두 번째 보드의 두 번째 줄도 출력하는 식이다.) 숫자를 출력할 때는 이미 불린(열린) 숫자이면 괄호로 표시하고 어느 쪽 보드이던 숫자를 공개하지만, 그렇지 않은 경우 공개 대상인 보드(게임 중에는 첫 번째 보드)인 경우만 공개하고 그 외에는 숫자가 나올 위치에 전부 '-'를 출력하게 했다. 공개 대상인 보드로 0번째를 받는 경우(게임 종료 시에 해당), 모든 보드가 공개 대상인 보드로 보고 모든 보드를 공개하도록 했다. 이 외에도 확장성을 고려해 빙고 참가자 증가로 보드 개수가 증가하여 모든 보드를 한 줄에 출력하기 어려운 경우를 대비해 좌우로 정렬하는 보드의 개수를 제한하고, 이 제한을 넘으면 보드 개수만큼만 좌우로 정렬하고 남은 것들은 아랫줄에 정렬되도록 했다. 또한 숫자 출력으로 printf("%2d", board[k][j][l]);를 사용하면 지금 당장은 숫자가 올바르게 정렬되겠지만 보드 크기가 증가하여 3자리 숫자가 나오는 경우의 대처가 어려우므로, 코드 앞에 #define을 이용해 Num_Digit을 앞으로 빼고, printf("%*d", Num_Digit, board[k][j][l]);를 사용함으로써 숫자의 자릿수가 증가할 때 앞 #define 부분만 바꾸면 문제없게 만들었다. 이 Num_Digit은 보드의 이름이나 완성한 줄 수를 출력할 때 중앙정렬을 맞추기 위한 보드의 가로 길이 연산에도 활용된다.

지워진 줄의 수를 확인하는 과정은 다음과 같다. 과정 시작 전 결괏값을 저장하는 변수를 0으로 초기화한 후 시작한다. 먼저 한 가로줄에 양수인 숫자(불리지(열리지) 않은 숫자)가 있는지 왼쪽부터 확인한다. 해당하는 숫자가 하나라도 있으면 아무런 작업 없이 확인을 종료하고, 해당하는 숫자 없이 하나의 가로줄을 전부 확인하였으면 완성된 줄임을 확인하고 완성된 줄의 수를 저장하는 변수를 1만큼 증가시킨다. 이를 모든 가로줄에 대해 반복하고, 같은 방법으로 세로줄도 확인한다. 대각선 줄의 경우 왼쪽 위에서 오른쪽 아래로 향하는 대각선과 오른쪽 위에서 왼쪽 아래로 향하는 대각선이 존재한다. 보드를 좌표로 보고 왼쪽 위의 좌표를 (1, 1), 오른쪽 아래의 좌표를 (5,

5)로 할 때, 왼쪽 위에서 오른쪽 아래로 향하는 대각선은 좌표의 x값과 y값이 같다는 특징이, 오른쪽 위에서 왼쪽 아래로 향하는 대각선은 x값과 y값의 합이 6(보드의 한쪽 변의 길이 + 1)으로 일정하다는 특징이 있다. 따라서 대각선 완성 여부는 변수(예를 들어 j)를 1부터 5까지 1씩 증가시켜 가면서, (j, j)의 값 5개, (j, 6-j) 값 5개씩만 확인하면 된다. 위 가로줄, 세로줄 확인하듯이 2번만 더 확인하면 완성된 줄 수 확인은 끝났다. 이를 빙고 참여자(여기서는 2)만큼 반복해 모든 보드의 완성된 줄 수를 구할 수 있다. 줄 수 확인 작업을 숫자 하나 부를 때마다 반복하므로 이번에 불린(열린) 수의 위치를 지나는 줄만 판정하는 것도 가능하나, 확장성을 고려해 번호를 여러 개 부르거나 다른 코드를 수정해 일부가 게임 시작 전에 열린 상태로 시작하는 경우에도 문제없는 확인을 위해 매번 보드 전체를 확인하도록 했다.

앞에서 계속 언급했지만, 확장성을 많이 고려했기 때문에 코드 앞의 #define에서 정의된 숫자와 main 함수를 바꾸는 것만으로 더 많은 사람이 동시에 빙고를 한다거나, 빙고 판의 크기를 바꾸거나, 승리 조건(필요한 줄의 수), 보드 출력 시 숫자끼리의 간격, 보드끼리의 간격, 좌우로 출력되는 최대 보드 수, 각 플레이어가 숫자를 뽑는 방법이나 보드를 설정하는 방법을 바꿀 수 있다.

6. 문제 2 – 컴퓨터가 숫자를 똑똑하게 부르도록 하는 법

앞에서처럼 컴퓨터가 완전 무작위로 숫자를 부르면, 이미 불렸던 숫자가 다시 나오는 경우가 많다. 코드를 약간 수정해 2명의 참가자가 모두 무작위 값 만을 부르게 하면, 5줄이 완성되어야 하는 특성상 숫자를 25번 이상 불러도 게임이 끝나지 않는 경우도 생긴다. 우선 이것 먼저 해결하기 위해 이미 불린 숫자는 피하도록, 무작위 값을 뽑고 내(컴퓨터의) 보드를 처음부터 끝까지 확인하여 숫자가 중복되었으면 다시 뽑도록 바꾸었다. 보드를 만들 때처럼 불린(열린) 수의 개수를 확인하는 방법으로 무작위 값을 다시 뽑는 일은 없도록 만들 수도 있었으나, 사람도 어떤 구간 내의 숫자 중 특정한 숫자 몇 개를 빼고 아무거나 부르라고 하면 일단 아무 숫자나 뽑은 후 그 숫자가 특정한 숫자에 포함되는지 보고, 포함되면 다시 뽑는 과정을 거쳤다는 점을 고려하여 사람이 숫자 뽑듯이 무작위 값을 뽑도록 하기 위해 보드를 만들 때처럼 무작위 값을 뽑지는 않았다. 이렇게 하는 경우 2명의 플레이어가 모두 무작위 값을 부르면 게임은 빨리 끝나지만, 거기서 그칠 뿐 아직 똑똑하게 번호를 부를 수 있다고는 할 수 없다.

컴퓨터가 더욱 똑똑하게 숫자를 고를 수 있도록 하기 위해 각 위치별로 우선도를 계산해 우선도가 가장 높은 위치에 있는 숫자를 부르도록 했다. 기본 우선도는 이미 불린(열린) 수의 경우 0, 그 외에는 각 위치를 지나가는 줄의 수로 설정했다. 예를 들어 2개의 대각선과 관계없는 위치면 2, 대각선 중 1개만 지나가면 3, 2개의 대각선이 모두 지나가는 보드의 중앙은 4이다. 여기에 가로줄, 세로줄, 대각선 별로 불린(열린) 숫자의 수를 계산해 우선도를 더한다. 여기서 더해지는 우선도에 대해서는 후술한다. 우선도의 계산이 끝나면 가장 높은 우선도에 해당하는 위치에 있는 숫자를 부르며, 가장 높은 우선도가 여러 개 있으면 그중에서 부를 숫자의 위치를 무작위로 정한다. 무작위로 정하는 방법은 보드를 만들 때와 비슷한 방법을 사용한다. 먼저 가장 높은 우선도

수치와 그와 같은 우선도를 가지는 위치가 몇 개 있는지를 구하고, 그 우선도를 가진 위치 중 앞에서부터 몇 번째에 위치한 숫자를 부를 지를 무작위로 뽑는다. 다시 우선도가 저장된 배열을 처음부터 확인하면서 앞에서 구한 가장 높은 우선도를 발견하면, 무작위로 뽑은 값을 1씩 줄이다가 무작위로 뽑은 값이 0이 된 이후, 처음 만난 (가장 높은 우선도의 위치)에 있는 숫자를 고른다.

a			
	*	c	
	b	*	

그림 11

이 경우 가로줄, 세로줄, 대각선별로 불린(열린) 숫자의 수로 계산되어 더해지는 우선도가 숫자 결정에 영향을 미치기에 매우 중요하다. 처음에는 (지워진 줄의 수)*4 만큼을 더했다. 4를 곱한 이유는 기본 우선도는 이 더해진 우선도가 같은 경우에만 영향을 미치도록 하기 위함이다. 그러나 이 경우, 왼쪽 그림 11과 같은 상황에서 불린(열린) 숫자의 위치를 '*'이라 했을 때, 한 대각선에 2개의 숫자가 불리(열리)었음에도 a, b, c 위치의 우선도가 전부 같아 b와 c 위치의 숫자를 부를 가능성이 있다. 이에 따라 우선도 식을 수정하여 (지워진 줄의 수)*(지워진 줄의 수)*4만큼을 더하도록 했다. 이 경우 어지간해서는 이상한 값을 부르지는 않았지만, 이렇게 설정하고 테스트를 한 결과 다음 사진과 같은 문제가 발생하였다.

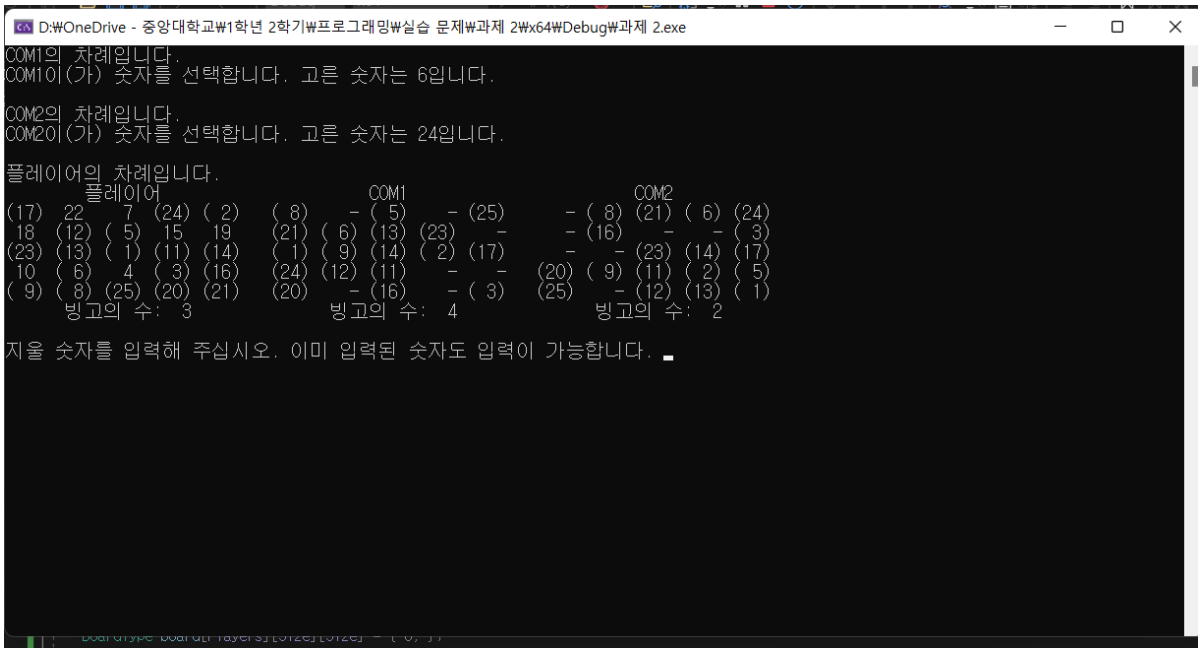

그림 12

위 그림 12는 #define으로 정의된 값을 바꾸어도 코드가 정상 작동하는지 확인하기 위해 게임 참가자 수를 3으로 변경하고 방금 언급한 우선도를 적용한 컴퓨터를 둘 두고 빙고 게임을 하던 중 발생한 상황이다. 여기서 COM1이 자기 차례에 숫자 6을 불렀다. 24를 불렀으면 한 줄을 완성할 수 있음에도 불구하고 6 위치의 우선도가 높게 나와 6을 부르게 되었다. 결과적으로는 COM2가 24를 불러 COM1 입장에서 상관은 없게 되었지만, 이건 우연히 맞아떨어진 것이고 한 줄을 완성하는 위치보다 다른 위치가 우선도가 높게 나온 것은 피해야 한다. COM1 보드에서 6과 24가 열리기(불리기) 전의 2개 위치에 대한 우선도를 계산해 보자면, 6 위치는 대각선 1개가 지나가는 위치이므로 우선도 기본값은 3이고, 가로선의 경우 3개가 지워졌(열렸)으므로 우선도가 3*3*4=36

만큼 증가하고, 세로선의 경우 2개가 지워졌(열렸)으므로 우선도가 2*2*4=16만큼 증가하고, 대각선의 경우 3개가 지워졌(열렸)으므로 우선도가 3*3*4=36만큼 증가하니, 계산된 우선도 값은 91이다. 24 위치는 2개의 대각선 모두 지나지 않는 위치이므로 우선도 기본값은 2이고, 가로선의 경우 4개가 지워졌(열렸)으므로 우선도가 4*4*4=64만큼 증가하고, 세로선의 경우 2개가 지워졌(열렸)으므로 우선도가 2*2*4=16만큼 증가하니, 계산된 우선도 값은 82이다. 기본 우선도를 빼고 우선도 값을 4로 나누어 (지워진 줄의 수)*(지워진 줄의 수) 값의 합만 봐도 각각 22와 20으로 2 차이가 난다. 따라서 이를 해결하기 위해 우선도 식을 또다시 수정하여 (지워진 줄의 수)*(지워진 줄의 수)*(지워진 줄의 수)*4만큼을 더하도록, 그러니까 (지워진 줄의 수)를 세제곱 하도록 했다. 이 경우 일단 가로, 세로의 길이가 5인 지금은 문제가 발생하지 않지만, 보드의 크기를 키워 게임을 한다면 문제가 생길 가능성이 높다.

실제 사람끼리 빙고를 진행할 때는 상대의 심리에 따라 상대 보드의 숫자 배치를 예측하기도 하고, 상대방의 표정 등 감정을 확인하여 이를 숫자를 고를 때 활용하기도 하지만, 이 사람과 컴퓨터와의 빙고 게임의 경우 서로가 서로의 감정이나 심리를 알 수도, 분석할 수도 없으며, 보드의 숫자 배치의 경우 양쪽 모두 무작위로 정해지기 때문에 이를 반영하지 않았다. 또한 이 보고서를 작성하는 본인의 경우 이미 불렸던 숫자와 가장 가까운 숫자를 먼저 부르는 경우가 있는데, 이는 그냥 작성자 본인의 스타일일 뿐 게임 진행이나 결과에 크게 영향을 끼치지 않는다고 판단하여 반영하지 않았다.

사실 가장 좋은 우선도 계산식을 얻는 방법은 AI를 이용하는 것이지만, 이 과제에서는 이용하지 않았다. 꼭 AI가 아니더라도 개개인의 관점에 따라 우선도 계산식이 바뀔 수도 있다. 예를 들어 위 예시 그림에서 COM1이 24가 아닌 6을 부르는 게 더 좋은 선택이라고 생각하는 사람도 있을 수 있고 제곱이 아닌 다른 방법을 사용하는 게 더 좋다고 생각하는 사람도 존재할 것이다. 위의 우선도 계산 방식은 보고서 작성자 본인이 빙고에서 숫자를 고르는 방식을 반영한 것이며, 작성자 본인이 생각하지 못한 더 좋은 방법이 존재할 수 있다.

7. 문제 2의 결과

문제 2의 코드는 숫자를 입력할 때마다 화면에 나타난 모든 글자를 지우므로, 화면을 지우기 전 출력 결과를 하나하나 캡처했다.

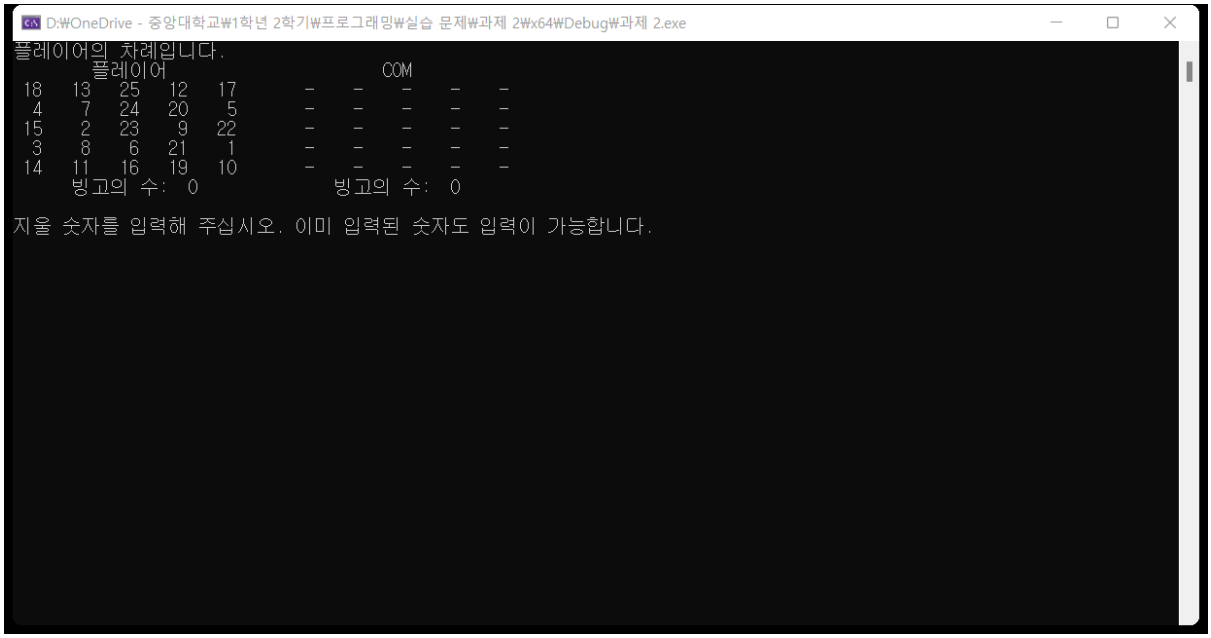

그림 13

위 그림 13은 실행하자마자 나오는 출력 결과이다. 보드 설정은 양쪽 모두 출력이 없으므로 사람의 차례에 현재 보드를 보여주고 사람의 입력을 기다리는 상태이다. 컴퓨터의 보드는 이미 불린(열린) 번호만 보이지만, 현재 불린(열린) 번호가 없기 때문에 아무 숫자도 보이지 않는다.

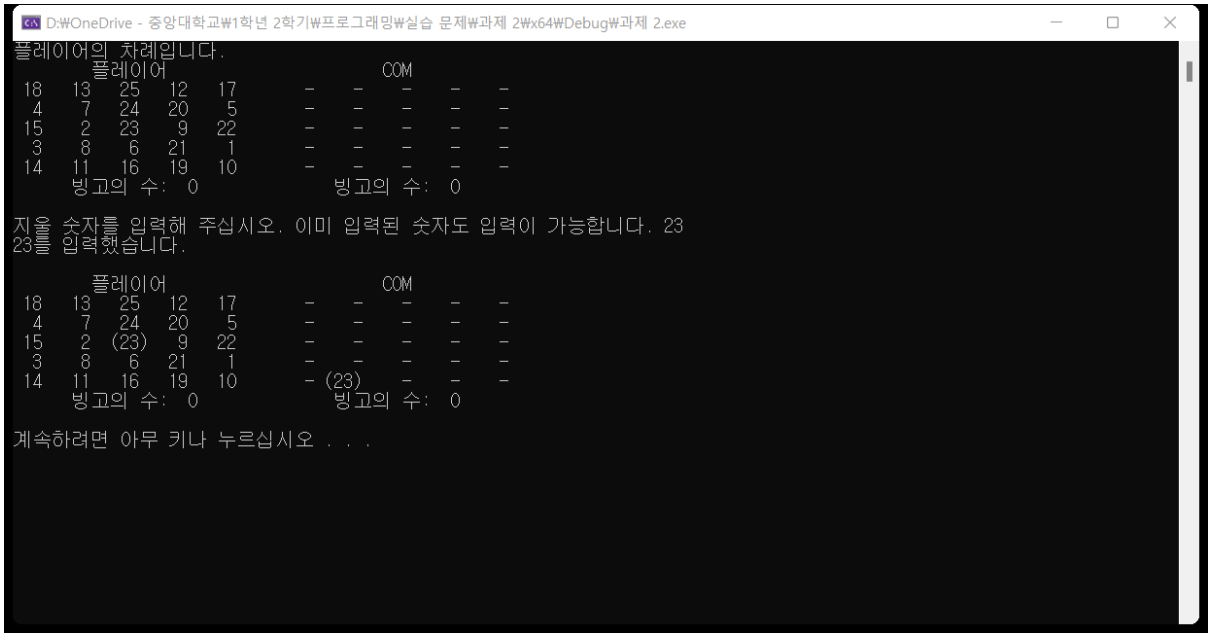

그림 14

위 그림 14는 그림 13의 상태에서 사람이 숫자 23을 입력한 직후의 출력 결과이다. 지운 직후에 화면을 지우는 게 아니라 지운 후의 보드를 한 번 보여준 후 화면에 출력된 글자를 지우기 때문에, 그림 13에서 출력된 글자도 남아 있는 것을 볼 수 있다. 보드 출력의 경우, 컴퓨터의 숫자 중 23이 지워져(열려) 공개되었기 때문에 컴퓨터의 숫자 중 23과 그 위치만이 공개되어 있고, 나머지는 공개되어 있지 않다. 여기서 아무 키나 누르면 아래의 그림 15처럼 컴퓨터가 숫자 하나를

고르고 다시 사람의 차례가 된다.

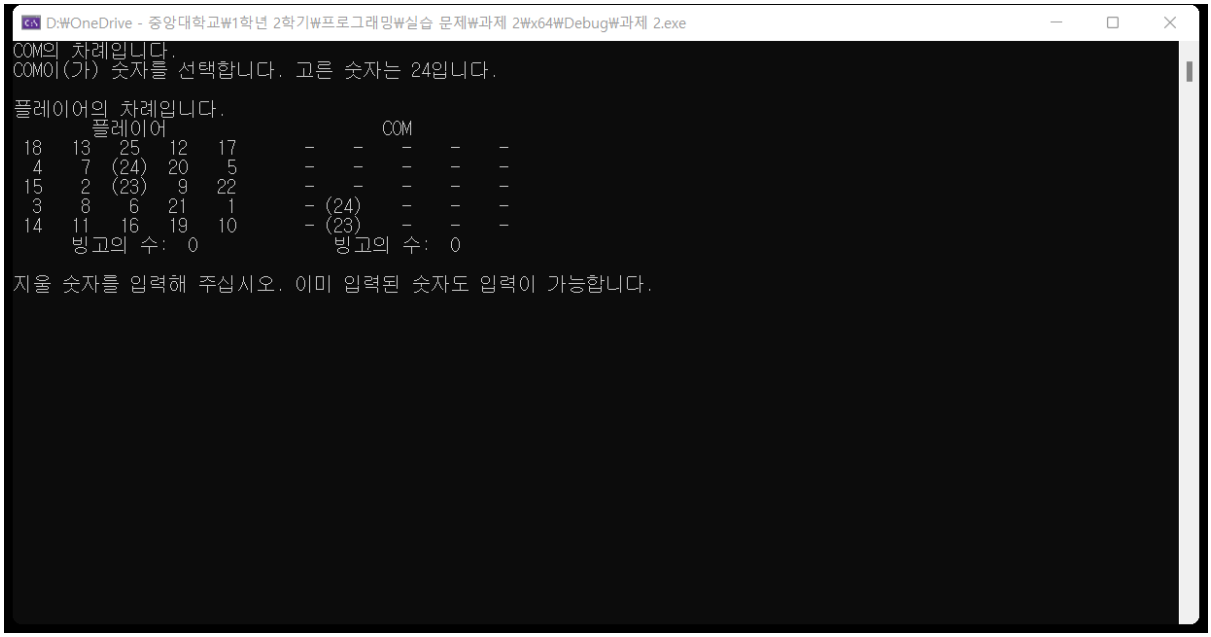

그림 15

위 그림 15는 그림 14에서 아무 키나 누른 이후의 상황이다. 컴퓨터가 숫자 24를 불렀고, 다시 사람의 차례가 되었다. 추가로 컴퓨터의 보드에서 숫자 24와 그 위치도 공개되었다. 우선도 계산 시에 23을 포함한 가로줄과 세로줄에 각각 아직 불리지 않은(열리지 않은) 위치에 기본 우선도 외 추가 우선도가 더해졌을 것이고, (이때 둘 다 줄에서 1개만 지워졌(열렸)으므로 더해지는 우선도는 모두 같다.) 이렇게 더해진 위치 8개 중 대각선 1개가 지나가는 4곳이 가장 우선도가 높았을 것이다. 이 4곳 중 컴퓨터는 무작위로 1곳을 정해 숫자를 고르게 되고, 최종적으로 컴퓨터는 숫자 24를 고르게 되었다.

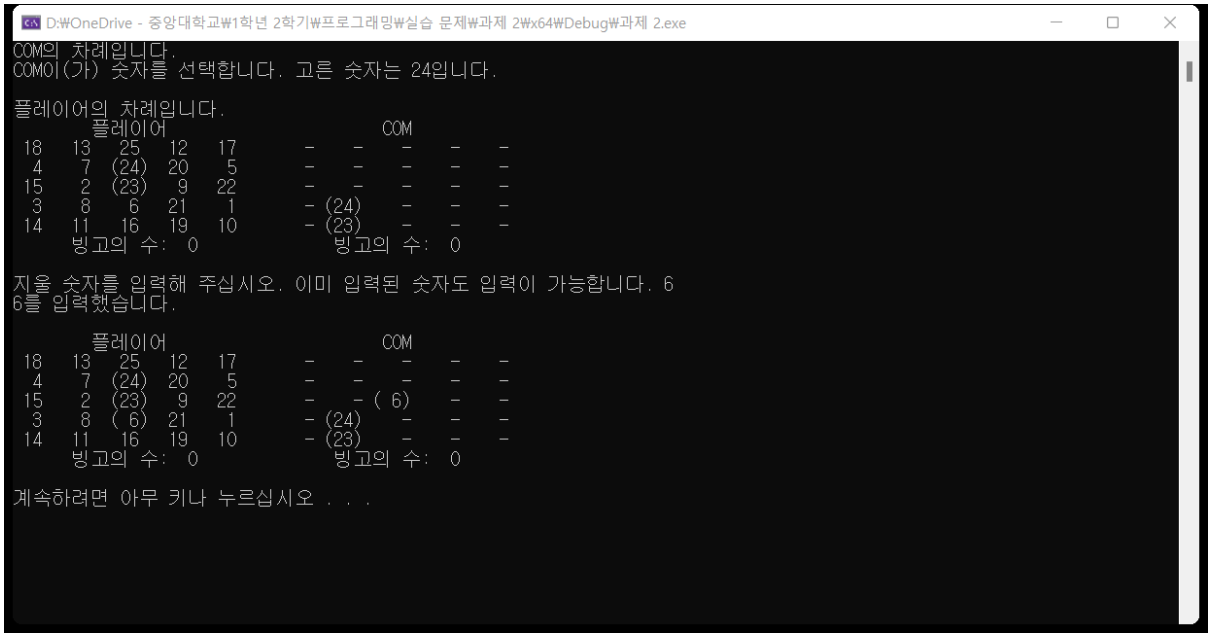

그림 16

위 그림 16은 그림 15의 상태에서 6을 입력한 상태이다. 그림 14의 상태와 마찬가지로, 아직 화면의 모든 글자가 지워지지는 않았고, 컴퓨터의 보드에서 추가로 숫자 6과 그 위치가 표시되었다. 여기서 아무 키나 누르면 컴퓨터의 우선도 계산 결과는 그림 17과 같을 것이다. 여기서 주황색 배경은 우선도가 가장 높은 위치, 회색 배경은 이미 지워진(열린) 숫자의 위치이다. 따라서 컴퓨터는 우선도가 가장 높은 주황색 배경의 위치 2곳 중 1곳의 숫자를 고르게 된다.

7	34	6	2	35
2	39	6	35	2
6	38		6	6
6		10	11	6
39		10	6	11

그림 17

이후 결과 캡처 사진이 너무 많아질 것을 고려해, 그림 13, 그림 15와 같이 화면의 글자가 즉시 지워지지 않는 경우의 캡처 사진은 생략하고 그림 14, 그림 16과 같이 화면의 글자를 전부 지우기 직전의 캡처 사진만 이용했다.

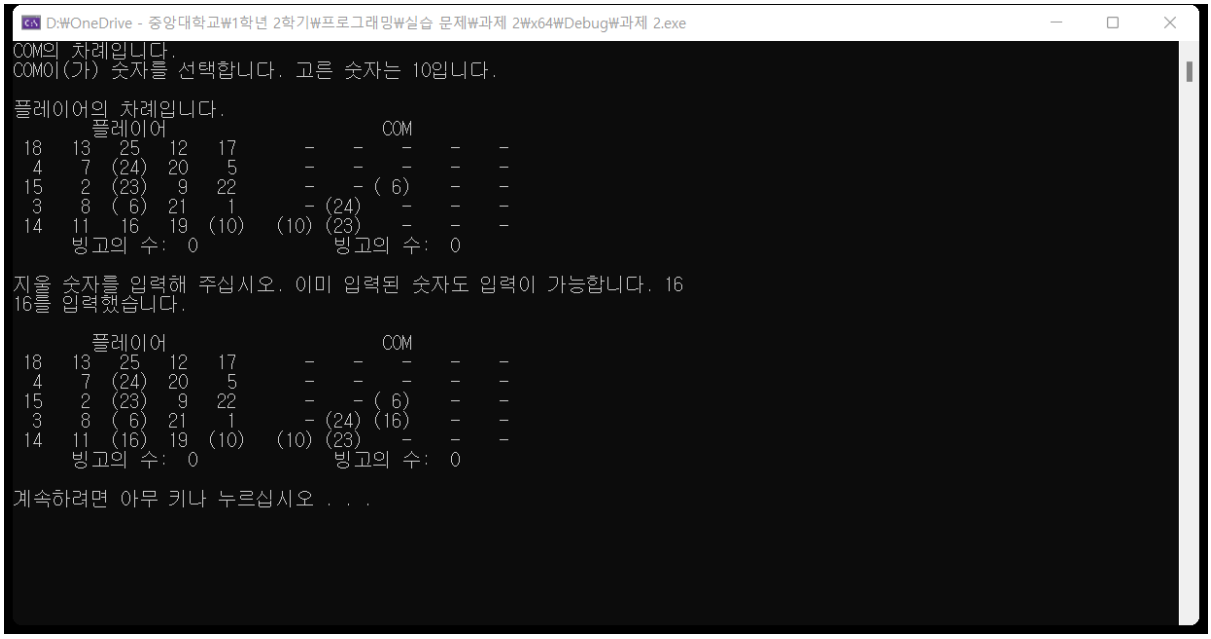

그림 18

위 그림 18은 그림 16에서 컴퓨터가 10을 고르고, 사람은 16을 입력한 상황이다. 현재 상황에서 컴퓨터의 우선도는 그림 19과 같을 것이고, 컴퓨터는 1줄의 5개 위치 중 3개가 지워진(열린) 줄의 나머지 2개를 먼저 지우기 위해, (우선도 또한 이를 노릴 수 있도록 설계되어 있다.) 주황색 배경의 위치 2곳 중 1곳의 숫자를 고르게 된다.

11	34	34	2	111
6	39	34	111	2
10	38		6	6
38			39	34
		66	34	39

그림 19

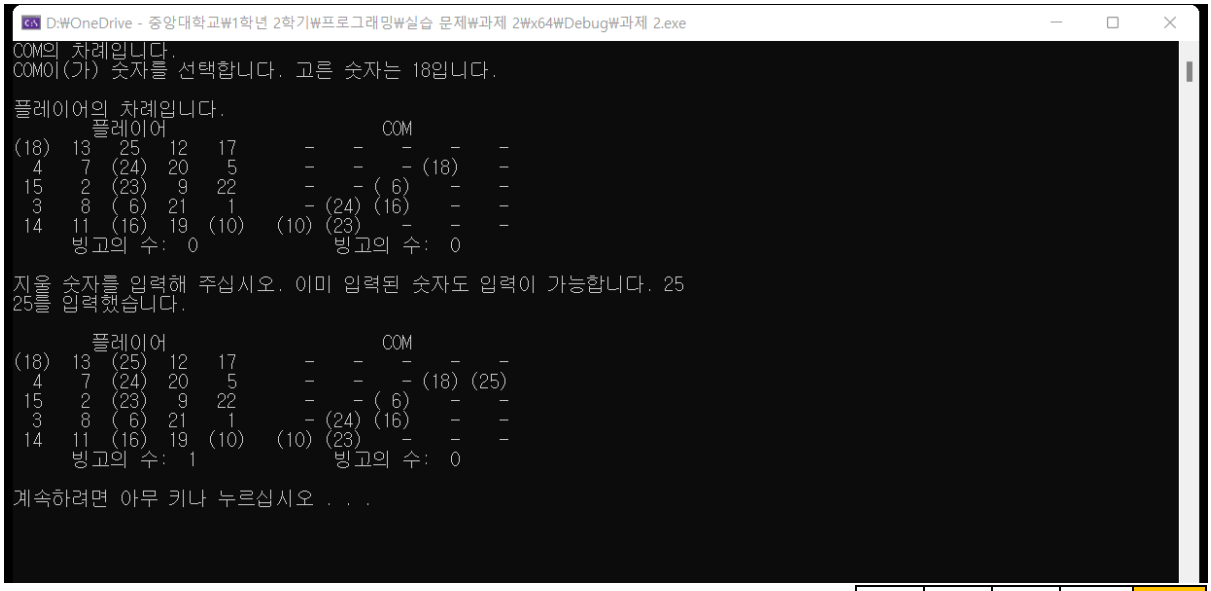

그림 20

위 그림 20은 그림 18에서 컴퓨터가 18을 고르고, 사람은 25를 입력한 상황이다. 사람이 1줄을 완성하였고 이에 따라 완성한 줄의 수가 1 증가한 점도 볼 수 있다. 현재 상황에서 컴퓨터의 우선도는 그림 21과 같을 것이고, 컴퓨터는 1줄을 완성하기 위해, 주황색 배경의 위치 1곳의 숫자를 고르게 된다.

11	34	34	6	263
38	71	66		
10	38		10	10
38			43	38
		66	38	33

그림 21

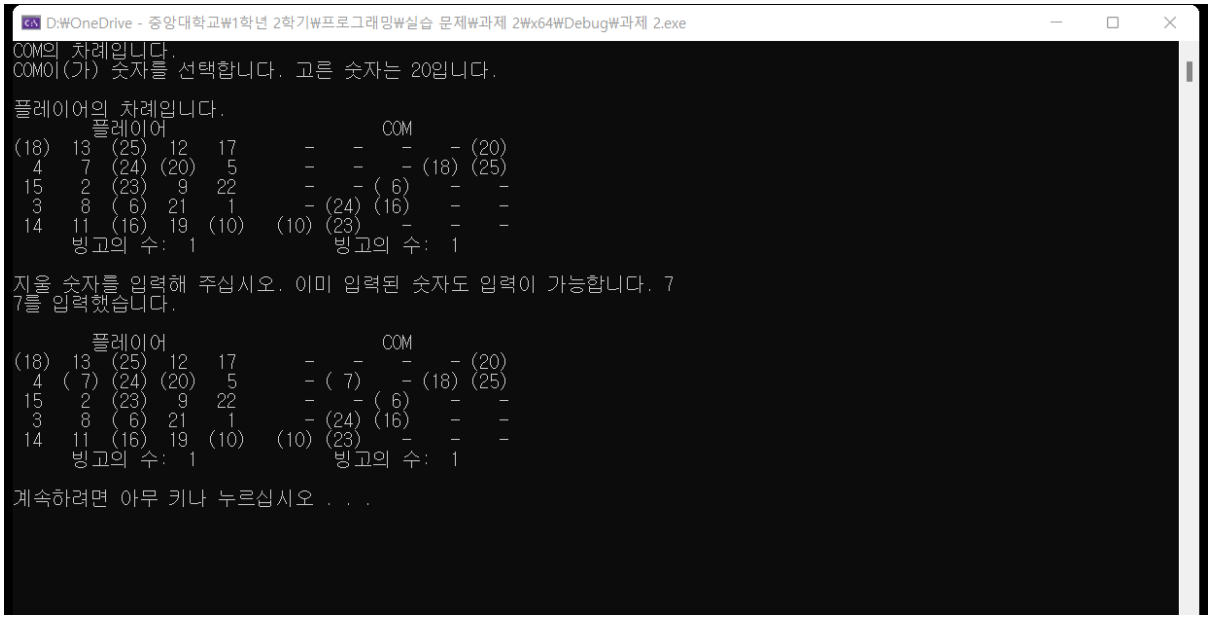

그림 22

위 그림 22는 그림 20에서 컴퓨터가 20을 고르고, 사람은 7을 입력한 상황이다. 컴퓨터도 대각선 줄 1개를 완성하여 완성한 줄의 수가 1 증가했다. 현재 상황에서 컴퓨터의 우선도는 그림 23과 같을 것이고, 컴퓨터는 주황색 배경 위치의 숫자를 고르게 된다.

43	114	38	10	
114		142		
10	114		10	38
38			71	66
		66	38	99

그림 23

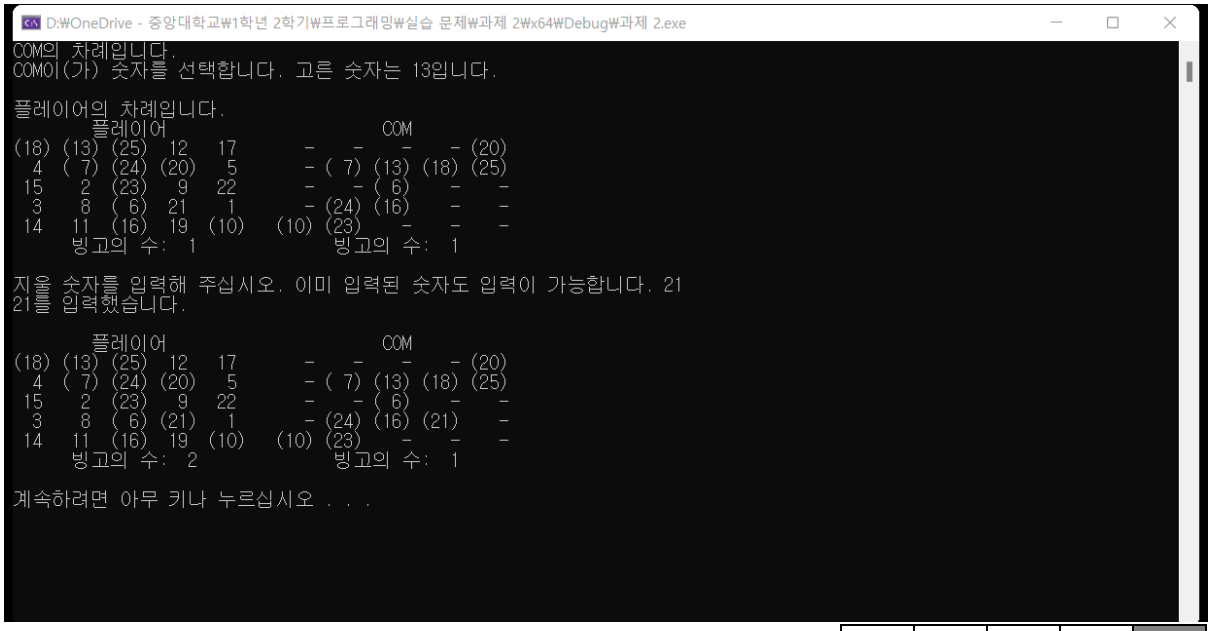

그림 24

위 그림 24는 그림 22에서 컴퓨터가 13을 고르고, 사람은 21을 입력한 상황이다. 사람이 완성한 줄의 수는 이제 2가 되었다. 현재 상황에서 컴퓨터의 우선도는 그림 25와 같을 것이고, 컴퓨터는 1줄을 완성하기 위해 주황색 배경 위치의 숫자를 고르게 된다.

119	114	114	38	
262				
10	114		38	38
114				142
		142	66	175

그림 25

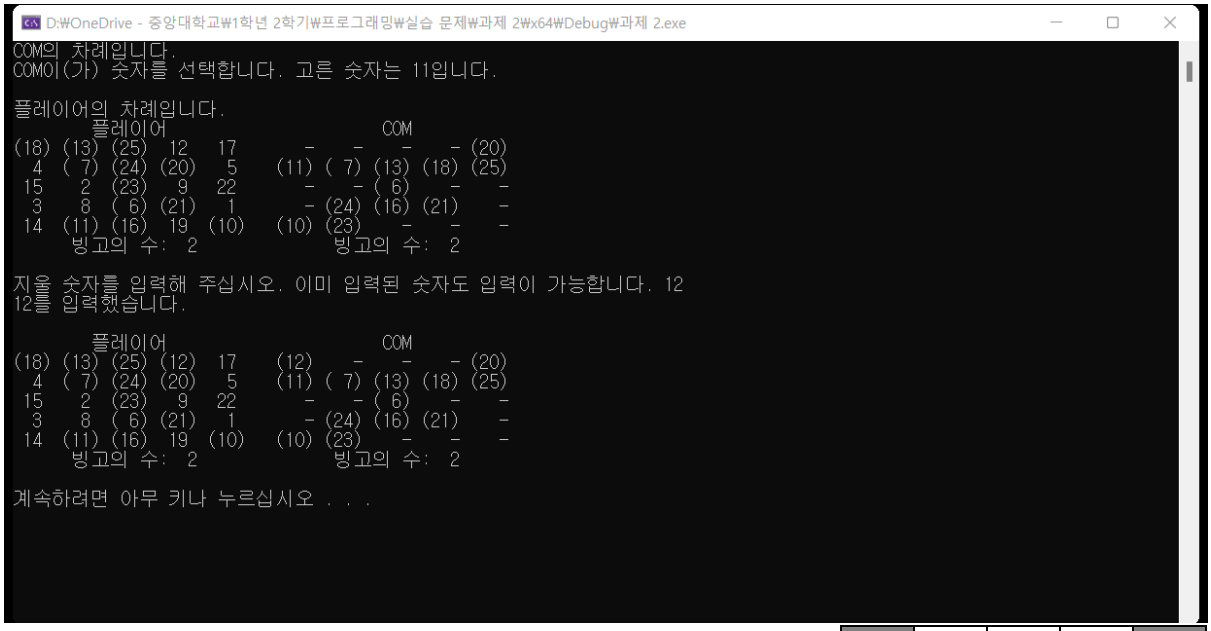

그림 26

위 그림 26은 그림 24에서 컴퓨터가 11을 고르고, 사람은 12를 입력한 상황이다. 컴퓨터가 완성한 줄 수도 이제 2가 되었다. 현재 상황에서 컴퓨터의 우선도는 그림 27과 같을 것이고, 컴퓨터는 또 1줄을 완성하기 위해 주황색 배경 위치의 숫자를 고르게 된다.

	142	142	66	
114	114		38	38
218				142
		142	66	323

그림 27

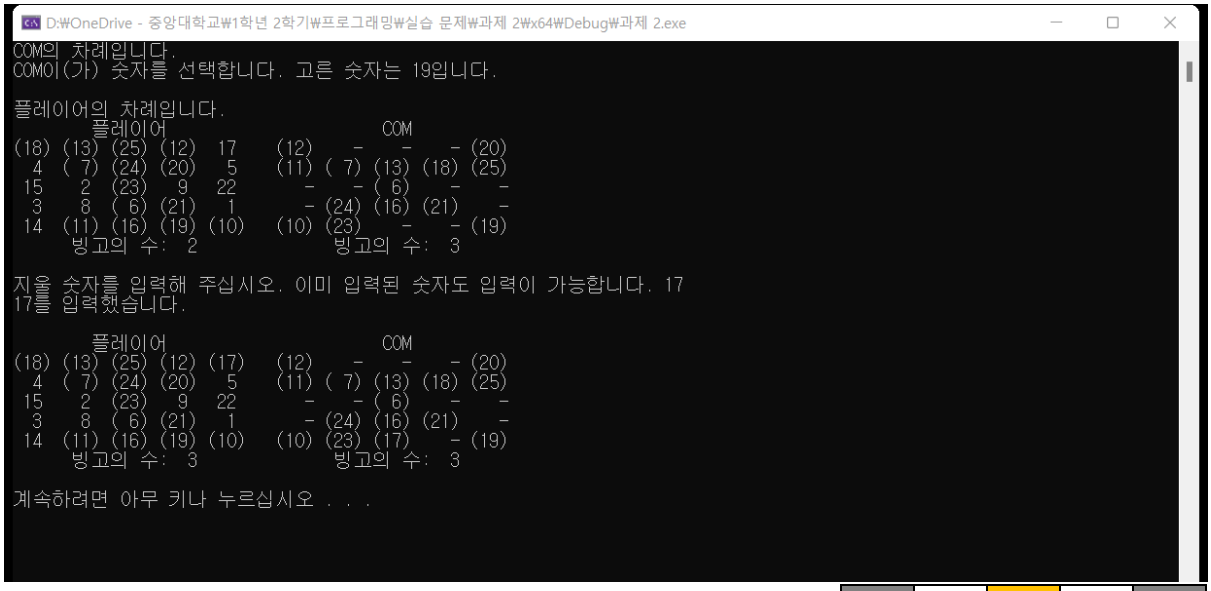

그림 28

	142	290	66	
114	114		38	114
218				218
			290	

그림 29

위 그림 28은 그림 26에서 컴퓨터가 19를 고르고, 사람은 17을 입력한 상황이다. 사람과 컴퓨터 둘 다 완성한 줄의 수는 이제 3이 되었다. 현재 상황에서 컴퓨터의 우선도는 그림 29와 같을 것이고, 컴퓨터는 또다시 1줄을 완성하기 위해 주황색 배경의 위치 2곳 중 무작위로 1곳의 숫자를 고르게 된다.

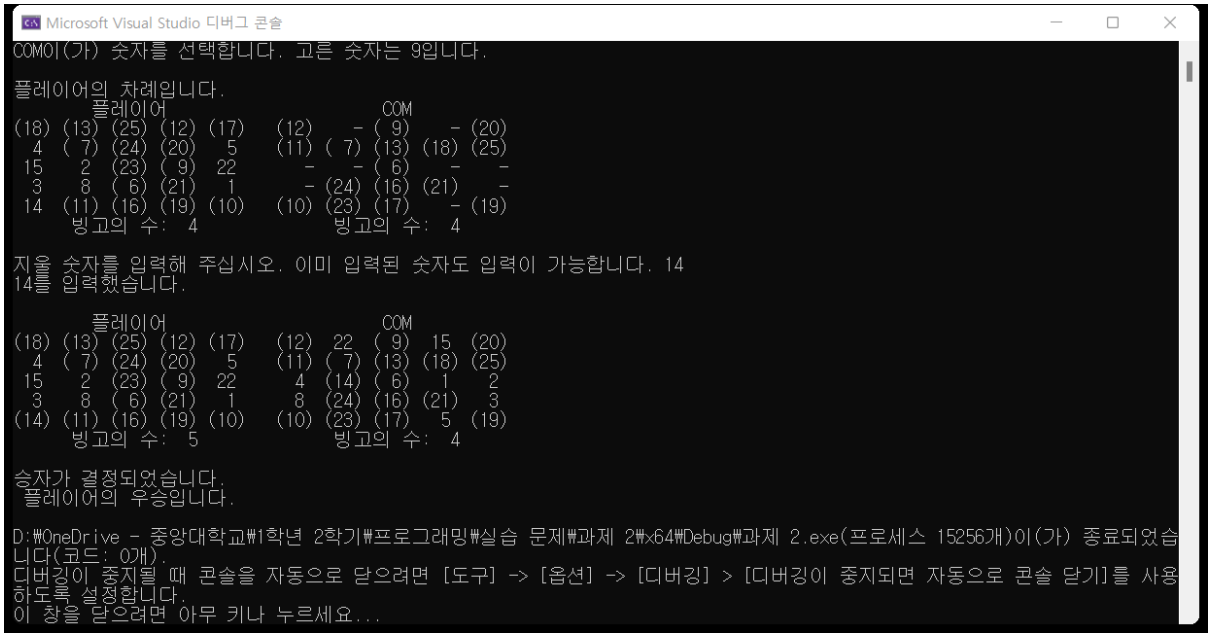

그림 30

위 그림 30은 그림 28에서 컴퓨터가 9를 고르고, 사람은 14를 입력한 상황이다. 사람이 완성한 줄의 수가 5줄이 되어 게임이 종료되고 사람의 승리가 되었다. 게임 종료 후에는 컴퓨터의 보드 전체를, 불리지(열리지) 않은 숫자이더라도 볼 수 있고, 위 캡처 사진을 보면 실제로도 이 기능이 정상 작동한다는 점을 확인할 수 있다.

프로그램을 한 번 더 실행하여 실행 결과를 하나 더 가져와 보았다.

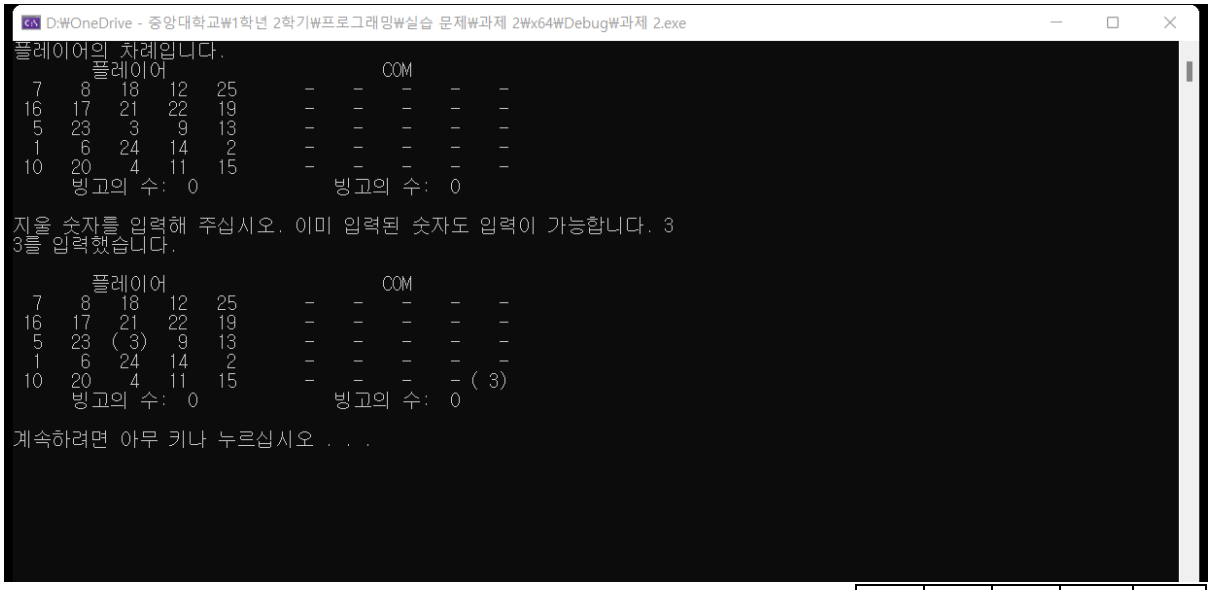

그림 31

위 그림 31은 프로그램을 실행하여 보드를 무작위로 설정하고, 사람이 3을 입력한 상황이다. 현재 상황에서 컴퓨터의 우선도는 그림 32와 같을 것이고, 컴퓨터는 보드의 정중앙인, 주황색 배경의 위치에 있는 숫자를 고르게 된다.

7	2	2	2	7
2	7	2	3	6
2	2	8	2	6
2	3	2	7	6
7	6	6	6	

그림 32

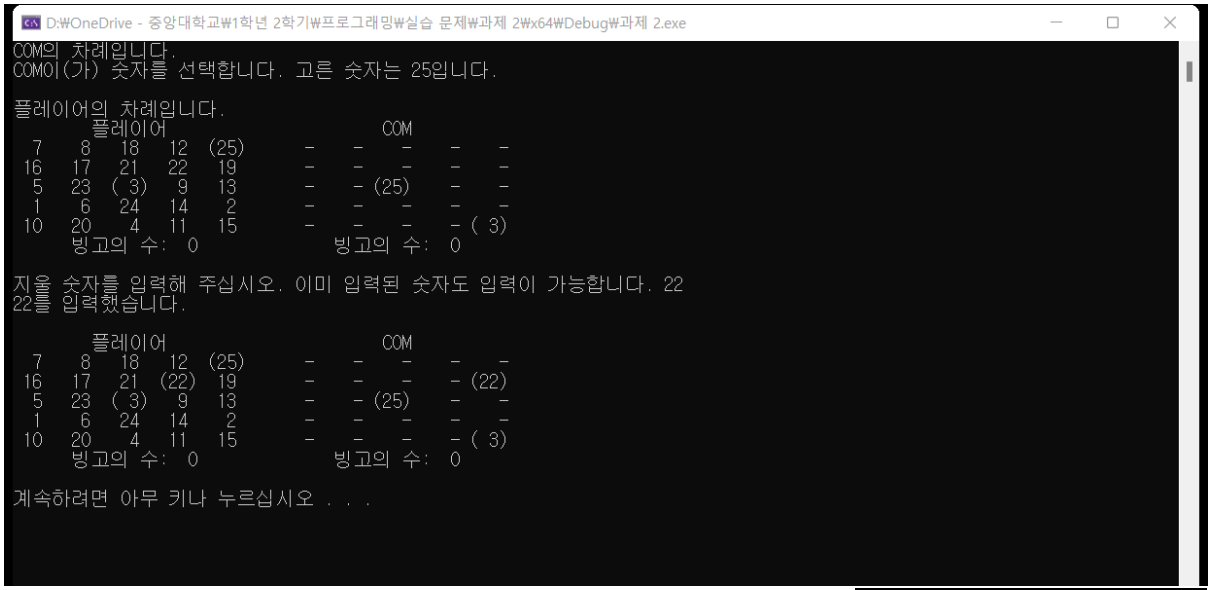

그림 33

위 그림 33은 그림 31에서 컴퓨터가 25를 고르고, 사람은 22를 입력한 상황이다. 현재 상황에서 컴퓨터의 우선도는 그림 34와 같을 것이고, 컴퓨터는 주황색 배경의 위치에 있는 숫자를 고르게 된다.

35	2	6	2	39
6	39	10	11	
6	6		6	38
2	7	6	35	34
11	6	10	6	

그림 34

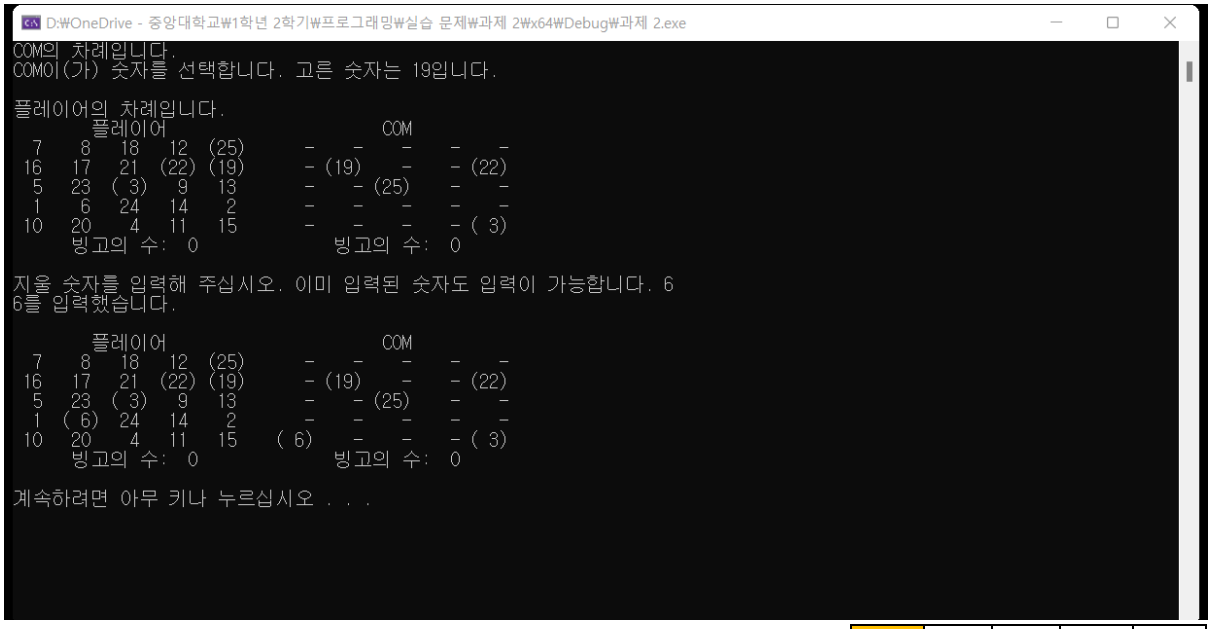

그림 35

위 그림 35는 그림 33에서 컴퓨터가 19를 고르고, 사람은 6을 입력한 상황이다. 현재 상황에서 컴퓨터의 우선도는 그림 36과 같을 것이고, 컴퓨터는 주황색 배경의 위치에 있는 숫자를 고르게 된다.

115	6	6	2	67
38		38	67	
10	10		6	38
6	39	6	111	34
	38	38	34	

그림 36

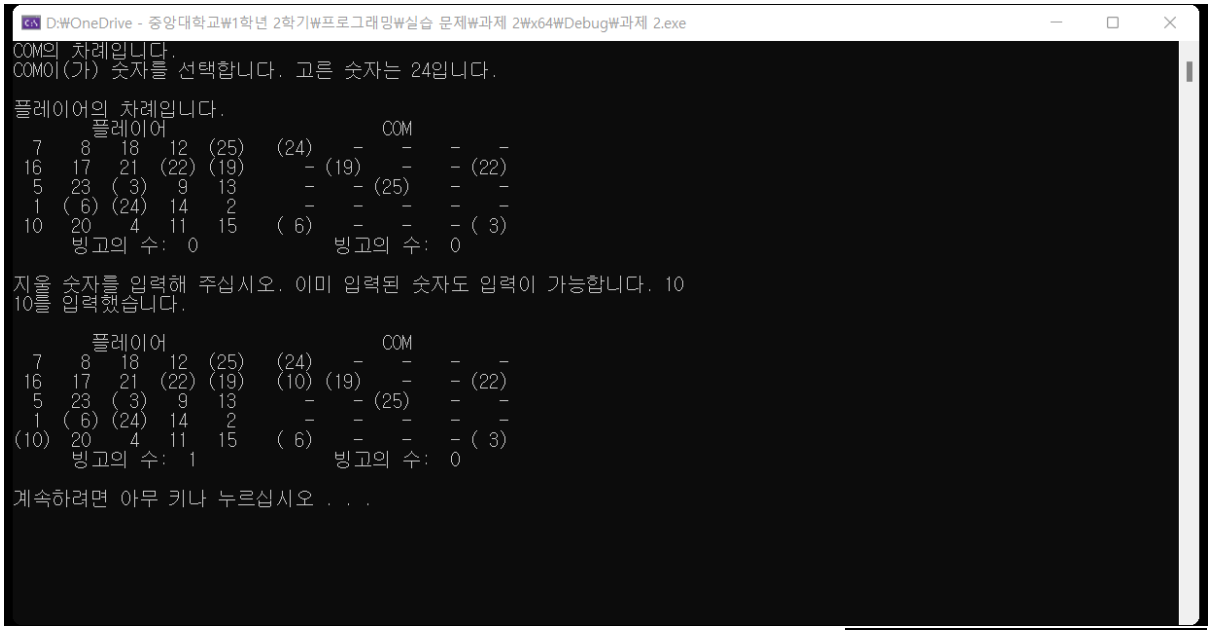

그림 37

위 그림 37은 그림 35에서 컴퓨터가 24를 고르고, 사람은 10을 입력한 상황이다. 사람은 대각선 1줄을 완성하였다. 현재 상황에서 컴퓨터의 우선도는 그림 38과 같을 것이고, 컴퓨터는 주황색 배경의 위치에 있는 숫자를 골라 1줄을 완성하게 된다.

	10	10	6	71
		114	143	
114	10		6	38
110	39	6	259	34
	38	38	34	

그림 38

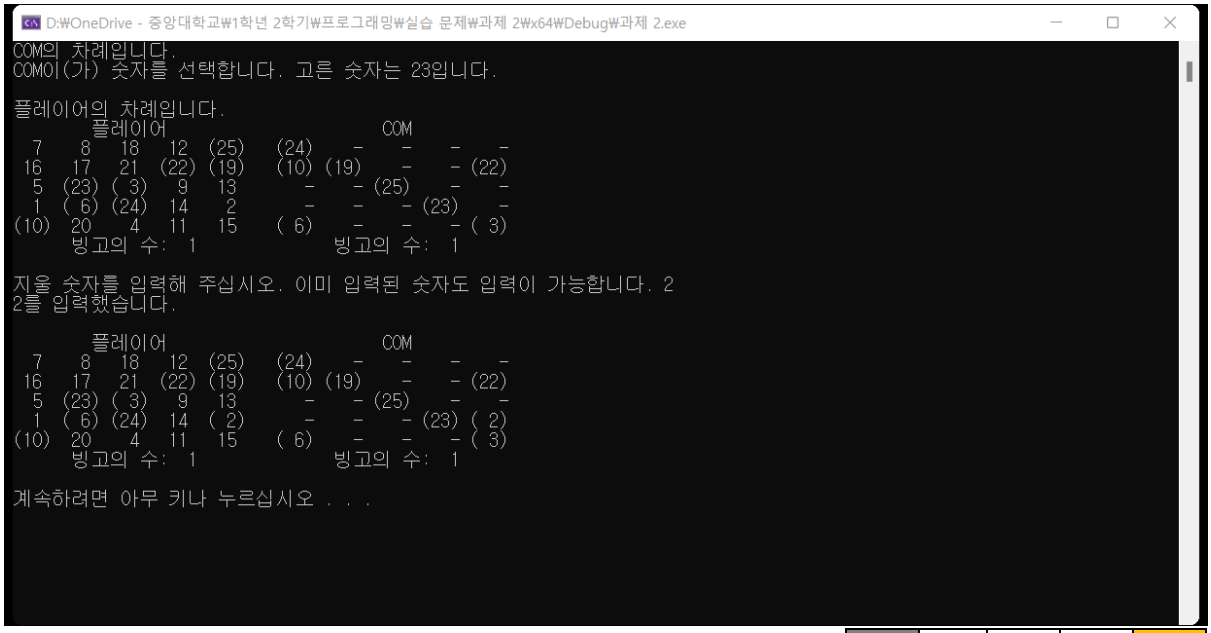

그림 39

위 그림 39는 그림 37에서 컴퓨터가 23을 고르고, 사람은 2를 입력한 상황이다. 컴퓨터도 이어 1줄을 완성하였다. 현재 상황에서 컴퓨터의 우선도는 그림 40과 같을 것이고, 컴퓨터는 주황색 배경의 위치 2곳 중 1곳에 있는 숫자를 고르게 된다.

	10	10	10	147
		114	147	
114	10		10	114
142	71	38		
	38	38	38	

그림 40

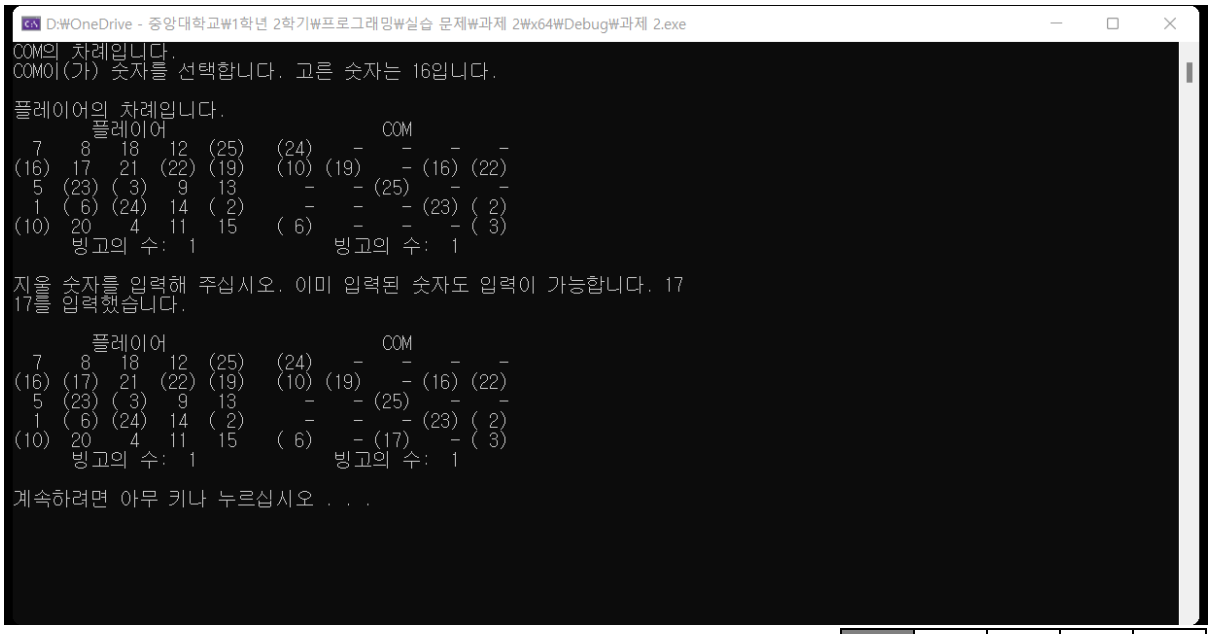

그림 41

위 그림 41은 그림 39에서 컴퓨터가 16을 고르고, 사람은 17을 입력한 상황이다. 현재 상황에서 컴퓨터의 우선도는 그림 42와 같을 것이고, 컴퓨터는 주황색 배경의 위치에 있는 숫자를 골라 2번째 줄을 완성하게 된다.

	10	38	38	223
		290		
114	10		38	114
142	147	66		
	114		142	

그림 42

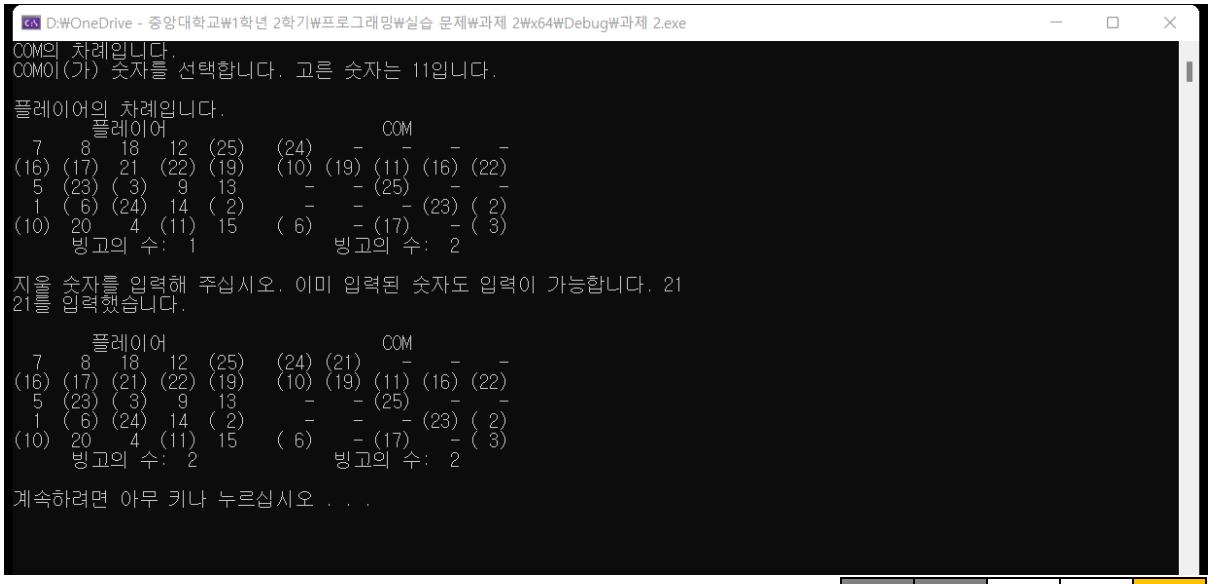

그림 43

위 그림 43은 그림 41에서 컴퓨터가 11을 고르고, 사람은 21을 입력한 상황이다. 컴퓨터가 먼저 2번째 줄을 완성했고, 사람도 이어 2번째 줄을 완성했다. 현재 상황에서 컴퓨터의 우선도는 그림 44와 같을 것이고, 컴퓨터는 여러 줄 완성을 동시에 노릴 수 있는, 주황색 배경의 위치에 있는 숫자를 고르게 된다.

		142	66	251
114	38		38	114
142	175	142		
	142		142	

그림 44

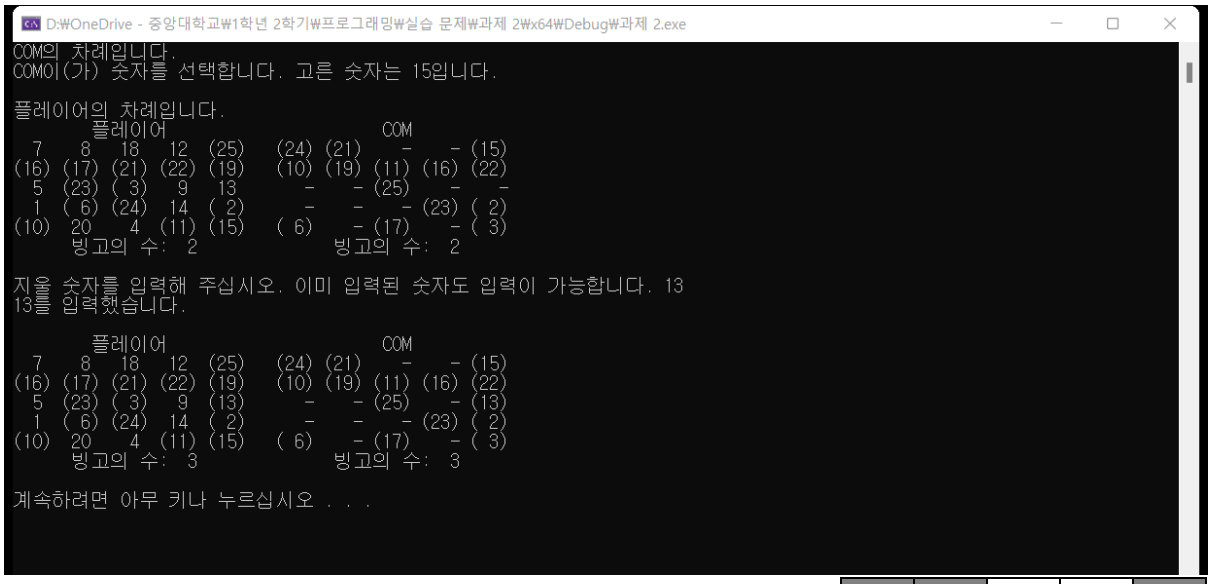

그림 45

위 그림 45는 그림 43에서 컴퓨터가 15를 고르고, 사람은 13을 입력한 상황이다. 사람이 입력한 13으로 사람과 컴퓨터 모두 3번째 줄을 완성하였다. 현재 상황에서 컴퓨터의 우선도는 그림 46과 같을 것이고, 컴퓨터는 4번째 줄을 완성하기 위해 주황색 배경의 위치에 있는 숫자를 고르게 된다.

		218	142	
142	66		66	
142	323	142		
	142		142	

그림 46

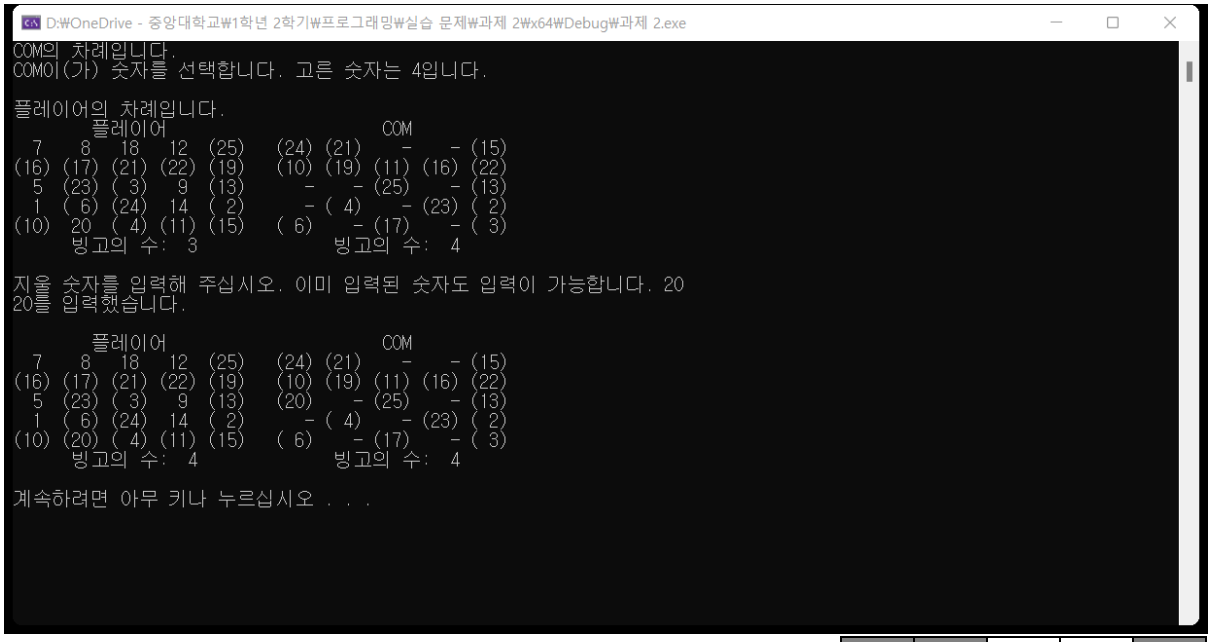

그림 47

위 그림 47은 그림 45에서 컴퓨터가 4를 고르고, 사람은 20을 입력한 상황이다. 컴퓨터가 먼저 4번째 줄을 완성했고, 사람도 이어 4번째 줄을 완성했다. 현재 상황에서 컴퓨터의 우선도는 그림 48과 같을 것이고, 컴퓨터는 주황색 배경의 위치에 있는 숫자를 고르고 5번째 줄을 완성해 아래 그림 49와 같이 컴퓨터의 승리로 게임이 끝나게 된다.

		218	142	
	218		142	
366		218		
	218		142	

그림 48

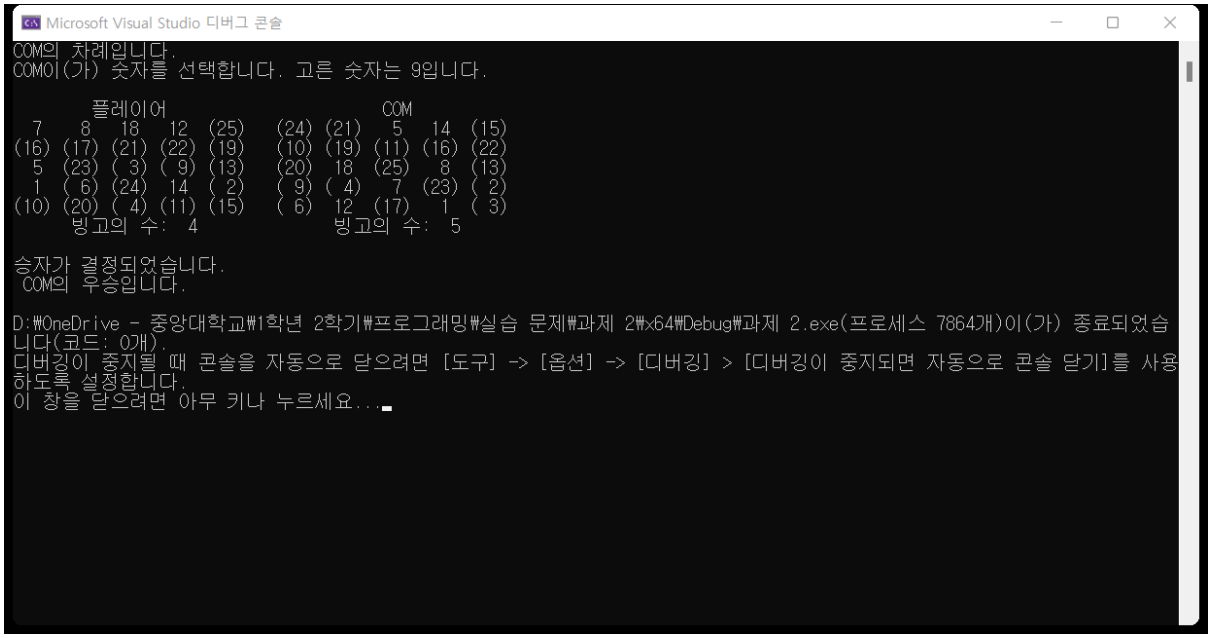

그림 49

보드의 크기를 바꾸거나, 승리에 필요한 줄의 수를 바꾸거나, 참가자 수나 각 플레이어가 숫자를 선택하는 방법 변경도 코드 앞부분에 #define에 정의된 숫자들과 main 함수를 변경함으로써 할

수 있고, 이를 통해 보드의 크기가 다르거나, 승리에 필요한 줄의 수가 다르거나, 3인 이상이 빙고 게임을 하는 것도 가능하지만, 문제 2는 보드의 크기는 5*5, 승리에 필요한 줄의 수는 5, 사람과 컴퓨터가 1대 1로 게임을 하는 것만을 요구하였기에, 개인적으로만 테스트해 보고 실행 결과는 제외했다.

8. 소스코드

첨부파일 참조