CSED101. Programming & Problem solving
Fall 2022

Programming Assignment #1
(75 points)

이민종 (minjong.lee@postech.ac.kr)

■ **제출 마감일: 2022.10.31 23:59**

■ **개발 환경: Windows Visual Studio 2019**

■ **제출물**

- C 소스 코드 (assn1.c)
  - ➢ 프로그램의 소스 코드에 채점자의 이해를 돕기 위한 주석을 반드시 붙여주세요.
- 보고서 파일 (.docx, .hwp 또는 .pdf; assn1.docx, assn1.hwp 또는 assn1.pdf)
  - ➢ 보고서는 AssnReadMe.pdf를 참조하여 작성하시면 됩니다.
  - ➢ 명예 서약 (Honor code): 표지에 다음의 서약을 기입하여 제출해 주세요: "나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다." 보고서 표지에 명예 서약이 기입되어 있지 않은 과제는 제출되지 않은 것으로 처리됩니다.
  - ➢ 작성한 소스 코드와 보고서 파일은 PLMS를 통해 제출해 주세요.

■ **주의 사항**

- 컴파일이나 실행이 되지 않는 과제는 0점으로 채점됩니다.
- 제출 기한보다 하루 늦게 제출된 과제는 최종 20%, 이틀 늦게 제출된 과제는 최종 40% 감점됩니다. 제출 기한보다 사흘 이상 늦으면 제출 받지 않습니다 (0점 처리).
- 각 문제의 제한 조건과 요구 사항을 반드시 지켜 주시기 바랍니다.
- 모든 문제의 출력 형식은 채점을 위해 아래에 제시된 예시들과 최대한 비슷하게 작성해 주세요.
- 부정행위에 관한 규정은 POSTECH 전자컴퓨터공학부 학부위원회의 "POSTECH 전자컴퓨터공학부 부정행위 정의"를 따릅니다 (PLMS의 본 과목 공지사항에 등록된 글 중, 제목이 [document about cheating]인 글에 첨부되어 있는 disciplinary.pdf를 참조하세요).
- 이번 과제는 추가 기능 구현과 관련된 추가 점수가 따로 없습니다.

■ Problem: 인디언 홀덤

(목적)

이번 과제를 통하여 조건문, 반복문, 사용자 정의 함수 및 라이브러리 함수 사용법을 익힙니다.

(주의사항)

1. 이번 과제는 함수를 정의하고 사용하는 방법을 익히는 문제이므로 main() 함수에 모든 기능을 구현한 경우 감점 처리 합니다.
2. 문서에 반드시 정의해서 사용해야 할 사용자 정의 함수가 설명되어 있으니 확인 후 구현하도록 합니다. 이 때, 설명에서 지정한 사용자 정의 함수의 매개변수의 개수와 자료형, 함수 이름, 반환 자료형 등은 자유롭게 변경이 가능합니다. 그러나 동일한 기능을 하는 함수는 반드시 있어야 하며, 변경 시 무엇을 어떻게 변경해서 구현했는지 보고서에 기록하도록 합니다. 이외에 필요한 함수는 정의해서 사용할 수 있습니다.
3. 프로그램 구현 시, main() 함수를 호출을 직접 하지 않습니다. 즉, 소스 코드 내에 main(); 이라고 호출하지 않습니다.
4. 전역 변수, 배열 및 goto 문은 사용할 수 없으며, 포인터의 경우 수업시간에 다룬 내용에 한해서 사용이 가능합니다.
5. 프로그램에서 랜덤 시드는 프로그램 시작 시 main() 에서 srand(time(NULL)); 함수를 한 번만 호출하도록 하여 한번만 초기화 합니다.
6. 사용자 입력에서 숫자를 입력 받는 부분에는 숫자만 입력하는 것으로 가정합니다. 즉, 숫자 입력 받는 부분에는 문자 등의 입력에 대해서는 고려할 필요가 없습니다.
7. 명시된 에러 처리 외에는 고려하지 않아도 됩니다.
8. 문제의 출력 형식은 채점을 위해 아래의 실행 예시와 최대한 비슷하게 작성해 주세요.
9. 프로그램 실제 실행 예시를 본 문서 12~14쪽에서 확인할 수 있으니, 구현 전 확인하시기 바랍니다.

[인디언 홀덤]

인디언 홀덤은 받은 카드와 공유 카드의 조합이 높은 플레이어가 승리하는 게임으로, 인디언 포커처럼 상대의 카드는 볼 수 있지만 자신의 카드는 볼 수 없는 상태에서 게임이 진행됩니다.
자세한 룰은 https://www.youtube.com/watch?v=vcfTvU7wkL4 에서 확인해보기를 권장합니다. 구현할 프로그램의 기본적인 룰은 영상과 동일하며, 달라진 점은 굵은 글씨로 표시했습니다.

[상세 설명]

- 게임은 유저와 컴퓨터, 총 2 명의 플레이어로 진행되며, **최대 10 회의 게임**이 진행된다. 시작 시, 각 플레이어는 **50 개**의 칩을 받으며, 모든 게임이 진행되거나 한 플레이어가 칩을 모두 잃으면 프로그램이 종료된다.
- 게임은 1 부터 10 까지의 숫자 카드 4 세트, 총 40 장의 카드로 진행된다.
- 각 플레이어의 카드 1 장씩과 공유 카드 2 장, 총 4 장의 카드가 랜덤하게 생성되고, 각 플레이어의 카드는 상대 플레이어에게만 공개된다.
- 베팅 시작 시, 두 플레이어는 각각 칩을 1 개씩 내고, 이전 게임에서 진 플레이어부터 번갈아 가면서 베팅을 진행한다 (첫 게임은 유저부터 시작).
- 베팅 단계가 종료될 시, 각 플레이어의 핸드를 이용하여 승자를 결정한다.
  즉, 받은 카드와 공유 카드의 조합을 비교하여 승자를 결정한다.
- 승리한 플레이어는 해당 게임에서 베팅된 칩을 모두 갖고, 다음 게임을 진행한다.

[베팅 종류]

- **Call**: 상대 플레이어가 낸 칩과 동일한 개수의 칩을 베팅하고 베팅 단계를 종료한다 (해당 게임의 첫번째 베팅일 경우 (i.e. 첫번째 턴일 경우) Call 을 할 수 없다).
- **Raise**: 상대가 낸 칩의 개수를 받아들이고, 거기에 추가로 더 베팅한다.
- **Fold**: 현재 상태에서 해당 게임을 포기한다 (게임 패배).

[조합 (핸드)]

- **Double**: 플레이어의 카드와 공유 카드 1 장이 동일한 값을 가졌을 때 (ex) 2, (2, 5))
- **Straight**: 세 개의 숫자 카드가 연속된 값을 가질 때 (ex) 3, 4, 5)
- **Triple**: 세 개의 숫자 카드가 모두 같은 값을 가졌을 때 (ex) 4, 4, 4)
- **No pair**: 어떤 조합에도 해당되지 않을 때 (ex) 5, 2, 9)
- Triple, Straight, Double, No pair 순으로 높은 핸드로 처리한다.
- 더 높은 핸드를 갖고 있는 플레이어가 해당 게임을 승리하며, 같은 핸드를 갖고 있는 경우, 개인 카드의 숫자가 더 높은 플레이어가 승리한다.
- **같은 핸드를 갖고 있으면서 개인 카드의 숫자도 같을 경우, 유저가 승리한다.**

[구현 요구 사항]

[1. 프로그램 세팅]

- 각 플레이어의 칩의 개수를 50 개로 설정한다.
- 최대 게임 횟수를 10 회로 설정한다.
- ‘2. 게임 세팅’ 부터 ‘4. 베팅 종료(게임 종료)’를 프로그램 종료 상태를 만족하기 전까지 반복하도록 구현한다.

[2. 게임 세팅]

프로그램을 실행하면, 그림 1 과 같이 게임을 위해 세팅 된 후, 출력된다.

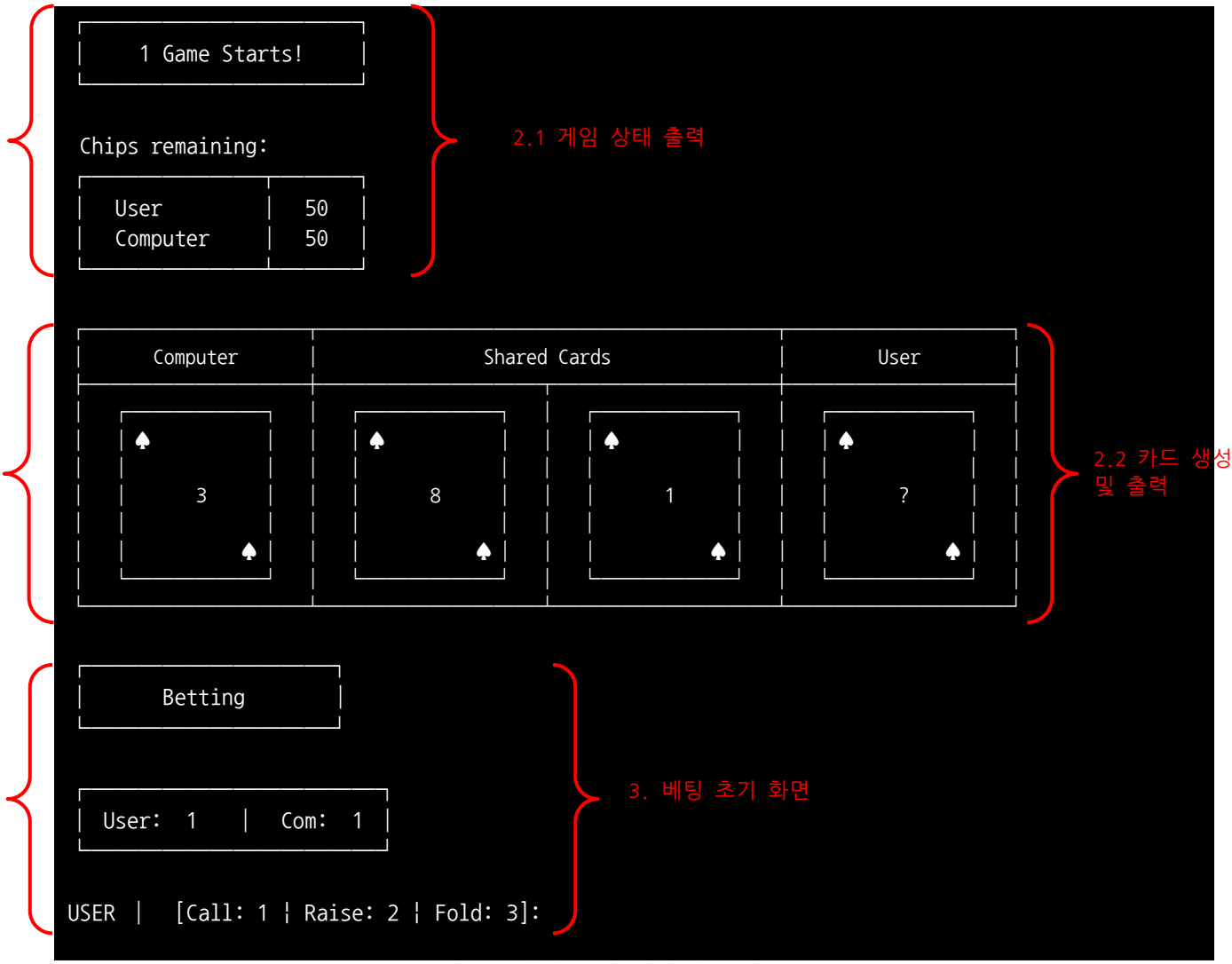


그림 1. 게임 초기 화면의 구성

2.1 게임 상태 출력

- 현재 게임이 몇 번째 게임인지 출력한다. (예. 1 Game Starts!)
- 각 플레이어의 현재 칩의 개수 출력한다.
- 사용자 정의 함수
  - **void print_game_status(int round, int user_chips, int com_chips)**: 현재 몇 번째 게임인지와 각 플레이어의 현재 보유 칩 수를 출력하는 함수

2.2 카드 생성 및 출력

공유 카드 2 장과 각 플레이어의 카드 1 장씩을 랜덤하게 생성하고 카드 정보를 출력하도록 한다.

- 카드 생성은 rand 함수를 이용한다.
- 매 게임마다 1부터 10까지의 숫자 카드 4세트, 총 40장의 카드에서 4장의 카드가 선택되도록 한다.
- 컴퓨터 카드, 공유 카드 2 장, 유저 카드를 순서대로 출력한다.
  - 카드 출력은 **Tips-카드 출력(마지막 쪽)**을 참고하여 구현한다.
  - 유저를 기준으로 출력되는 화면이므로 컴퓨터 카드와 공유 카드 2 장은 숫자를 출력한다. 유저는 자신의 카드를 확인하지 않은 채 진행하게 되므로 물음표(?)로 나타낸다.
- 사용자 정의 함수
  - **void card_shuffle(int *shared_card1, int *shared_card2, int *user_card, int *computer_card)**: 4 개의 카드를 생성하는 함수
  - **void print_card_info(int shared_card1, int shared_card2, int user_card, int computer_card)**: 생성된 카드를 출력하는 함수

[3. 베팅]

3.1 베팅 전체 프로세스

- [2. 게임 세팅] 이후, 그림 2 와 같이 베팅 시작 문구가 출력되고 베팅이 시작된다.
- 베팅 시작 시, 두 플레이어는 기본 베팅으로 각각 칩을 1 개씩 내고, 이전 게임에서 진 플레이어부터 번갈아 가면서 베팅을 진행한다 (첫 게임은 유저부터 시작).
  [예외처리]
  - 한 플레이어가 칩을 1 개 갖고 있다면, 베팅 단계를 건너뛰고 바로 게임 결과를 확인한다.
- 각 플레이어는 자신의 순서에서 Call, Raise, Fold 를 선택할 수 있다. (단, 해당 게임의 첫 턴인 경우 Call 을 할 수 없다.)
  - Raise 가 선택될 경우, 현재 베팅된 칩의 개수를 업데이트 하고, 상대 플레이어의 베팅이 진행된다.
  - Call 이나 Fold 가 선택될 경우, 베팅은 그 즉시 종료된다.

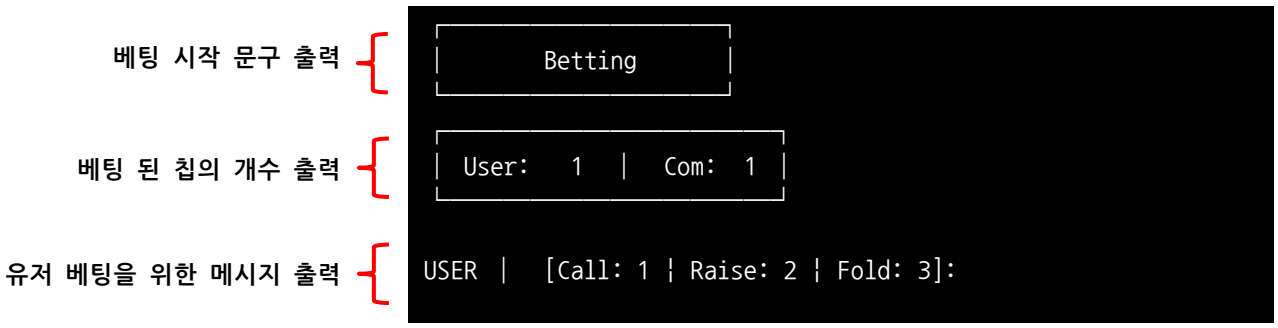


그림 2. 베팅 시작 화면

그림 2 는 베팅 시작할 때의 화면으로, 각 플레이어가 기본 베팅으로 칩 1 개씩을 베팅한 상태를 확인할 수 있다. 그림 2 는 유저부터 베팅을 시작했기 때문에, 유저의 베팅을 위한 메시지를 출력 후, 유저의 선택을 기다리고 있음을 볼 수 있다.

3.2 유저의 베팅

유저는 자신의 턴에서 베팅(Call, Raise, Fold)을 선택한다.

- Call 은 1, Raise 는 2, Fold 는 3 으로, 유저의 선택을 입력 받는다.
  [예외 처리]
  - 첫 턴에서 Call 을 선택한 경우와 1, 2, 3 이외의 입력을 한 경우에는 아래와 같이 에러 메시지 출력 후, 유효한 입력을 받을 때까지 반복하여 다시 입력 받는다. is_valid_num() 함수를 사용하여 구현한다.
- Raise 를 선택한 경우, 추가로 낼 칩의 개수를 입력을 받는다.
  - (1) 0 또는 음수나 (2) 갖고 있는 칩보다 베팅할 칩의 수를 더 많이 입력하는 경우는 없다고 가정한다.
- 이전 턴에서 컴퓨터가 유저가 갖고 있는 칩의 개수와 같거나 더 많은 칩을 Raise 한 경우, 유저는 Call 또는 Fold 만 선택하여 입력한다고 가정한다.
  [예외 처리]
  - 이때 Call 을 선택한 경우, 갖고 있는 칩을 모두 베팅한다.
  - 예시: 유저가 칩을 10 개 갖고 있을 때, 컴퓨터가 12 개를 베팅했다면, Call 또는 Fold 만 입력할 수 있고, Call 의 경우, 10 개를 베팅하고 베팅 단계를 종료한다.
- 각 플레이어의 베팅이 끝날 때마다 아래처럼 베팅된 칩의 개수를 출력하도록 한다.
- 사용자 정의 함수
  - **int is_valid_num(int a, int b, int num)**: 전달받은 숫자(num)가 a 이상 b 이하의 범위(a ≤ num ≤ b)에 해당하면 1 을, 아니면 0 을 반환한다.
  - **int user_turn(int user_chips, int *user_betting_chips, int betted_chips, int turn)**:
    - 유저의 베팅 턴을 수행하는 함수로 Raise 를 선택한 경우 추가로 낼 칩의 개수를 입력 받아 반환, Call 을 선택한 경우 0, Fold 를 선택한 경우 -1 을 반환한다.
    - 몇 번째 베팅 순서(turn)인지와 이전 턴에서 상대가 베팅한 칩수(betted_chips)를 매개변수로 전달받아 유저의 베팅 턴을 구현한다.
    - 유저의 베팅 선택에 따라 유저의 베팅된 칩의 개수를 업데이트한다.

(예시의 빨간색 밑줄은 사용자 입력에 해당)

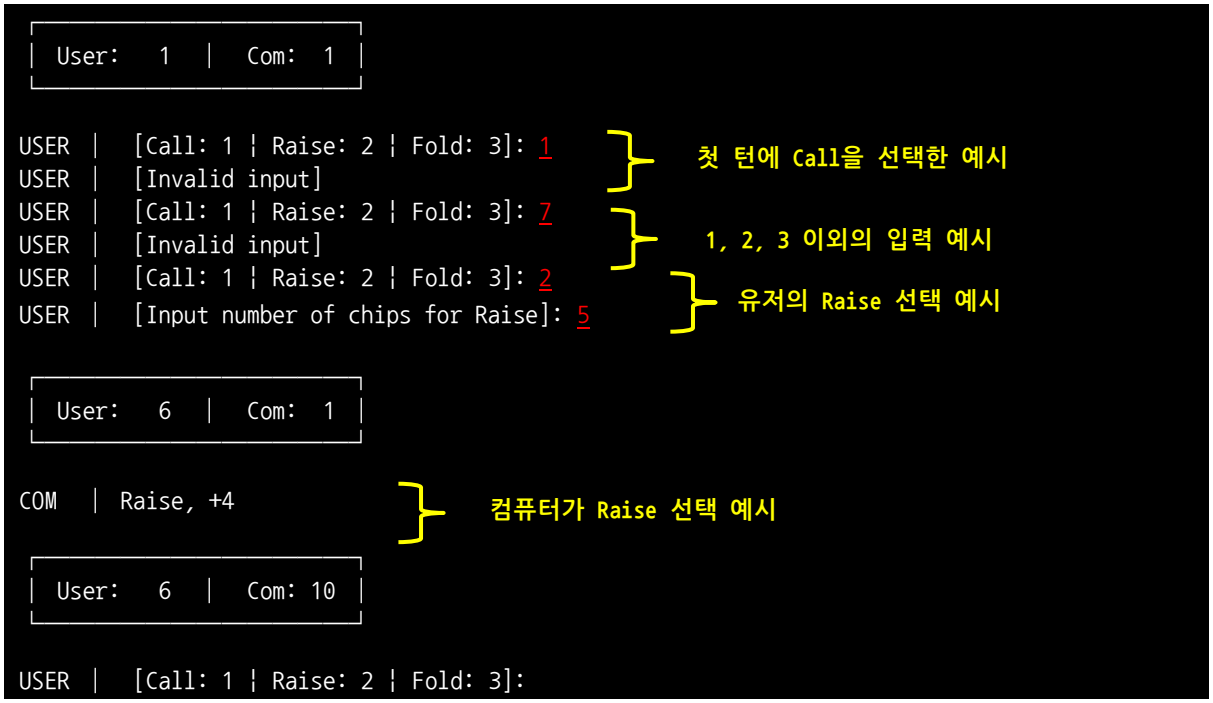


그림 3. 베팅 예시

3.3 컴퓨터의 베팅

컴퓨터는 아래의 규칙에 따라 베팅하도록 구현한다.

- **[경우 1]** 만약 상대의 패가 Double, Straight 혹은 Triple 을 만족할 때,
  [1-1] 70%의 확률로 Fold 를 한다.
  [1-2] 30%의 확률로 Call 을 한다.
  첫 턴일 경우일 경우에는 Call 을 할 수 없으므로 1 개의 칩을 Raise 한다.
- **[경우 2]** 경우 1 이 아닐 때,
  [2-1] 첫 턴일 경우, 1 개의 칩을 Raise 한다.
  [2-2] 50%의 확률로 Call 을 한다.
  [2-3] 50%의 확률로 Raise 를 한다.
  추가로 베팅할 칩의 개수는 1 ~ 5 개 중 랜덤하게 선택한다.
- 확률은 rand 함수를 사용하여 구현한다.

[예외 처리]

- Call 을 하고자 할 때, 갖고 있는 칩보다 더 많은 칩을 베팅하고자 할 경우, 현재 갖고 있는 칩을 모두 베팅한다.
- Raise 를 하고자 할 때, 갖고 있는 칩보다 더 많은 칩을 베팅하고자 할 경우, 현재 갖고 있는 칩을 모두 베팅한다. 이때 베팅은 Call 로 처리한다 (즉, 베팅 단계를 종료한다).

컴퓨터의 베팅 결과는 그림 4 와 같이, Call(Fold)을 선택할 경우, Call(Fold)을 출력한다. Raise 를 선택할 경우, Raise 와 함께 추가로 베팅한 칩의 개수를 출력한다.

```
COM   |   Call
```

```
COM   |   Raise, +5
```

그림 4. Call 을 한 경우와 5 개의 칩을 Raise 한 경우

- 사용자 정의 함수
  - **int calc_hand(int card, int shard_card1, int shard_card2):**
    공유 카드와 상대방의 카드를 비교하여 카드 조합(Double, Triple 등)에 따라 적절한 숫자를 반환한다.
  - **int computer_turn(int user_hand, int com_chips, int *com_betting_chips, int betted_chips, int turn):**
    - 컴퓨터의 베팅 턴을 수행하는 함수로, 유저의 카드의 조합(user_hand)에 따라 위 규칙에 따라 베팅을 선택하도록 한다. Raise 가 선택된 경우 추가로 낼 칩의 개수를 반환, Call 을 선택한 경우 0, Fold 를 선택한 경우 -1 을 반환한다.
    - 몇 번째 베팅 순서(turn)인지와 이전 턴에서 상대가 베팅한 칩수(betted_chips)를 매개변수로 전달받아 컴퓨터의 베팅 턴을 구현한다.
    - 컴퓨터의 베팅 선택에 따라 컴퓨터가 베팅한 칩의 개수를 업데이트 한다.

[4. 베팅 종료(게임 종료)]

각 플레이어는 자신의 베팅 순서에서 Call 또는 Fold 를 선택하여 베팅을 종료할 수 있다. 베팅 종료는 1 회에 해당하는 카드 게임이 종료됨을 의미하며 해당 게임의 승자를 결정하여 출력하게 된다.

4.1 베팅 종료 메시지와 유저의 카드 출력

- 베팅이 종료되면 베팅이 끝났다는 메시지(Betting Finished)와 함께 모든 카드를 출력한다.
- 그림 5 를 보면 물음표로 출력되던 User 카드의 숫자가 출력됨을 볼 수 있다.

4.2 승리 플레이어 판별 및 출력

- 해당 게임의 승리 플레이어를 판별하여 출력한다.
- Call 을 선택하여 베팅 종료된 경우
  - 두 플레이어의 핸드를 계산한다.
  - 3 쪽의 '조합(핸드)' 설명을 바탕으로 승리 플레이어를 판별한다.
- Fold 를 선택하여 베팅 종료된 경우
  - Fold를 선택한 플레이어가 게임을 포기한 경우로, 해당 게임의 패자로 처리한다.

4.3 플레이어의 칩 개수 업데이트 및 출력

- 게임을 승리한 플레이어가 베팅된 칩을 모두 갖도록 두 플레이어의 칩을 업데이트한다.
- 그림 5 와 같이 업데이트된 두 플레이어의 칩을 출력한다.

4.4 게임 진행 여부 입력 및 처리

- 유저에게 게임을 계속 진행하고 싶은지를 물어 유저의 선택에 따라 프로그램을 종료하거나 다음 게임을 진행하게 된다.
- 유저가 1 을 입력한 경우, ‘2. 게임 세팅’부터 다시 시작한다. 즉, 두 플레이어는 현재 보유한 칩을 가지고, 새로운 4 장의 카드를 분배 받아 게임에서 이번 게임에서 진 플레이어부터 베팅을 시작하여 새로운 게임을 계속 진행하게 된다.
  이 때, 현재 화면을 초기화하고 다음 게임을 시작하도록 한다. 새로운 게임화면 출력 예시는 그림 6 의 두번째 게임 화면의 예시를 참고하도록 한다.
  - 화면을 초기화하는 방법은 **Tips-화면 지우기**(마지막 쪽)을 참고하여 구현한다.
- 유저가 -1 을 입력한 경우, 프로그램 종료를 위해 아래 [5. 프로그램 종료]로 넘어간다.
- 유저 입력으로 -1 과 1 외의 입력은 없다고 가정한다.

[예외 처리]

- 10 번째 게임이 종료되거나 한 플레이어가 칩이 없을 경우, 사용자 입력과 상관없이 무조건 [5. 프로그램 종료]로 넘어가도록 구현한다.

- 사용자 정의 함수
  - **int calc_hand(int card, int shard_card1, int shard_card2):**
    공유 카드와 상대방의 카드를 비교하여 카드 조합(Double, Triple 등)에 따라 적절한 숫자를 반환
  - **calc_winner(int shared_card1, int shared_card2, int user_card, int computer_card):**
    Call 을 선택하여 베팅 종료된 경우, 두 플레이어의 조합을 계산하여 승자를 결정한다. 적절히 user 와 computer 를 나타내는 값을 반환한다.

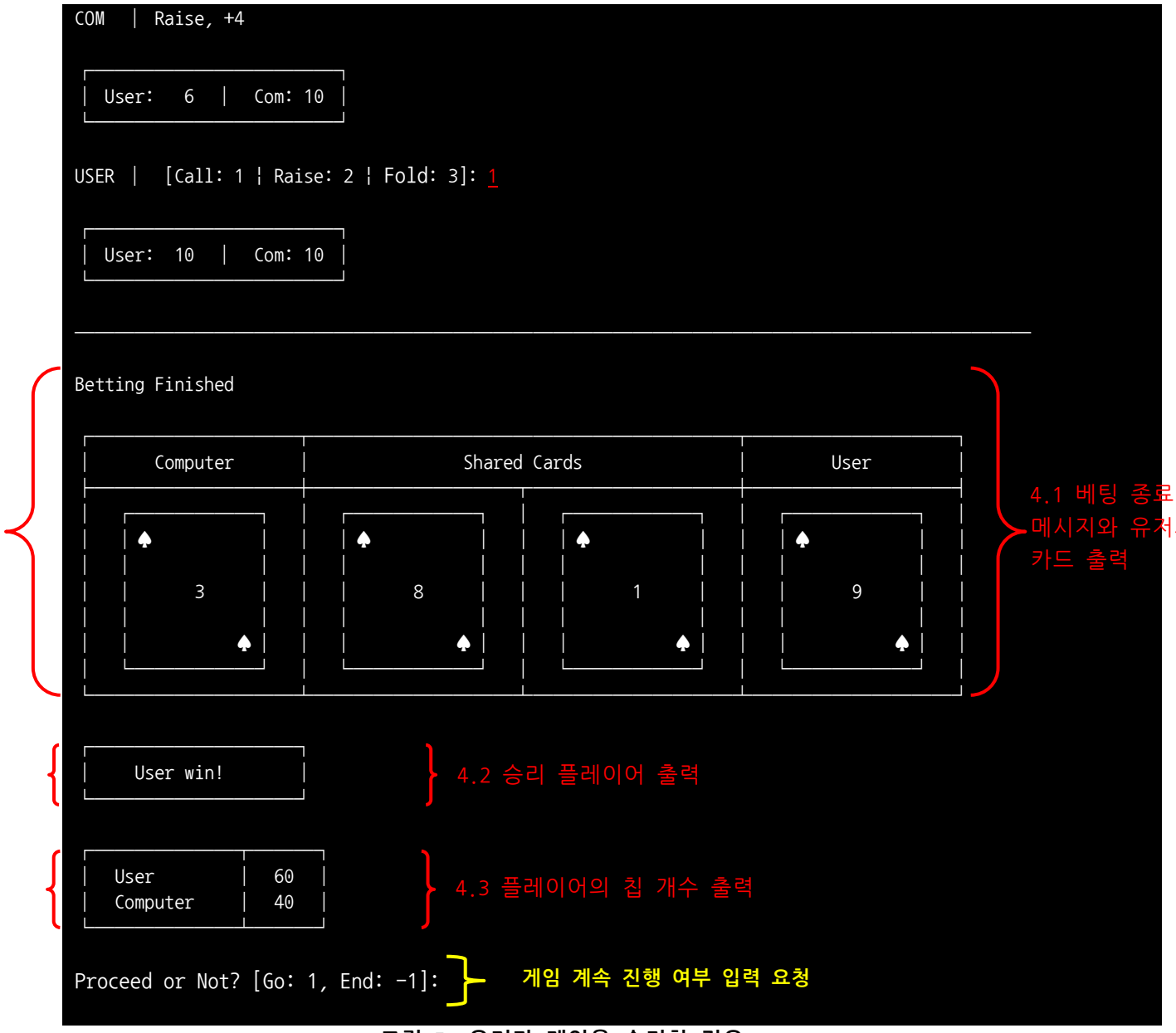


그림 5. 유저가 게임을 승리한 경우

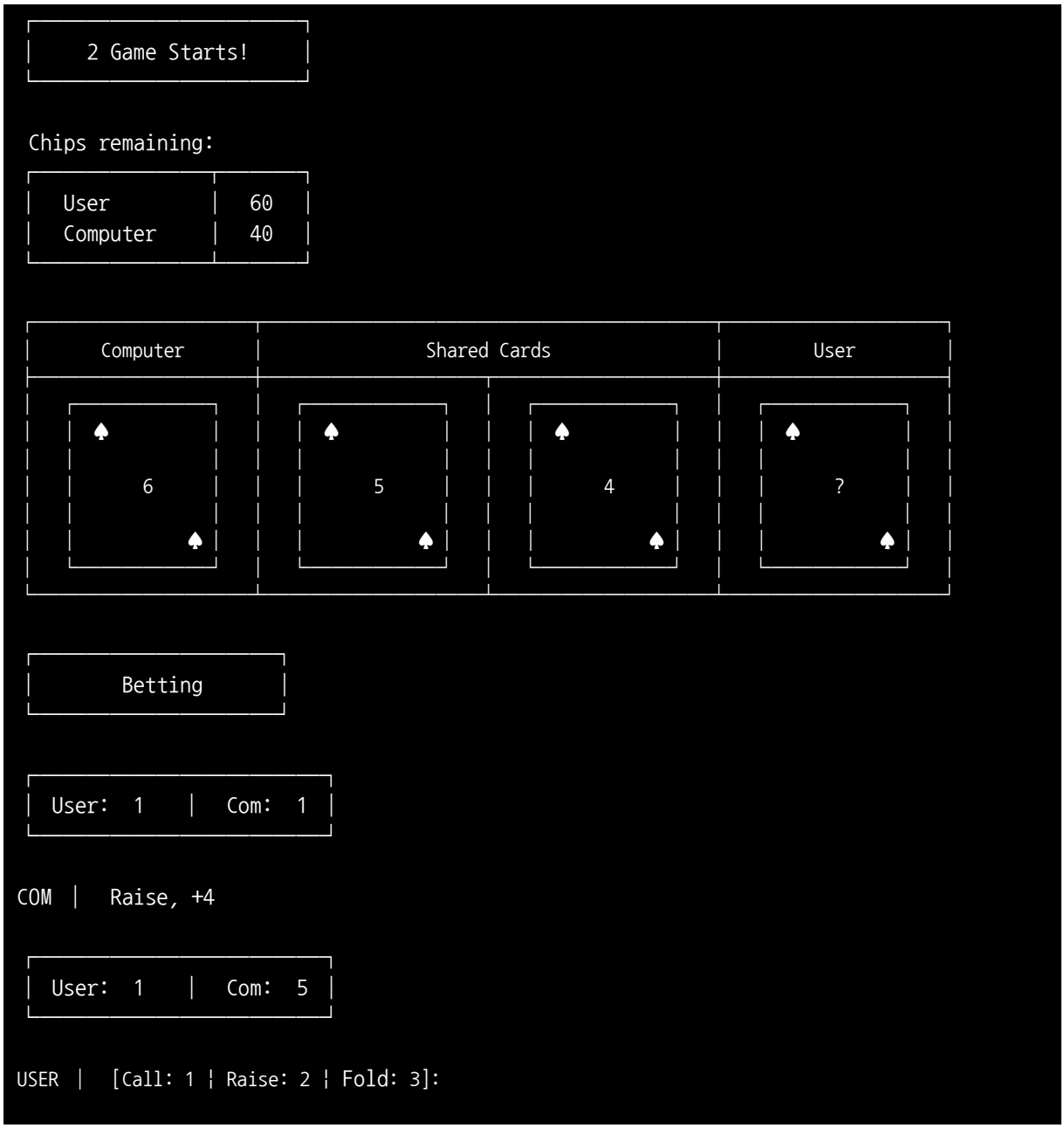


그림 6. 두 번째 게임 진행 예시

그림 6은 그림 5에서 유저가 1을 입력하여 게임을 계속 진행을 선택한 경우의 예시에 해당한다.

- 2번째 게임이 진행된다는 메시지를 출력한다 (2 Game Starts!).
- 이전 게임의 결과로 보유한 칩을 출력한다(User: 60, Computer: 40).
- 새로운 게임을 위해 카드를 다시 분배 받은 후, 카드를 출력한다.
- 이전 게임에서 진 플레이어인 컴퓨터가 베팅을 먼저 시작한다.

[5. 프로그램 종료]

- (1) 한 플레이어의 칩이 없거나 (2) 10 회의 게임을 모두 진행했거나 (3) 유저가 -1 을 입력하여 게임 진행을 포기한 경우, 그림 7 과 같이 출력하고 프로그램을 종료한다.
- 종료 화면 구성
  - 카드 게임 진행 횟수(최대 10 회) 출력
  - 최종 보유 칩 수 출력
  - 최종 승리한 플레이어 출력
    칩이 더 많은 플레이어가 승리한다. 칩의 개수가 같을 경우 유저가 승리한다.
    유저가 승리한 경우 User win!으로, 컴퓨터가 승리한 경우 Computer win!으로 출력

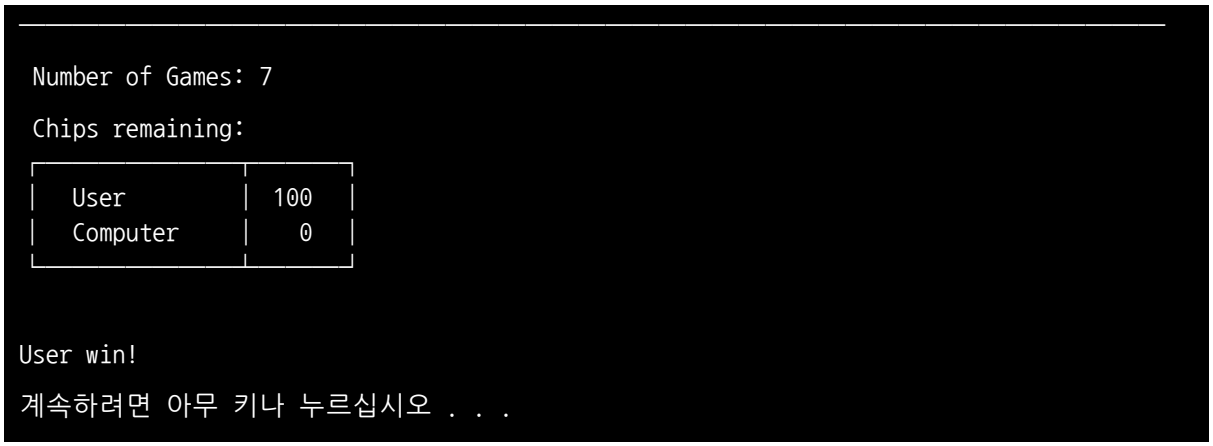
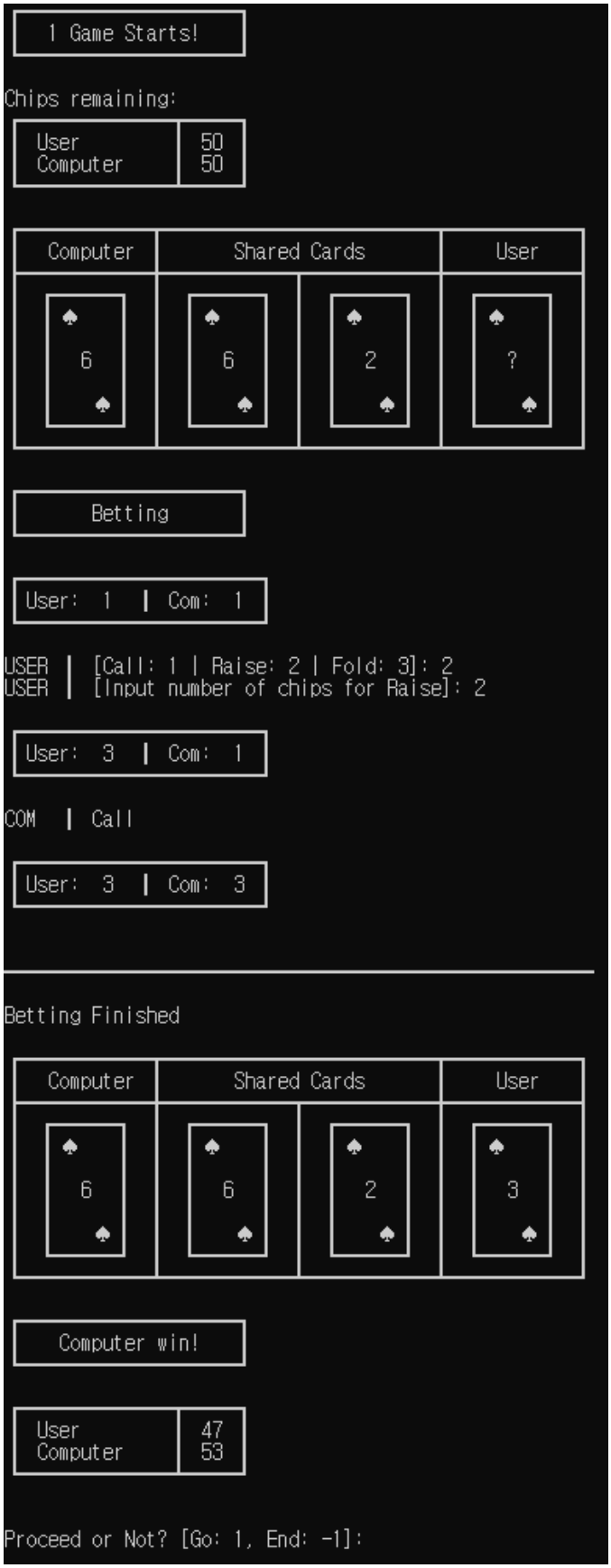


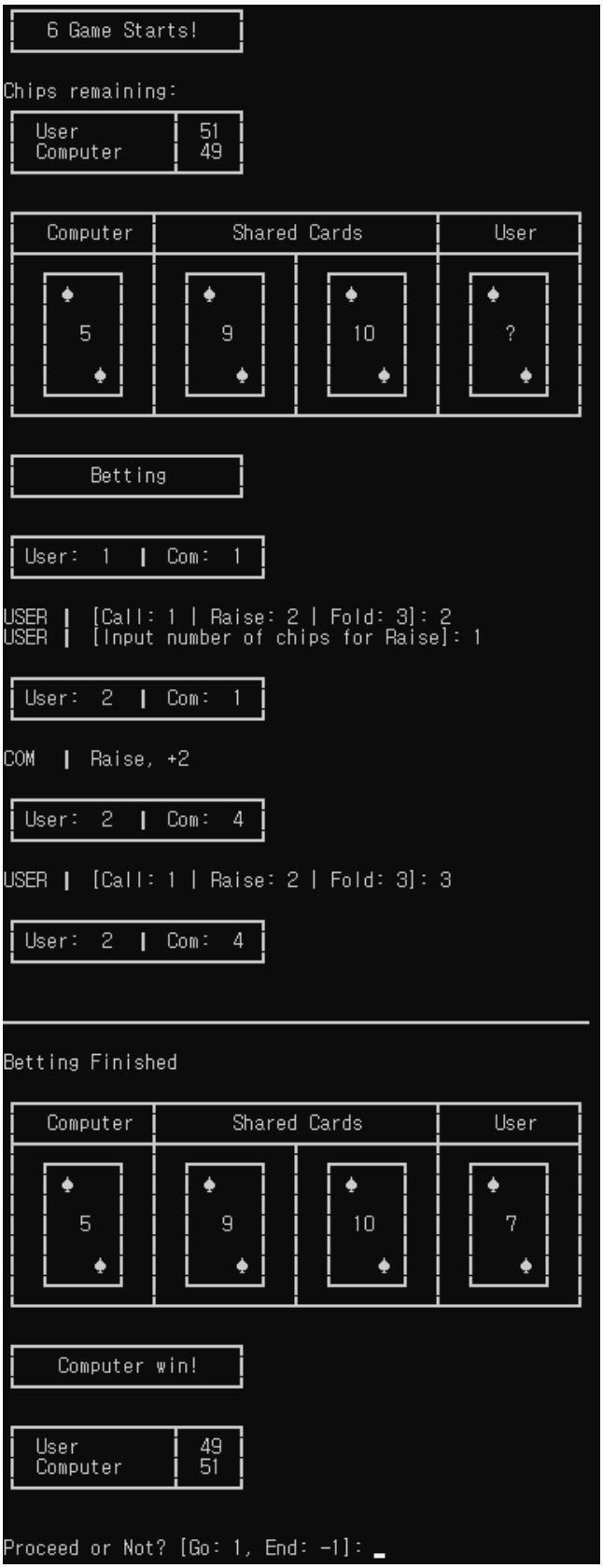
그림 7. 유저가 최종 승리한 경우

[프로그램 실행 예시]

한 게임의 전체 프로세스의 예시

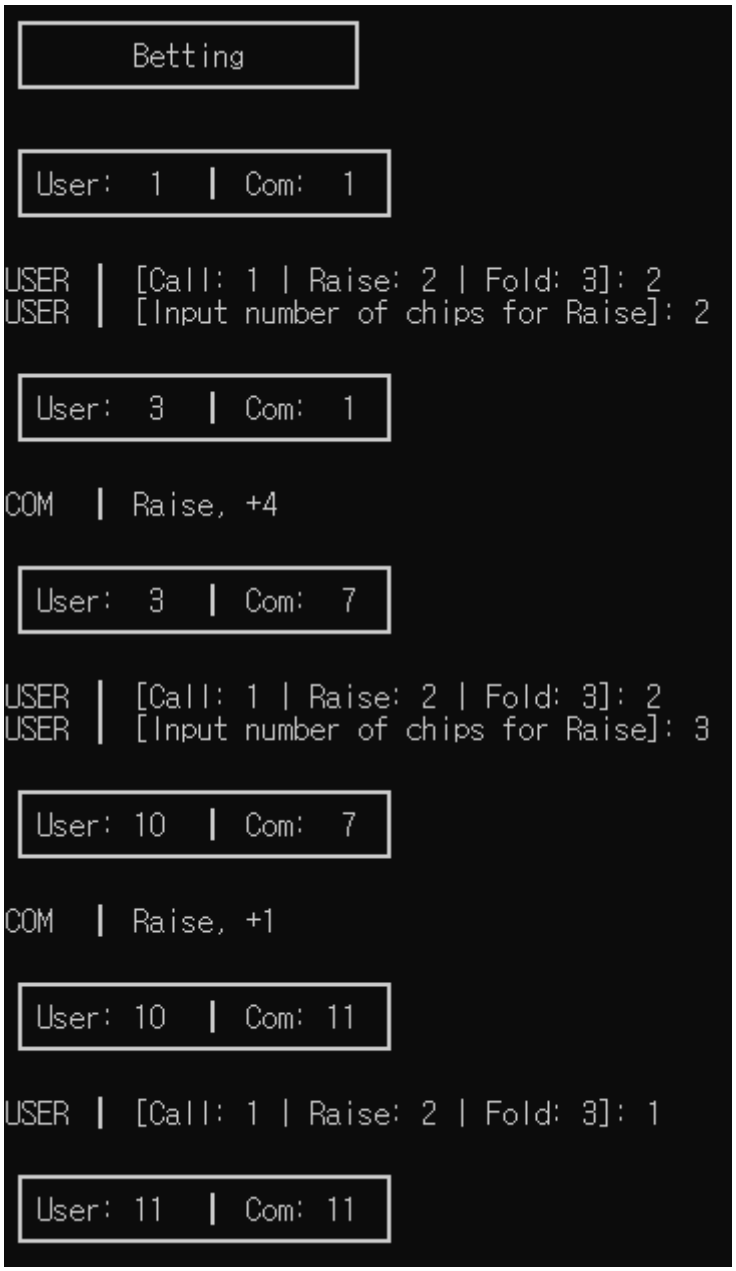


Fold 가 발생하는 경우



- **유저의 카드 조합이 더 좋지만, 유저가 Fold 를 했기 때문에, 컴퓨터가 승리했고, 베팅된 칩도 유저가 베팅한 2 개의 칩만 컴퓨터에게 전달된 것을 확인할 수 있다.**

베팅 전체 프로세스의 예시



- **베팅된 칩의 현황은 계속 출력되어야 합니다.**

[Tips]

1) 박스 그리기

- 박스를 그릴 때는 https://en.wikipedia.org/wiki/Box-drawing_character 의 표를 참고하여 사용할 수 있습니다.
- 다른 포맷의 출력 방식을 사용하거나 다른 문자를 사용하는 것도 허용됩니다.

2) Define 사용 권장

```
#define NOPAIR 1
#define DOUBLE 2
#define STRAIGHT 3
#define TRIPLE 4

void example() {
        int user_hand = TRIPLE;
        int computer_hand = DOUBLE;
        if (user_hand > computer_hand) { // True
                // codes
        }
}
```

- Define 을 사용하여 위의 코드와 같이 직관적으로 조합을 비교 및 응용할 수 있습니다.
- 그 외의 기능을 구현할 때도 적절히 사용하기를 권장합니다.

3) 카드 출력

- 카드 결과 출력을 구현할 때 복사 및 변형해서 사용하세요.

```
printf("┌──────────────┬──────────────┬──────────────┬──────────────┐\n");
printf("│   Computer   │ Shared Cards │     User     │\n");
printf("├──────────────┼──────────────┼──────────────┼──────────────┤\n");
printf("│  ┌────────┐  │  ┌────────┐  │  ┌────────┐  │  ┌────────┐  │\n");
printf("│  │ ♠      │  │  │ ♠      │  │  │ ♠      │  │  │ ♠      │  │\n");
printf("│  │        │  │  │        │  │  │        │  │  │        │  │\n");
printf("│  │   1    │  │  │   1    │  │  │   1    │  │  │   1    │  │\n");
printf("│  │        │  │  │        │  │  │        │  │  │        │  │\n");
printf("│  │      ♠ │  │  │      ♠ │  │  │      ♠ │  │  │      ♠ │  │\n");
printf("│  └────────┘  │  └────────┘  │  └────────┘  │  └────────┘  │\n");
printf("└──────────────┴──────────────┴──────────────┴──────────────┘\n");
```

4) 화면 지우기

- 화면을 지우고 싶을 경우, stdlib 헤더 파일을 포함시킨 후 system() 함수를 사용할 수 있습니다. system() 함수의 매개변수로 "cls"를 넘겨주면 됩니다. 아래의 소스 코드를 컴파일하여 실행하면 콘솔 화면에 "Erase me!" 문자열 이 출력됩니다. Enter 키를 입력하면 화면이 지워진 후에 "Erased" 문자열이 출력되는 것을 볼 수 있습니다.

```
#include <stdio.h>
#include <stdlib.h> // system() 사용을 위해 포함
int main() {
    char c;
    printf("Erase me!\n");
    scanf("%c", &c);
    system("cls"); // 리눅스 또는 맥의 경우 system("clear");
    printf("Erased\n");
    return 0;
}
```

5) 예외 처리

- 구현 요구 사항에서 언급하지 않은 예외 상황이 더 존재합니다. 앞서 언급했듯이, 요구되지 않은 사항은 처리하지 않아도 됩니다.