# OOP Term Project

## Go-Stop Game Programing

02분반 03조
김민규(201902660), 박형기(201902696), 이예성(201902727)

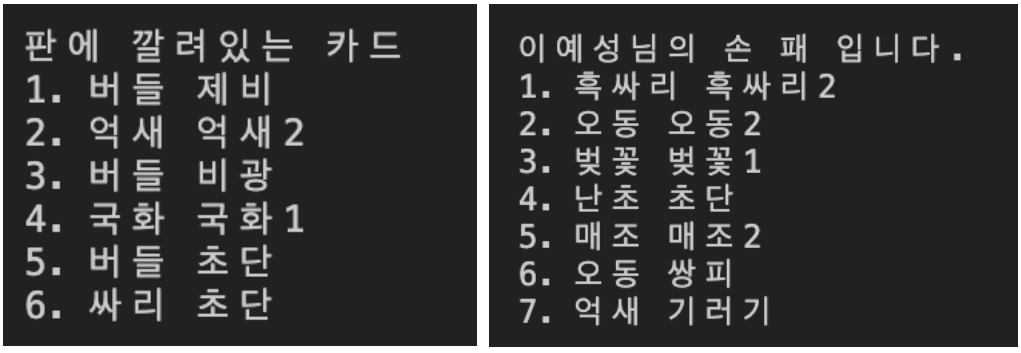
시간이 부족해서 완전한 게임이 동작하지 않는 것이 아쉽다.. 아쉬움을 뒤로 하고 일단 구현된 코드부터 설명을 하고자 한다.

〈 card.h 〉

```
class Card {
 public:
  Card(std::string name, bool gwang, bool yeolkkeut, bool godori, int pi, std::string _tti);
  CardInfo getCardInfo();     // 카드의 정보(id를 제외한 모든 필드)를 구조체 형태로 반환
  std::string toString(int i);     // 카드의 이름을 공백 기준으로 구분한 문자열을 반환
 private:
  bool gwang_;                // 광인지 아닌지
  bool yeolkkeut_;            // 열끗인지 아닌지
  bool godori_;               // 고도리인지 아닌지
  int pi_;                    // 피인경우 1, 쌍피인 경우 2, 아닌경우 0
  std::string tti_;           // 띠의 종류를 한글을 로마자 표기로 작성
  std::string name_;          // 카드의 이름 예) 송학 광, 송학 홍단, 송학 피1, 송학 피2
};

struct CardInfo {
  bool gwang;                // 광인지 아닌지
  bool yeolkkeut;            // 열끗인지 아닌지
  bool godori;               // 고도리인지 아닌지
  int pi;                    // 피인경우 1, 쌍피인 경우 2, 아닌경우 0
  std::string tti;           // 띠의 종류를 한글을 로마자 표기로 작성
  std::string name;          // 카드의 이름 예) 송학 광, 송학 홍단, 송학 피1, 송학 피2
};
```

거의 다 주석에 설명을 해두었다. 추가적으로 설명을 하자만 toString()함수는 int i를 받아서 해당 카드의 순서를 표시하기 위해서 사용(아래 사진)했다.




손에는 7장, 바닥에는 6장이 깔렸을 때의 모습이다. 카드 이름 뒤에 숫자가 붙은 것들은 각각의 월별로 2개의 피가 존재하는데 이를 구분하기 위해서 저렇게 만들었다.

〈 player.h 〉

```
class Player {
 public:
  explicit Player(std::string name);    // 생성자
  PlayerInfo GetPlayerInfo();           // 플레이어 정보 출력
  bool isPlaying();                     // 플레이어가 플레이중인지 확인하는 함수
  void UpScore(int amount);             // 점수를 더하는 함수
  void DownScore(int amount);           // 점수를 빼는 함수
  void ResetScore();                    // 점수를 0으로 초기화 하는 함수
  void draw(Card* card);                // 카드를 하나 핸드에 얻는 함수
  void earn(Card* card);                // 카드를 하나 필드에 얻는 함수
  Card* drop(int number);
 private:
  std::string name_;              // 플레이어 이름
  std::vector<Card*> hand_;       // 플레이어가 손에 들고 있는 카드들
  std::vector<Card*> field_;      // 플레이어가 가지고 있는 카드들
  bool isPlaying_;                // 플레이어가 플레이중인지
  int score_;                     // 플레이어의 점수
};

struct PlayerInfo {
  std::string name_;              // 플레이어 이름
  std::vector<Card*> hand_;       // 플레이어가 손에 들고 있는 카드들
  std::vector<Card*> field_;      // 플레이어가 가지고 있는 카드들
  bool isPlaying_;                // 플레이어가 플레이중인지
  int score_;
};
```

할 말이 참 많은 클래스이다. 초기에는 판돈이 있는 고스톱 게임을 만들어서 여러라운드를 거쳐서 고를 외칠 수 있고 스톱을 외칠 수 있어서 얻은 점수에 따라 돈을 분배하는 방식으로 last man standing 까지 진행하려고 설계를 했다. 하지만 점수도 계산하고 점수에 따라 돈도 분배 해야해서 나중에는 그냥 한 라운드만 진행하고, 3점을 먼저 내면 승리하는 방식으로 설계를 수정했다.(결국 이것도 못함,,,,)
Readme에도 설명을 했지만 플레이어의 이름은 사용자의 입력으로 받고, 입력한 문자열의 길이가 너무 길다면 다시 입력 받게 구현을 했다.

```
1번째 플레이어의 이름을 입력하세요 : 이예성
2번째 플레이어의 이름을 입력하세요 : 박형기
3번째 플레이어의 이름을 입력하세요 : 김민규
1번째 플레이어 : 이예성
2번째 플레이어 : 박형기
3번째 플레이어 : 김민규
```

## 〈 calculator.h 〉

```
class Calculator {
 public:
  void calculateScore(std::vector<Player*> &playerList); // 점수를 계산하는 함수
 private:
  static Calculator* instanse_; // Operator instance, instance함수로 만들기
};
```

플레이어가 가지고 있는 카드를 기준으로 점수를 계산하는 클래스이다. 점수를 계산하는 과정이 복잡할것 같아서 해당 클래스 설계를 하게 되었다. 원래는 싱글톤으로 구현하려 했으나.. 뭔가가 잘 안되는 느낌이라 다시 그냥 일반적인 객체 생성 방식으로 바꾸었다. 싱글톤을 위해서 남겨둔 instance_필드,,,
card.cpp 파일을 만들기는 했으나 시간이 부족해서 구현을 못했다.

## 〈 operator.h 〉

```
class Operator {
 public:
  void setGame(std::vector<Card*>& cardList, std::vector<Player*>& playerList,
  std::vector<Card*>& fieldCard);                                    // 첫 게임 셋팅을 위한 함수
  void playerDraw(std::vector<Card*>& cardList, Player* Player);     // 플레이어가 카드를 드로우 하는 행동
  void openCard(std::vector<Card*>& cardList, std::vector<Card*>& fieldCard);  // 카드 덱에서 하나를 필드로 까는 행동
  void giveCardTo(Player* Player, Card* card);                       // 플레이어에게 카드를 주는 함수
  bool isGameOver(std::vector<Player*>& playerList);                 // 게임이 종료됐는지 구하는 함수
 private:
  static Operator* instance_; // Operator instance, instance함수로 만들기
};
```

게임의 진행을 담당하는 클래스이다. 게임의 진행 과정을 위해서 필요한 각가지 함수들을 가지고 있는 객체이다. 게임 세팅을 위해서 카드를 셔플하고 바닥에 깔고 플레이어에게 분배하는일을 한다. 게임에는 사용하지 못했지만, 플레이어에게 카드를 주고, 바닥에 카드를 까는 함수도 있다. 게임을 끝났는지도 감지하는데 누구라도 3점을 넘어간다면 게임이 끝났다고 판단한다.
이 클래스도 점수 계산기와 동일하게 싱글톤으로 하려고 했으나 계산기와 동일한 이유로 설계를 변경했다...

## 〈 gostop.cpp 〉

```
> void initCard(std::string path, std::vector<Card*> &cardlist, bool debug) {…
> void initPlayer(std::vector<Player*> &playerList) {…


> void startGame() {…
> void showHand(Player* player) {…
> Card* pickHandCard(Player* player) {…
> void showField(std::vector<Card*> &fieldCard) {…
```

게임이 돌아가는 main문이 있는 곳이다. 여기에 저장이 되어있는 함수들은 card.txt파일에서 카드의 정보를 불러와서 카드 덱을 만들어주는 initCard 함수와 플레이어의 이름을 하나씩 입력받고, 길이가 적절한지 검증하는 initPlayer 함수가 있다.

그 아래에 있는 함수들은 콘솔에 나오는 내용을 다루는 함수들이다. 각각을 설명하면 다음과 같다.

- startGame : 게임 스타트를 알리는 함수
- showHand : 플레이어의 손패를 보여주는 함수
- pickHandCard : 손패에서 한장을 선택하고 그 카드를 보여주고 손패에서 제거하는 함수
  - 원래는 여기에 더해 바닥에 있는 매칭되는 카드와 함께 플레이어의 필드로 넣어주는 과정이 필요하나.. 구현을 못했다..
  - 지금은 카드를 선택하고 Y를 입력하면 턴이 넘어간다.
- showField : 바닥에 깔려있는 카드들의 정보를 보여주는 함수

## 〈 가정한 내용 〉

- 모든 플레이어는 정해진 입력만 입력한다.
  - 숫자를 입력해야할 때는 숫자만 입력한다.
  - 입력에서 예외가 발생하지 않는다고 가정한다.
- 플레이어들은 딱 한 라운드만 진행한다.
  - 3점이 먼저나면 그 플레이어가 승리한다.
  - 승리 보상은 딱히 없다.

## 〈 설계 변경사항 〉

- 플레이어는 500(만원)을 가지고 시작한다. 〉〉 삭제
- 점수에 따라서 해당 라운드의 판돈을 1등이 분배 받는다. 〉〉 삭제
- 1000만원을 먼저 만들면 승리한다. 〉〉 삭제
- 플레이어는 중간에 게임을 포기할 수 있다 〉〉 삭제
- 플레이를 포기한다면 모든 돈은 나머지 플레이어에게 n등분 된다 〉〉 삭제
- Operator, Calculator는 Singletond으로 만든다. 〉〉 일반 클래스로 디자인

## 〈 구현하지 못한 내용 〉

- 플레이어가 카드를 내면 바닥에서 매칭되는 카드를 찾아내는 함수
- 찾아낸 카드와 플레이어가 낸 카드를 플레이어의 필드에 넣는 함수
- 플레이어의 점수를 계산하는 함수(점수를 더하고 빼는 부분은 player 부분에 정의 되어있음)
- 플레이어의 필드를 보여주는 함수
- 플레이어의 점수를 보여주는 함수
- 게임이 끝났고, 누가 이겼는지 보여주는 함수