

Kode Kelompok : OPZ

Nama Kelompok : Ooploverz

1. 13521050 / Naufal Syifa Firdaus
2. 13521058 / Ghazi Akmal Fauzan
3. 13521066 / Muhammad Fadhil Amri
4. 13521070 / Akmal Mahardika N. P.
5. 13521168 / Satria Octavianus Nababan

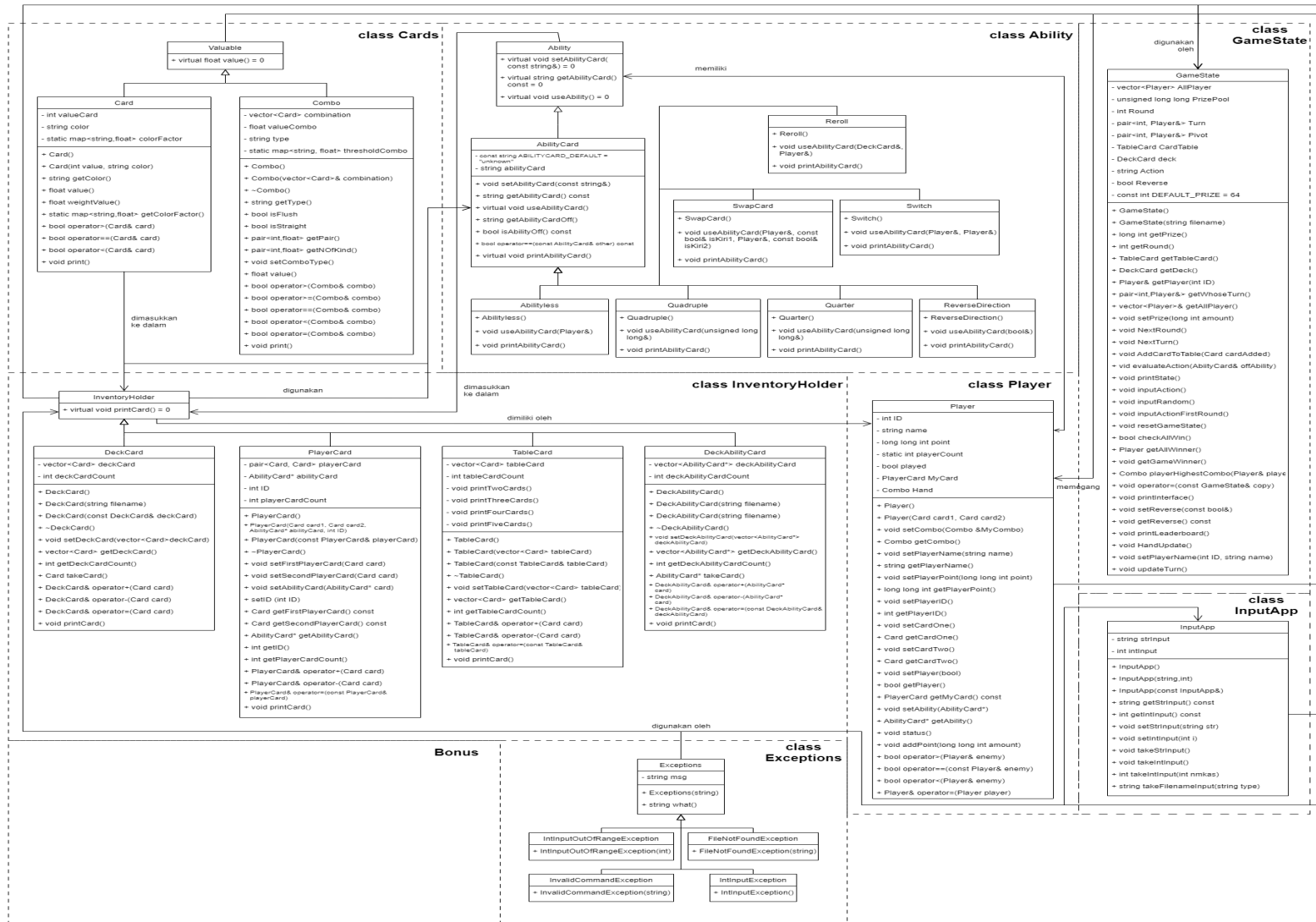
Asisten Pembimbing : Fabian Savero Diaz Pranoto

## 1. CLASS DIAGRAM

Pada diagram kelas ini, beberapa bagian yang membutuhkan implementasi 4 sekawan (constructor, destructor, copy constructor, dan operator assignment=), setter, dan getter, kelompok kami secara lengkap membuat implementasinya, tetapi terdapat beberapa bagian lainnya yang dirasa tidak memerlukan implementasi seluruh 4 sekawan sehingga tidak dimasukkan untuk efisiensi program.

Dibawah ini merupakan lampiran struktur desain diagram kelas kelompok kami, jika pada gambar dirasa tidak terlihat jelas dapat klik pada tautan berikut ini :

<https://www.figma.com/file/Qzkyo5bcRT1GAFN7P1CnVu/Mapping-Tubes1-OOP?node-id=0%3A1&t=PVmrSqRH7Ra74FgY-0>



Gambar 1.1. Flowchart Diagram Kelas

Alasan kami memilih diagram kelas yang seperti itu karena diagram kelas kami mengelompokkan spesifikasi tugas besar ini ke dalam beberapa kelas bagian, sehingga setiap diagram kelas termodularisasi dengan baik dan dalam pengerjaannya dapat terstruktur untuk masing-masing pembagian tugas setiap anggota yang lebih terarah. Kelebihan lainnya juga kami mengimplementasikan program dengan interface yang menarik yaitu terdapat Ascii Art dari gambar kartu berikut warna setiap kartunya, sehingga pengguna dapat bermain dengan melihat langsung gambar kartu yang dimilikinya. Namun, kekurangan dari implementasi program kami yaitu inkonsistensi dalam penggunaan *reference*, yang mana pada objektif program yang sama terkadang menggunakan *reference* tetapi juga *reference* terkadang tidak digunakan, sehingga hal ini dapat menimbulkan kebingungan. Kendala yang kami alami dalam menggunakan diagram kelas ini yaitu sulitnya proses pengintegrasian setiap kelas yang ada sebagai konsekuensi dari modularisasi kelas.

## 2. Penerapan Konsep OOP

Dalam OOP, terdapat empat pilar dasar, yaitu *inheritance*, *encapsulation*, *abstraction*, dan *Polymorphism*.

### 2.1 Inheritance & Polymorphism

*Inheritance* adalah konsep yang memungkinkan sebuah class untuk mewarisi sifat-sifat (*properties*) dan perilaku (*behaviors*) dari class lain yang lebih umum atau dikenal sebagai superclass atau parent class. Dengan menggunakan inheritance, sebuah class dapat menurunkan (*inherit*) sifat-sifat dan perilaku dari *superclass*-nya tanpa harus menuliskan kembali kode program yang sama, sehingga menghemat waktu dan usaha dalam pengembangan perangkat lunak. Dalam *inheritance*, terdapat hubungan *parent-child* antara class-class yang terlibat. *Child* class atau subclass adalah class yang mewarisi sifat-sifat dan perilaku dari superclass atau *parent* class, sedangkan superclass adalah class yang memberikan sifat-sifat dan perilaku kepada subclass-nya.

Polymorphism adalah konsep yang memungkinkan sebuah objek untuk memiliki banyak bentuk (*forms*) atau perilaku (*behaviors*) yang berbeda, tergantung pada konteks penggunaannya. Dalam OOP, *Polymorphism* seringkali diimplementasikan melalui penggunaan konsep *inheritance* dan *overriding*. Penggunaan *Polymorphism* memungkinkan programmer untuk menuliskan kode yang lebih fleksibel dan mudah dimengerti. Dalam *Polymorphism*, sebuah objek dapat dianggap sebagai objek dari kelas induknya (*parent* class) atau objek dari kelas turunannya (*child* class), tergantung pada konteksnya. *Polymorphism* juga memungkinkan programmer untuk menulis kode yang lebih generik atau abstrak, sehingga dapat digunakan kembali dalam situasi yang berbeda.

### 2.1.1 AbilityCard

```
class AbilityCard : public Ability
```

Gambar 2.1 Inheritance Pada AbilityCard

Terdapat konsep *inheritance* yang diterapkan dalam AbilityCard, yang mana pada implementasi AbilityCard terdapat implementasi dari method pure virtual dari Ability yaitu setAbilityCard, getAbilitycard dan useAbilityCard.

### 2.1.2 Abilityless

```
class Abilityless : public AbilityCard
```

Gambar 2.2 Inheritance Pada Abilityless

Pada Abilityless terdapat konsep *inheritance* yang memanfaatkan *attribut* dan *method* dari AbilityCard. Dalam hal ini juga terjadi konsep *polymorphism* pada *method* useAbility dari AbilityCard.

### 2.1.3 Quadruple

```
class Quadruple : public AbilityCard
```

Gambar 2.3 Inheritance Pada Quadruple

Pada Quadruple terdapat konsep *inheritance* yang memanfaatkan *attribut* dan *method* dari AbilityCard. Dalam hal ini juga terjadi konsep *polymorphism*, yang menggunakan *method* useAbility dari AbilityCard.

### 2.1.4 Quarter

```
class Quarter : public AbilityCard
```

Gambar 2.4 Inheritance Pada Quarter

Pada Quarter terdapat konsep *inheritance* yang memanfaatkan *attribut* dan *method* dari AbilityCard. Dalam hal ini juga terjadi konsep *polymorphism*, yang menggunakan *method* useAbility dari AbilityCard.

### 2.1.5 Reroll

```
class Reroll : public AbilityCard
```

Gambar 2.5 Inheritance Pada Reroll

Pada Reroll terdapat konsep *inheritance* yang memanfaatkan *attribut* dan *method* dari AbilityCard. Dalam hal ini juga terjadi konsep *polymorphism*, yang menggunakan *method* useAbility dari AbilityCard.

### 2.1.6 ReverseDirection

```
class ReverseDirection : public AbilityCard
```

Gambar 2.6 Inheritance Pada ReverseDirection

Pada Reroll terdapat konsep *inheritance* yang memanfaatkan *attribut* dan *method* dari AbilityCard. Dalam hal ini juga terjadi konsep *polymorphism*, yang menggunakan *method* useAbility dari AbilityCard.

### 2.1.7 SwapCard

```
class SwapCard : public AbilityCard
```

Gambar 2.7 Inheritance Pada SwapCard

Pada SwapCard terdapat konsep *inheritance* yang memanfaatkan *attribut* dan *method* dari AbilityCard. Dalam hal ini juga terjadi konsep *polymorphism*, yang menggunakan *method* useAbility dari AbilityCard.

### 2.1.8 Switch

```
class Switch : public AbilityCard
```

Gambar 2.8 Inheritance Pada Switch

Pada Switch terdapat konsep *inheritance* yang memanfaatkan *attribut* dan *method* dari AbilityCard. Dalam hal ini juga terjadi konsep *polymorphism*, yang menggunakan *method* useAbility dari AbilityCard.

### 2.1.9 Card

```
class Card : public Valuable
```

Gambar 2.9 Inheritance Pada Card

Pada Card terdapat konsep *inheritance* yang memanfaatkan *method* value dari Valuable.

#### 2.1.10 Combo

```
class Combo: public Valuable
```

Gambar 2.10 Inheritance Pada Combo

Pada Combo terdapat konsep *inheritance* yang memanfaatkan *method* value dari Valuable.

#### 2.1.11 DeckAbilityCard

```
class DeckAbilityCard : public InventoryHolder
```

Gambar 2.11 Inheritance Pada DeckAbilityCard

Terdapat penerapan konsep *inheritance* pada DeckAbilityCard dari InventoryHolder yang sekaligus menerapkan konsep *polymorphism* pada implementasi *method* printCard.

#### 2.1.12 DeckCard

```
class DeckCard : public InventoryHolder
```

Gambar 2.12 Inheritance Pada Deckcard

Terdapat penerapan konsep *inheritance* pada DeckCard dari InventoryHolder yang sekaligus menerapkan konsep *polymorphism* pada implementasi *method* printCard.

#### 2.1.13 Playercard

```
class PlayerCard : public InventoryHolder
```

Gambar 2.13 Inheritance Pada PlayerCard

Terdapat penerapan konsep *inheritance* pada PlayerCard dari InventoryHolder yang sekaligus menerapkan konsep *polymorphism* pada implementasi *method* printCard.

### 2.1.14 TableCard

```
class TableCard : public InventoryHolder
```

Gambar 2.14 Inheritance Pada TableCard

Terdapat penerapan konsep *inheritance* pada TableCard dari InventoryHolder yang sekaligus menerapkan konsep *polymorphism* pada implementasi *method* printCard.

### 2.1.15 Exception

```
class FileNotFoundException: public Exceptions
```

```
class FileNotFoundException: public Exceptions
```

```
class IntInputException: public Exceptions
```

```
class IntInputException: public Exceptions
```

Gambar 2.15 Inheritance Pada Exception

Terdapat penerapan konsep *inheritance* pada FileNotFoundException, IntInputOutException, IntInputException, dan InvalidCommandException dari Exceptions.

## 2.2 Method/Operator Overloading

Operator overloading adalah kemampuan untuk mendefinisikan perilaku operator tertentu pada suatu kelas atau tipe data. Dalam pemrograman OOP, operator seperti +, -, \*, /, dan lain-lain dapat digunakan pada tipe data dasar seperti int, float, dan double. Namun, operator-operator ini juga dapat didefinisikan ulang untuk berperilaku sesuai dengan kebutuhan pada kelas-kelas atau tipe data yang kita buat.

Dalam operator overloading, kita dapat mendefinisikan ulang operator untuk kelas-kelas atau tipe data yang kita buat dengan menggunakan fungsi operator yang disebut sebagai operator overloading function. Operator overloading function ditandai dengan nama fungsi yang sama dengan nama operator yang ingin kita overload, diikuti dengan tipe data yang berhubungan dengan operator tersebut.

```
bool operator>(Card& card);
bool operator==(Card& card);
bool operator<(Card& card);
```

```
bool operator>(CapsaCard& card);
bool operator==(CapsaCard& card);
bool operator<(CapsaCard& card);
```

```
bool operator>(Combo<C>& combo);
bool operator>=(Combo<C>& combo);
bool operator==(Combo<C>& combo);
bool operator<(Combo<C>& combo);
```

```
DeckCard& operator+(Card card);
DeckCard& operator-(Card card);
DeckCard& operator=(const DeckCard& deckCard);
```

```
PlayerCard& operator+(Card card);
PlayerCard& operator-(Card card);
PlayerCard& operator=(const PlayerCard& playerCard);
```

```
TableCard& operator+(Card card);
TableCard& operator-(Card card);
TableCard& operator=(const TableCard& tableCard);
```

```
bool operator==(const AbilityCard& other) const;
```

```
DeckAbilityCard& operator+(AbilityCard* card);
DeckAbilityCard& operator-(AbilityCard* card);
DeckAbilityCard& operator=(const DeckAbilityCard& deckAbilityCard);
```

Gambar 2.16 Penerapan Konsep Operator Overloading

Pada lampiran diatas dapat dilihat penerapan konsep operator overloading yang kami implementasikan pada Card, Combo, deckCard, PlayerCard, tableCard, AbilityCard dan dexkAbilityCard. Implementasi operator overloading tersebut terbagi menjadi beberapa jenis, diantaranya :

- Operator Aritmatika (+, -, )

Operator aritmatika dapat di-overload pada kelas-kelas atau tipe data untuk melakukan operasi matematika pada objek-objek tersebut. Contohnya, Fungsi operator+() digunakan untuk menambahkan dua objek Card dengan mengembalikan objek Card baru yang berisi hasil penjumlahan kedua objek tersebut. Dalam operator overloading function ini, kita menggunakan operator+() untuk menjumlahkan Card player yang bertambah seiring dengan berjalannya permainan.

- Operator Perbandingan (==, <, >, <=, >=)

Operator perbandingan dapat di-overload pada kelas-kelas atau tipe data untuk membandingkan objek-objek tersebut. Contohnya, operator == pada kelas AbilityCard dapat di-overload untuk membandingkan dua objek kartu ability.

- Operator Pemberian Nilai (=)



Operator pemberian nilai (=) dapat di-overload pada kelas-kelas atau tipe data untuk menetapkan nilai objek tersebut. Contohnya, operator = pada kelas PlayerCard dapat di-overload untuk menetapkan nilai objek Card yang satu ke objek Card yang lain.

## 2.3 Template & Generic Function

Template dan generik function adalah konsep yang memungkinkan pembuatan fungsi atau class yang dapat bekerja dengan tipe data yang berbeda secara generik atau umum.

Template adalah fitur pada C++ yang memungkinkan programmer untuk menulis kode yang dapat diterapkan pada tipe data yang berbeda secara umum. Dalam template, programmer dapat menuliskan sebuah class atau sebuah fungsi dengan parameter tipe data yang generik atau umum, sehingga dapat digunakan pada tipe data yang berbeda tanpa harus menuliskan kode program yang sama berulang-ulang.

Kedua konsep ini memungkinkan programmer untuk menulis kode program yang lebih generik dan dapat digunakan kembali, sehingga memudahkan dalam pengembangan aplikasi yang kompleks. Dalam OOP, template dan generik function merupakan fitur yang sangat penting dan sering digunakan.

```
template <class T>
// Preconditional: objects Ti
T& max(vector<T> &objects){
```

```
template<class T> // ASCENDING
vector<T> sort(vector<T> list){
```

```
template<class T> // Descending
vector<T> sortDsc(vector<T> list){
```

Gambar 2.17 Generic Function Pada UtilityFunction

Penerapan konsep *generic function* terdapat pada UtilityFunction dalam mencari elemen maksimum dan melakukan sorting list tipe generik pada minimal kartu, combo, dan player.

## 2.4 Exception

Exception atau pengecualian adalah suatu kondisi yang terjadi saat sebuah program mengalami kesalahan atau kegagalan dalam menjalankan suatu tugas. Pada OOP, exception diimplementasikan melalui penggunaan class Exception atau class turunannya. Ketika sebuah kesalahan atau kegagalan terjadi, program akan melempar (*throw*) sebuah *exception* yang sesuai dengan kondisi kesalahan tersebut. *Exception* ini kemudian akan ditangkap (*catch*) oleh program untuk melakukan penanganan atau pengelolaan kesalahan yang terjadi. Penggunaan exception sangat berguna dalam mengatasi kesalahan atau kegagalan yang terjadi saat program dijalankan. Dengan menggunakan exception, programmer dapat menangani kesalahan dengan lebih mudah dan mengurangi kemungkinan program mengalami crash atau berhenti bekerja secara tiba-tiba.

Pada tugas besar ini, penerapan konsep *Exception* kami implementasikan secara khusus pada file `Exception.hpp` yang kemudian *method* yang dibutuhkan akan dipanggil dari file tersebut. Berikut merupakan tampilan implementasi *exception* dari kelompok kami :

```
class Exceptions{
protected:
    string msg;
public:
    Exceptions(string);
    string what();
};

class FileNotFoundException: public Exceptions{
public:
    FileNotFoundException(string);
};

class IntInputOutOfRangeException: public Exceptions{
public:
    IntInputOutOfRangeException(int);
};

class IntInputException: public Exceptions{
public:
    IntInputException();
};

class InvalidCommandException: public Exceptions{
public:
    InvalidCommandException(string);
};
```

Gambar 2.18 Penerapan Konsep *Exception*

Pada gambar diatas dapat dilihat bahwa kami menerapkan exception untuk beberapa kondisi, yaitu *exception* jika file tidak ditemukan, *exception* jika input diluar dari *range* yang ditentukan, *exception* jika input yang diberikan tidak valid dan *exception* jika *command* yang diberikan tidak sesuai.

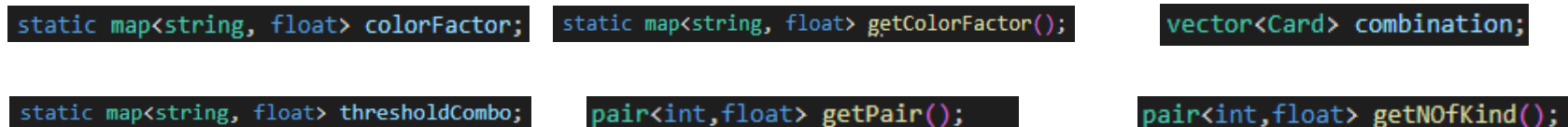
## 2.5 C++ Standard Template Library

Standard Template Library (STL) adalah salah satu fitur pada bahasa pemrograman C++ yang menyediakan kumpulan *template* class dan *function* yang digunakan untuk mempermudah dan mempercepat pengembangan program. STL menyediakan banyak kelas dan fungsi yang siap pakai, seperti *container*, *algorithm*, dan *iterator*.

Beberapa kelas *container* yang disediakan oleh STL adalah vector, deque, list, set, dan map. Sedangkan beberapa fungsi *algorithm* yang disediakan oleh STL adalah sort(), binary\_search(), dan find(). STL juga menyediakan *iterator* yang digunakan untuk mengakses elemen-elemen pada kelas *container*.

Penggunaan STL dapat mempercepat pengembangan program, karena programmer tidak perlu menuliskan kode dari awal untuk mengimplementasikan kelas container atau fungsi yang sering digunakan. Selain itu, penggunaan STL juga memudahkan dalam manajemen memori dan pengolahan data, karena STL sudah memiliki fitur-fitur tersebut secara otomatis.

### 2.5.1 Card



```
static map<string, float> colorFactor; static map<string, float> getColorFactor(); vector<Card> combination;

static map<string, float> thresholdCombo; pair<int, float> getPair(); pair<int, float> getNOfKind();
```

Gambar 2.19 Penerapan Konsep STL Pada Card

Penerapan STL yang pertama terdapat pada Card seperti yang terlihat pada gambar yang dilampirkan bahwa kami menggunakan kelas *container* map untuk menyimpan data dalam bentuk pasangan key-value colorFactor, getColorFactor(), dan tresholdCombo yang berguna untuk memudahkan penyimpanan dan pengaksesan data berdasarkan key-nya dan mempercepat waktu akses data yang mempunyai key unik dan tidak ingin menggunakan index pada array. Terdapat juga penggunaan pair untuk memudahkan dalam menyimpan dan mengakses dua data yang berbeda tipe dalam satu variabel, penerapannya terdapat pada getPair() dan getNOfKind(). Terdapat pula penggunaan vector pada combination untuk memudahkan penyimpanan dan pengaksesan data dalam bentuk array yang dapat diubah ukurannya secara dinamis saat program berjalan.

## 2.5.2 InventoryHolder

```
vector<AbilityCard*> deckAbilityCard; vect vector<Card> getTableCard(); tyCard(); vector<Card> deckCard; eCard; pair<Card, Card> playerCard;
```

Gambar 2.20 Penerapan Konsep STL Pada InventoryHolder

Pada InventoryHolder terdapat penggunaan *container* vector `deckAbilityCard()`, `getDeckAbilityCard()`, `deckCard()`, `getDeckCard()`, `getTableCard()`, `getTableCard()` dan `tableCard()` untuk memudahkan penyimpanan dan pengaksesan data dalam bentuk array yang dapat diubah ukurannya secara dinamis saat program berjalan. Terdapat juga penggunaan pair pada `playerCard` untuk memudahkan dalam menyimpan dan mengakses dua data yang berbeda tipe dalam satu variabel.

## 2.5.3 GameState

```
vector<Player> AllPlayer; pair<int, Player&> Turn; pair<int, Player&> Pivot;
```

Gambar 2.21 Penerapan Konsep STL Pada GameState

Pada GameState terdapat penggunaan *container* vector `AllPlayer` untuk memudahkan penyimpanan dan pengaksesan data beberapa player dan dapat diubah ukurannya secara dinamis saat program berjalan. Terdapat juga penggunaan pair pada `Turn` dan `Pivot` untuk memudahkan dalam menyimpan dan mengakses dua data yang berbeda tipe dalam satu variabel.

# 3. BONUS YANG DIKERJAKAN

## 3.1 Bonus yang diusulkan oleh spek

### 3.1.1 Generic Class

Generic class atau kelas generik adalah kelas yang didefinisikan tanpa menentukan tipe data yang akan digunakan. Sebaliknya, tipe data yang akan digunakan ditentukan saat membuat objek kelas tersebut. Kegunaan dari generic class pada OOP adalah untuk membuat kelas yang dapat digunakan dengan tipe data yang berbeda-beda tanpa perlu menulis kelas baru untuk setiap tipe data. Hal ini memudahkan programmer dalam membuat kelas yang dapat digunakan secara fleksibel dan menghindari pengulangan kode.

```
template <class C>
class Combo: public Valuable{
private:
    vector<C> combination;
    float valueCombo;
    string type;
    static map<string, float> thresholdCombo;

public:
    Combo();
    Combo(vector<C>& combination); // Combination sudah terurut sesuai besar nilai kartu
    ~Combo();
    string getType();
    bool isFlush();
    bool isStraight();
    pair<int, float> getPair();
    pair<int, float> getNOFKind();
    void setComboType();
    float value();
    bool operator>(Combo<C>& combo);
    bool operator>=(Combo<C>& combo);
    bool operator==(Combo<C>& combo);
    bool operator<(Combo<C>& combo);
    // Combo& operator=(Combo& combo);
    void print();
};
```

Gambar 3.1 Bonus Generic Class

Pada program diatas dapat dilihat bahwa kelas Combo merupakan sebuah generic class dengan menggunakan template dan tipe data C. Metode-metode dalam kelas Combo seperti operator>(), operator>=(), multiply(), operator<(), dan operator<() dapat digunakan dengan tipe data

apapun seperti int, float, atau double. Untuk membuat objek dari kelas Combo dengan tipe data yang berbeda, kita hanya perlu menentukan tipe data tersebut saat membuat objek.

### 3.1.2 Game Kartu Lain

```
class CapsaCard : public StandardCard{
private:
    string interface; // Biar masukin J,Q,K,A
    string kind;
    static map<string, float> kindFactor;
    static map<string, float> interfaceValue;
public:
    // Constructor
    CapsaCard();
    CapsaCard(string interface, string kind);

    // Methods
    string getColor();
    string interfaceVal();
    float value();
    float weightValue();
    static map<string, float> getKindFactor();
    static map<string, float> getInterfaceValue();
    bool operator>(CapsaCard& card);
    bool operator==(CapsaCard& card);
    bool operator<(CapsaCard& card);
    void print(); // Print kartu dalam bentuk kartu
    // Contoh
    // .------.------.------.------.
    // |C---. ||A---. ||R---. ||D---. |
    // | :/\: || (V) || :(): || :/\: |
    // | :V/: || :V/: || ()() || (__) |
    // | '--'C|| '--'A|| '--'R|| '--'D|
    // `-----'`-----'`-----'`-----'
};
```

Gambar 3.2 Game Kartu Lain

Capsa game atau permainan capsa susun adalah permainan kartu tradisional yang populer di Asia. Aturan main capsa game bisa bervariasi di setiap negara atau daerah, tetapi pada program ini aturannya adalah sebagai berikut:

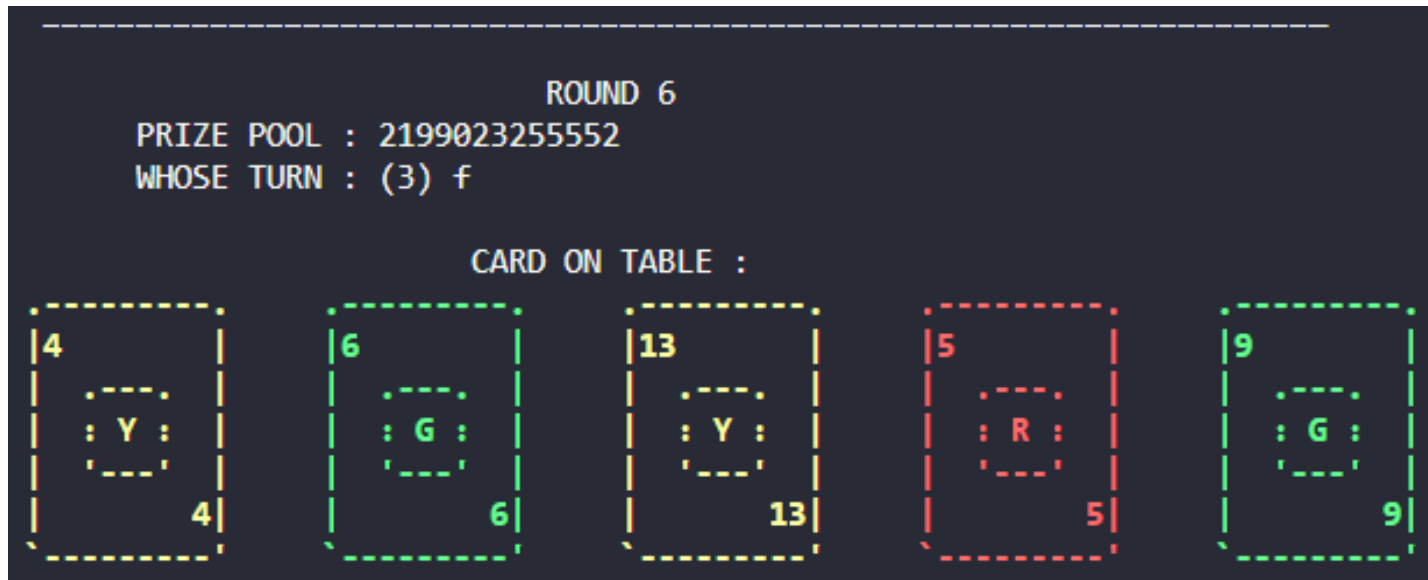
1. Setiap pemain diberikan 13 kartu yang harus disusun menjadi tiga set: satu set dengan tiga kartu (front), satu set dengan lima kartu (middle), dan satu set dengan lima kartu (back).
2. Urutan kartu dari yang tertinggi ke yang terendah adalah: A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2.
3. Front harus memiliki kombinasi kartu dengan nilai yang lebih rendah daripada middle dan back. Middle harus memiliki kombinasi kartu dengan nilai yang lebih rendah daripada back.
4. Setiap pemain membandingkan setiap set kartu miliknya dengan pemain lain. Pemain yang memiliki kombinasi kartu dengan nilai tertinggi pada setiap set akan memenangkan poin. Pemain dengan jumlah poin terbanyak akan memenangkan permainan.
5. Kombinasi kartu yang diakui pada capsa game umumnya mirip dengan poker, yaitu: *straight flush*, *four of a kind*, *full house*, *flush*, *straight*, *three of a kind*, *two pair*, *one pair*, dan *high card*.

Beberapa aturan tambahan pada capsa game yang diterapkan untuk membuat permainan lebih menarik, seperti:

- Jika ada dua pemain atau lebih yang memiliki kombinasi kartu yang sama, pemain yang memiliki kombinasi kartu dengan kartu tertinggi akan menang.
- Jika setiap pemain memiliki kombinasi kartu yang sama, maka pemain yang memiliki kombinasi kartu dengan kartu terendah di set front akan menang.
- Pemain yang menang pada setiap set kartu akan mendapatkan 1 poin, dan pemain yang menang pada ketiga set kartu akan mendapatkan bonus poin tambahan.

## 3.2 Bonus Kreasi Mandiri

### 3.2.1 Pencetakan Gambar Kartu (Representasi Fisik)

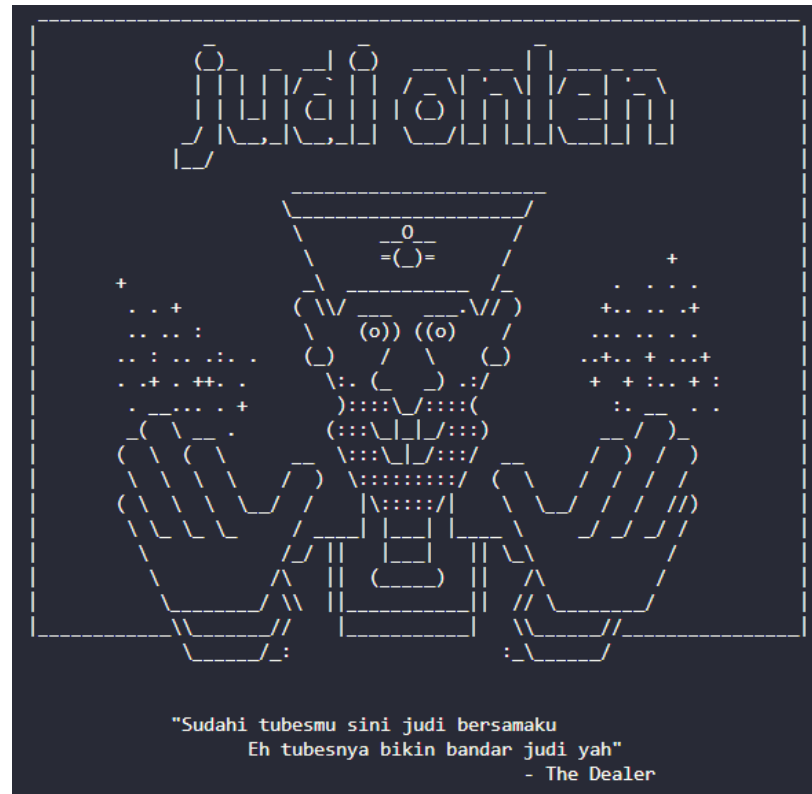


Gambar 3.3 Cetaklan Kartu pada CLI

Pada gambar diatas terlihat kreasi dari kami yang membuat tampilan gambar kartu yang sesuai dengan angka dan juga warna dari kartunya. Hal ini membuat permainan program kami menjadi lebih menarik dan pemain dapat bermain dengan langsung melihat tampilan kartu yang dimilikinya, selayaknya bermain kartu di dunia nyata.



### 3.2.2 Pencetakan Gambar saat Memulai Game



Gambar 3.4 Gambar saat Memulai Game

Kreasi selanjutnya yaitu gambar ascii art dari tampilan awal game yang sangat menarik, sehingga ketika pemain baru memulai program akan langsung melihat tampilan tersebut yang mengindikasikan bahwa pada game ini akan bermain kartu yang cara bermain dan aturan permainannya mirip seperti bermain judi poker pada umumnya.

## 4. Pembagian Tugas

Tabel 4.1 Pembagian Tugas

Modul (dalam poin spek)	Implementer	Tester
Alur Program (Game State)	13521050, 13521070, 13521066, 13521168	13521050, 13521058, 13521066, 13521070, 13521168.
Alur Program (main)	13521050	13521050
Mekanisme (Deck Kartu, Deck Ability)	13521058	13521058
Mekanisme (Player Card, Table Card, inventory Holder)	13521058	13521058
Mekanisme (Putaran Permainan, Poin Hadiah, Perintah Double dan Half serta Next, )	13521050	13521050
Mekanisme (Semua Ability)	13521070	13521070
Mekanisme (Combination, Exception)	13521066	13521066
Mekanisme (Player)	13521168	13521168,13521050
Bonus	13521066, 13521058	13521066
Lain-lain (utilityFunction)	13521066	13521066
Lain-lain (Input App)	13521070	13521070
Lain-Lain (Laporan, README)	13521168	13521168,13521050

## 5. Lampiran

### 5.1 Foto Kelompok



Gambar 5.1 Foto Kelompok

### 5.2 Link Penting

Link Repository : [https://github.com/akmaldika/Tubes1\\_OOP](https://github.com/akmaldika/Tubes1_OOP)

Link Figma : [Figma](#)

Link flowchart lebih jelas : [Flowchart Diagram Kelas](#)