知能情報工学実験演習 II

C++演習 (Part 1)

担当: 中村貞吾・下薗真一 (TA: 原田/井上)
E-mail: {teigo, m_harada}@dumbo.ai.kyutech.ac.jp,
sin@ai.kyutech.ac.jp,
t_inoue@pluto.ai.kyutech.ac.jp

期間: 第 3 年次前期

機材: CAD 実験室 (共通教育研究棟 3 階 S303 室) の Linux PC 端末

注意: 実験は Part 1 の 3 回 と Part 2 の 3 回，計 6 回に分けて行うが，実験・演習に割り当てられている時間以外でも，自由に計算機を使用してかまわない．研究棟側の端末でも実験は可能である．
Part 1 では実験は個人単位で行ない，各人別々にプログラムを作成すること．

1 演習の目標

C 言語および C++ 言語を使うプログラミングをしながら，実用的なプログラムを作るときに必要なプログラミング作法，つまり読みやすくミスの生じにくいプログラムの書き方を身につける．またその中で，構造化プログラミングと，その発展形であるオブジェクト指向プログラミングの理解を深め，その一部を活用できるようになることを目標とする．

一言でいえば，初心者プログラミングからの卒業が目標である．まずその指針を簡単に挙げておく．

1.1 同じことなら単純なものほどよい（オッカムの剃刀）

プログラムを書くとき，あまり深く考えず，とりあえず必要そうな変数を山のように宣言するのをよく見かける．その結果，使わない変数や値を与えない引数があっても気にしないでいる．もしそうなら，今日からやめること．

プログラムをなぜ書くのか．それは，プログラムに盛り込まれる規則，知識，アイディアを，プログラムの共同開発者，プログラムの利用者など，他の人と共有するためである．他の人と共有するためには，客観的な基準で整理整頓されている必要があり，そして単純簡潔であることが，整理整頓を容易に実現する最も有効な方法なのである．

1.2 まず “データ型” から作る

整数型や文字型，つまりバイトやワードなどCPUが直接操作できる型だけで実用プログラムを作ることは，組み込みマイコンのプログラムなどをのぞけば，めったにない．たいていプログラムの作成は，人間にとって意味のあるデータの単位をプログラムで扱う「型」として表現することからはじめる．実はこの作業が，プログラムに行わせる作業をきれいに整理しよく理解するための，つまり問題解決の鍵である．

C/C++は，構造体もしくはクラスという仕組みでデータ構造とその操作をはっきり対応させたプログラムを書くことができる言語である．この特徴を生かしてこそプログラミング言語である．

1.3 大域変数は禁止

大域変数はいつでもどこででも使え，引数に書かなくても関数の間でデータ受け渡しができるので即席プログラムには便利である．しかし，プログラムが大きく複雑になると，変数がどこでなんのため使われているかわからなくなる．とりわけ，オブジェクト指向プログラミングでは “大域変数を使いたい病” は大きな障害になる．

データを型として組み合わせてまとめ，使う変数を最少限にするのが，プログラムをうまく作るコツである．関数のデータのやり取りも，関数の動作や操作対象を単純明快にするため，引数と返り値のみで行いたい．

2 復習：C言語で型を作る

2.1 データ型とは一定の意味を持たせたモデルと操作の組合せである

デジタル機器では，どんなデータもビットの列で表す．例えば，整数`1040187392`と実数`0.125F`はどちらも 32 ビットの列

```
0011 1110 0000 0000 0000 0000 0000 0000
```

(`3E 00 00 00`) である．いや実際は，アルファベット 8 文字であるかもしれないし，音楽 CD に記録されたステレオ音声データの 1 サンプル目かもしれない．計算機のハードウェアには，どのアドレスのデータが何型かといった情報は自分でつけないかぎり存在しない．我々がさまざまなデータの異なる演算を使い分けられるのは，プログラムの中で `int` 型のデータには`int` 型の操作 (命令) を，`float` 型のデータには`float` 型の操作を（コンパイラが）選んでいるからである．

データ型の本質は，内部表現の方法ではなく，**モデル** (例えば数学でいう「整数」) と**操作** (例えば整数の加算演算) の対応関係にある．これは，`int` 型や`float` 型がメモリの中でどのように表されるかを知らなくても，プログラムを書いたりできることからも明らかである．データ型がどんな内部表現か知らなくても（普通は知らない），操作がその内部表現に対応すれば使えるのだ．

C++ で構造体／クラスを使ってデータ型と関数をまとめて書けるようになっているのは，この対応をプログラマが記述できるようにして，+-*/ など CPU が直接行える演算と同等に使用できるようにするためである．

2.2 計算機中に存在しないものを表すのが抽象データ型

プログラムで必要となるデータ型には様々なものがありうるので，プログラミング言語があらかじめそのすべてを用意することはできない．極端な話，整数や浮動小数点数ですら，桁や精度の制限があり，数学的な整数や実数を表しているわけではない．必要なデータ型は，使える型を組み合わせて**自分で作るもの**なのである．これが**抽象データ型**である．

自分で型を作るとき，以下の二つのことが必須となる．

(1) データ一個を表すのに必要なデータ（値）は，すべて一つにまとめる．
既存の型でも，例えば，浮動小数点数の符号部と指数部と仮数部は必ず一まとまりにして扱っている．分けて見たり操作する必要はない．

(2) 一個のデータは，いつもまとめて操作する．
既存の型でも，例えば，浮動小数点数の演算をすると，符号部も指数部も仮数部も一緒に計算される．

なぜなら，人間にとって，一つと考えられるものは「一つ」として扱えるほうがわかりやすく，楽で間違いがないからである．

2.3 データを一まとめにするには構造体を使う

データを一まとめにするには，必要なデータを組み合わせる．C 言語では，構造体 (struct) を宣言することによって行う．(C++言語では，構造体をクラスともよぶ．) 組み合わせに使うデータの型が言語で用意されたものか，別の抽象データ型かは問われない．

例えば，数学でいう $\mathbf{R}^2$ の要素，2 次元ベクトルを表すには，それぞれの成分を例えば double 型で表すことにして次のような構造体 vector2 を作ればよい．

```
struct vector2 {
        double x;       /* x 値 */
        double y;       /* y 値 */
};
```

もちろん，double val[2]; のようにして，x, y の各値を配列の要素で表すこともできる．

2.4 データの操作は一まとめで行なう

データには，内部表現を気にすることなく一まとまりとして扱えるよう操作を準備する．例えば浮動小数点数の加算では，指数部や仮数部の計算は表には見えない．同じように，抽象データ

型の操作も，その型のデータそのものを扱うような関数を準備して使う．こうすることによって，内部表現を変更しても操作関数を変更するだけで済む．

例えば，2 次元ベクトルの加法演算は，

```
struct vector2 addv2(struct vector2 u, struct vector2 v)
{
        struct vector2 ret;

        ret.x = u.x + v.x;
        ret.y = u.y + v.y;

        return ret;
}
```

と用意し，プログラムで 2 次元ベクトル同士の加算が必要なら，

```
vec0 = addv2(vec1, vec2);
```

などと書く．操作関数を作る手間を惜しんで，いちいち

```
vec0.x = vec1.x + vec2.x;
vec0.y = vec1.y + vec2.y;
```

などと書いていれば，かえって手間がかかるばかりでなく，後で何をしたかったのかがわかりにくいコードになり，プログラミングを破綻させる原因になる．

2.5 抽象データ型は関数の作り方の指針でもある

今までは，関数の宣言のしかたはわかっていても，なにを一つの関数にすればよいのか，わからなかったかもしれない．抽象データ型の考えを使えば，**あるデータ型をあつかう関数は，その型のデータの操作として意味のある関数にする**という，関数の作り方の指針が明確になる．

プログラムを作る際は，組み込みの型から小さな型を，小さな型から大きな型を作っていく．同様に，関数もデータに必要な操作ごとに，小さな型のための関数を組み合わせ大きな型のための関数を作ってゆく．こうすることで抽象度の高いレベルで処理を簡潔に記述でき，自然ときれいな —— つまり読みやすくミスの起きにくい —— プログラムが書けるのである．

演習 1 (C 言語による) 2 次元ベクトル型のソースファイルが，vector2c.h および vector2c.c として，またそれを使った簡単な主プログラムが vec2ex1c.c として，それぞれ用意されている．以下の演習を実施せよ．

(1) プログラムをコンパイルし，実際に動くことを確かめよ．

(2) 減算の関数を vector2c.h および vector2c.c に追加し，また，vec2ex1c.c の主プログラムに減算のテストを追加し，実際に動くことを確かめよ．

ソースファイルの構成法とコンパイル ——————————————————————

C や C++ では，原則として，以下のようにソースファイルを分割する．

(a) **ヘッダファイル**

定数，型の定義 (具体的には，構造体 (や C++ のクラス) の定義になる)，関数のプロトタイプ宣言を，名前が.h で終るファイルにまとめる．これを，ヘッダファイルという．ヘッダファイルは，関連する型 (1 つの場合もある) ごとにまとめる．

(b) **プログラムファイル**

C や C++ の関数の処理を記述するファイルをプログラムファイルという．プログラムファイル名は，C 言語では .c で，C++ 言語では .cc，.c++，.cpp などで終るようにする．プログラムファイルの先頭では #include で必要なヘッダファイルをインクルードする．

プログラムファイルは，例えば，それぞれの型毎に 1 つずつにわけ，特定のアプリケーションの主プログラムなども，型とは別のファイルにする．このようにすると，例えば上の例では，2 次元ベクトルを用いた主プログラムを変更しても，2 次元ベクトルの加算，減算などの関数のファイルには影響がないことになる．

複数のソースファイルから成るプログラムをコンパイルするには，

```
% cc vector2c.c vec2ex1c.c
% c++ vector2p.cc vec2ex1p.cc
```

のように，必要なソースファイル名を並べればよい．また，

```
% cc -c vector2c.c
% cc -c vec2ex1c.c
```

のように，個々のファイル毎にコンパイルしておき，後で，

```
% cc vector2c.o vec2ex1c.o
```

のようにリンクだけ行なうこともできる．これを**分割コンパイル**という．

3　C++言語で型を作る

3.1　C 言語での型作りの弱点

前節のように，C 言語でも抽象データ型を作ることはできるが，ソースコードがもっと複雑で大規模になった場合, あるいは複数のプログラマで分担して作業するような場合に，次のような問題や不都合が生じる．

(1) 構造体 vector2 のメンバ変数に，どのような状況でも直接アクセスができる．逆に，メンバ変数の値を直接変更した場合，なにが起きるか予想できない．プログラマの心がけに任せるばかりでなく，むしろ積極的に，たとえば vector2 をベクトルとして利用する部分では

```
vec0.x = vec1.x + 2 * vec2.x;
```

のような，ベクトルとしての意味がはっきりしないメンバ変数の直接操作はできないようにしたい．

(2) 加算には，2 次元ベクトルの加算だけでなく，3 次元ベクトルや行列の加算もある．それらは「加算」という意味では同じであるが，引数の型が異なるためaddv2，addv3，addmatrix など，それぞれ別の関数名を付ける必要があり，命名と利用の両面で煩わしい．

(3) 変数の宣言や動的割り付けを行った直後に，その変数の各メンバ変数の値がどうなっているのか，あるいはどうする必要があるのかがプログラムからはっきりしない．初期値が 0 や NULL なのか，あるいは不定なのかはコンパイラや環境にも依存する．メンバ変数に抽象データ型が使用され入れ子になっている場合，どこまで初期値の面倒をみればいいのかも不明であり，めんどうである．

3.2　C++言語ではクラスによって型作りが楽になる

C++ では，こののような問題を解決するために**クラス**というデータ型定義の機構が提供されている．クラス class は，C 言語の構造体 struct を拡張したもので,「データを一まとめにする」機能に加え，初期化や操作，アクセスなどそのデータに必要な関数も一まとめにすることができ，またそのデータの内部，つまりメンバ変数や関数へ外からアクセスできるかどうかの制御もできる[1]．これは**データと手続きのカプセル化 encapsulation** と呼ばれ，オブジェクト指向プログラミングでの考え方の一つである．また，変数の宣言や動的領域の確保など，変数が用意される時，初期化の関数を指示しなければデフォルトの初期化関数が自動的に呼び出される．必要な初期化が必ず行われるようにできるのである．これも，オブジェクト指向プログラミングでの考え方の一つである．

2 次元ベクトル型をクラスで表現すると以下のようになる．

[1] 実は C++ では構造体 struct も関数を持てるよう拡張されていて，その違いは情報隠蔽を明示的に行うか否かの違いだけである．

```
class Vector2 {
// メンバ変数
private:
        double x;       // x 値
        double y;       // y 値
// メンバ関数
public:
        Vector2(void)   { }
                // デフォルトコンストラクタ (初期値不定)
        Vector2(double x0, double y0);
                // x0, y0 を与えて定数 (x0, y0) を得るコンストラクタ
        Vector2 add(Vector2 u);
                // 自身のベクトルとベクトル u との和を求める
        ....
};
```

このように，メンバ変数と同様に，その型専用の関数も記述できるのである．

クラスでのメンバ変数の宣言は，構造体での宣言と同じである．ただし，アクセス指定子 `private:` によって，メンバ変数 `x` と`y` をこのクラスのメンバ関数 (後述) だけがアクセスできる「プライベート (私的) な」変数とすることができる．これにより，メンバ変数が知らないうちに — つまりクラスの定義以外の場所で — 変更されるプログラムが書かれてしまう心配がなくなる．

データのクラスにそなわるメンバ関数を呼び出すときは，構造体のメンバ変数と同様に，変数に `.` で結び付けた形で記述する．例えば，

```
Vector2 v1, v2, v3;
...
v3 = v1.add(v2);
```

という形で呼び出す．これは，C 言語の場合には，

```
v3 = addv2(v1, v2);
```

のように書いていたプログラムに相当する．

クラス内で宣言したメンバ関数の実体は，例えば次のようにプログラムする．

```
Vector2 Vector2::add(Vector2 u)
{
        Vector2 ret;

        ret.x = x + u.x;
        ret.y = y + u.y;

        return ret;
}
```

ここで，ただの `x` と `y` は[2] `add` 関数を呼び出した (作用する) 変数のメンバ変数，つまり `v1.add(v2)` では `v1` のメンバ変数 `v1.x`, `v1.y` を指す．

[2] 実際には「このデータ (オブジェクト)」のポインタを表す `this` を使って `this->x` と `this->y` と書くところで，`this` が省略されている．

上の例では，メンバ関数はアクセス指定子 public: によって外部から使うことができる「公開された」関数になっている．メンバ変数，メンバ関数は，それぞれ private: (そのクラス以外からはアクセス不可能)，あるいは public: (そのクラス以外からもアクセス可能) にできる．メンバ変数を private: に，メンバ関数を public: にするのが典型的である．

3.3 C++言語では変数の初期化をはっきり型に書く

クラス名と同じ関数名のメンバ関数は，**コンストラクタ**と呼ぶ．コンストラクタはそのクラス (型) の変数の場所をメモリ上に確保するときに呼び出され，その変数が持つメンバ変数の値を設定する．その返り値は，そのクラス型のデータ (オブジェクト) と決まっているので，返値は記述しない．変数の初期化し忘れを防ぐために，コンストラクタは必ず呼び出されることになっている．コンストラクタは引数を持つこともでき，初期化に使う値を引数で指定したり，他の型のデータを初期化に使うものも作成できる．引数の無いコンストラクタは変数宣言などの際に暗黙のうちに呼び出されるので，特に **デフォルトコンストラクタ** と呼ばれる．

コンストラクタは，変数宣言などプログラマが意識して行う場合のほかにも，代入 = や型変換 (タイプ・キャスト) で気づかないうちに呼び出されることもあるため，プログラムのミスの原因になることがある[3]．代入やコピーで呼び出されるコンストラクタは**コピーコンストラクタ**とよばれ，引数は同じクラスのデータへの参照 & 一つのみである．C++ プログラミングでのミスを減らすには，自分で作るクラスにはデフォルトコンストラクタ，コピーコンストラクタはとりあえず作っておく習慣をつけるとよい．

演習 2 (C++言語による) 2次元ベクトル型のソースファイルが，vector2p.h およびvector2p.cc として，またそれを使った簡単な主プログラムがvec2ex1p.cc として，それぞれ用意されている．以下の演習を実施せよ．

(1) プログラムをコンパイルし，実際に動くことを確かめよ．

(2) クラス Vector2 の減算のメンバ関数をvector2p.h および vector2p.cc に追加し，また，vec2ex1p.c の主プログラムに減算のテストを追加し，減算が行われることを確かめよ．

[3] Smalltalk (Squeak) や Java などの言語ではクラス型データはポインタなど間接参照が基本なので目立たないが，C/C++ では構造体の代入や引数での値参照が可能なので，注意が必要である．

演習 3 コンストラクタが明示的に，また暗黙の内に選択的に呼び出されることを以下の方法で確かめよ．

(1) デフォルトコンストラクタ `Vector2(void)` で，`x` と `y` に初期値 `0` がセットされるよう変更し，`printf` 等で変数宣言などの直後に初期化されているかどうかを確かめよ．

(2) デフォルトコンストラクタと，コンストラクタ`Vector2(double, double)` の中で `printf` 等でメッセージの印字を行うようにして，適切なコンストラクタが呼び出されているかどうかの確認をせよ．

(3) 値のセットされた `Vector2` 型オブジェクトと全く同じ `x` およびy 成分の値を持つオブジェクトとして初期化するコピーコンストラクタ `public: Vector2(Vector2 & v)` を宣言し，(2) と同様に呼び出されているかどうかの確認をできるようにして，プログラム中で`Vector2` 型変数への代入を行い確かめよ．

演習 4 2 次元ベクトル型を参考にして，2 次正方行列型をクラスとして実現せよ．また，簡単な主プログラムを作成し，実際に動くことを確認せよ．
2 次正方行列型は，少なくとも，入力，出力，加算，減算，乗算の各メンバ関数を持つようにせよ．なお，クラスの実現には，2×2 の 2 次元配列を用いればよい．

3.4 C++では関数の引数の型や数が関数名の一部である

コンストラクタは，すべてクラス名と同じである．コンストラクタ以外にも，C++言語では，対象となるクラス（型）や引数，またその数が異なれば，異なる関数に同じ名前を付けることができる．この機能を正しく使うと，例えば次元の異なるベクトルの加算のように，扱うデータ型は異なるが，人にとって同じ意味や機能を持つ関数を，同じ名前にすることができ，便利である．これはオブジェクト指向プログラミング言語に広く取り入れられている考え方の一つで，**ポリモルフィズム** と呼ばれている．

この機能を使うと，まったく異なる意味や機能の関数を同じ名前にすることもできる．つまり，混乱やミスを誘発するような使い方もできるので，注意が必要である．

演習 5 先に作ったプログラムをあわせて，2 次元ベクトル型と 2 次正方行列型の双方を使うプログラムを作れ．このとき，加算や減算の関数が，同じ名前でもコンパイルでき，正しく動作することを確かめよ．

4 クラスで集合を表す

集合は数学の重要な基礎概念であると同時に，その実現方法はプログラミングの重要な基礎でもある．ここでは，トランプのカードを例にクラスによる集合の実現を扱う．

4.1 トランプの手は集合である

まず，カード 1 枚を表すためのデータ型を考えよう．トランプのカードには，スペード ♠，ダイヤモンド ♢，ハート ♡，クラブ ♣ の 4 つのスート[4] があり，それぞれのスートにエース (A)，2 から 10 までの字札，そして Jack (J)，Queen (Q)，King (K) のコートカード (絵札) の計 13 枚がある．ここでは，さらに 1 枚ジョーカー (道化師) Joker を加え，計 53 枚のセットとする．

ここでは，カードのスート (suit) と番号 (number) を別々のメンバ変数として持つトランプカード型 Card を考える．

```
class Card {
// メンバ変数
private:
        int suit;       // 組
        int number;     // 番号
// メンバ関数
public:
        ...
};
```

なお，suit の値とスペード，ダイアモンド等との具体的な対応は，enum によってつける．

次に，一人のプレーヤの「手」の表現方法を考える．これは，トランプカードの集合であると考えられる．ここでは集合の実現に配列を用いる[5] ．

```
class CardSet {
// メンバ変数
// 定義・宣言
public:
        const int maxnumcard = 53;      // カード総数
private:
        int numcard;            // 現在の集合内のカード数
        Card cdat[maxnumcard];  // カードのデータ
// メンバ関数
...
};
```

カードのデータは配列 cdat に入る．また numcard の値が 10 であれば，実際にカードのデータが納められているのは配列 cdat の添字が 0 〜 9 までの範囲とする．ここでは，トランプを 1 デッキしか使わないことを仮定しているので，配列は長さ 53 あれば十分である．この 53 という決まった数をプログラムで参照するのに，C 言語では#define コンパイラ指令を利用して，マクロとして名前をつけていた．C++ では，const 指定をして値を変更できない変数，すなわち定数として宣言し，利用する．

[4] タロットではそれぞれ剣，コイン (またはペンタクル)，カップ，ワンド (木杖) にあたり，アリストテレスの四元素説の 風，土，水，火 を表す．

[5] いわゆる線形リストのようなヒープメモリを用いる (動的な) 集合の表現は，今回は扱わない．

演習 6 トランプカード型およびトランプカードの集合型のソースファイルがcardset.h およびcardset.cc として，またそれを使った簡単な主プログラムが cardsetex1.cc として，それぞれ用意されている．以下の演習を実施せよ．

(1) プログラムをコンパイルし，実際に動くことを確かめよ．

(2) クラス CardSet のメンバ関数 remove() (2 種類あり) を実装して cardset.cc に追加し，また，cardsetex1.cc の主プログラムにカード削除のテストを追加し，動作を確認せよ．

5 クラスをもとにより大きなクラスを作る

次に，具体的なトランプゲームをもとに，ゲームの状態を表すクラスを考える．ここでは，いわゆる「ババ抜き」[6] を取り上げる．ババ抜きでは，各プレーヤの持ち手 (持ち札) がゲームの状態を表すと考えられる．この演習では，既にカードの集合型を実現しているので，それを利用することを考えると，ババ抜きゲームの状態を表す型は以下のようになる．

```
class BabaState {
// 定義・宣言
public:
        const int numplayer = 5;        // プレーヤ数
// メンバ変数
private:
        CardSet hand[numplayer];  // 各プレーヤの持ち手
// メンバ関数
        ...
};
```

ゲームの状態は，最初に全員にカードが配られた後は，あるプレーヤが別のプレーヤのカードを取り，必要なら番号が一致する 2 枚のカードを捨てることによってのみ遷移する．ここでは，前者をコンストラクタまたは関数 reset() で，後者を関数 move() で実現しよう．

演習 7 ババ抜きの状態を表現するための型がソースファイルbabastate.h およびbabastate.cc に，またそれを使った簡単なババ抜きプログラムがbabanuki1.cc として，それぞれ用意されている．以下の演習を実施せよ．

(1) BabaState::move() は実現されていない．これを実現し，プログラムをコンパイルし，動作を確かめよ．

6 習うより慣れよ

以下の 3 つの課題のうち，1 つを選んで取り組め[7] ．

[6] もとは ♣ の Queen (Argine) を抜いて行う Old maid というゲームで，Joker は日本でトランプに追加されたものだという．

[7] 無論，すべてを実装することを妨げない．

演習 8 ここまで，Card クラスは，suit とnumber という二つの整数をメンバ変数に用いて実現してきた．しかし，例えば，0〜12 はスペードの 1〜13, 13〜25 はダイヤモンドの 1〜13 などと決めてやれば，一つの整数でも実現できる．
CardSet クラスやメインプログラムに影響を与えることなく，Card クラスの実現方法を変更し実行せよ．

演習 9 ババ抜きの状態を表すクラス BabaState を別の実現方法で実現せよ．例えば，長さ 53 の配列にそれぞれのカードの持ち主を記録しておく方法でも状態を表すことはできる．

◆レポートの形式内容と提出方法

1. A4 のレポート用紙を用い，上部をステープラでとめること．
2. 表紙には，演習名 (知能情報工学実験演習 II C++演習)，名前，学生番号，実験日，班名，レポート提出日を明記すること．
3. 以下の内容を明確に示すこと．
   (a) 演習ごとの
      i. (自由演習については) 設定した課題の説明
      ii. プログラムのリスト (ただし，最初に与えられたファイルそのままの場合および前の演習と重複する場合は省略可)
      iii. (演習 11 以降の場合は)(主な) 関数の実現の考え方 (工夫した点および不完全な点を含む)
      iv. 考察あるいはこの演習で理解した点
   (b) 全体の考察[8] とまとめ
4. レポートは，このテーマの実験最終日の翌週の同一曜日までに，研究棟 7 階 (E706) の嶋田のもとまで提出すること．レポートは評価され，不備のあるものは〆切から 1 週間以内に知能の掲示板に再レポートの掲示をするので，注意しておくこと．

[8] 感想ではなく考察をすること.「面白かった.」や「難しかった.」だけではダメ.