知能情報工学実験演習II
C++演習

下薗／中村

TA：石田、簑代

演習の目的と内容

- 前半：C++ 言語プログラミング演習
  - データの抽象化からオブジェクト指向プログラミングへ
- 後半：グループワーク
  - 前半使ったプログラムを利用しながらグループでカードゲームの思考ルーチン作成

※このスライド

~sin/C++/C++2015.pdf

レポートの作成、提出

- C++ 演習
  - 各自が課題にとりくみレポート提出
  - 実験の授業時間中にTAに形式のチェックをうけサインをもらったうえで、
    3回目の終了時以降、４月２８日17:00までに
    学科第一計算機室（研究棟東棟6F）レポートポストに提出

レポートの作成、提出

- グループ演習（後半）
  - 詳細は後半開始時に説明します

知能情報工学実験演習II
C++演習（第１回）

第1回目は・・・

- プログラミングの作法を身につける
  - 大域変数を用いない
  - シンプルにする
  - データと型の「意味」を考える
- C言語でのデータ型への意味づけから
  C＋＋言語での型への意味づけサポートへ
  - struct, union, class
  - enum, const, & 修飾（参照）

第1回目の内容（2）

- C++ での class, struct の拡張
  - 「情報隠蔽」が可能（class でデフォルト）
  - 「継承」が利用可能
  - データのライフサイクル
    （誕生と初期化、破棄と消滅）を明示化
  - 型と操作関数を結びつけ、
    ポリモルフィズムやオーバーライド／オーバーロードを可能に

第1回目の達成目標

❑本日の演習範囲

- ■演習1から演習5まで
- ■演習1
  - ・C言語による構造体：vector2
- ■演習2～5
  - ・C++言語によるクラス：vector2の改良
  - ・そのなかで，コンストラクタやポリモルフィズムについて学ぶ

サンプルプログラム

- ~sin/C++
  - 演習1
    - Makefile, vector2c.h, vector2.c, vec2ex1.c
  - 演習2以降
    - vector2.h, vector2.cpp, vec2ex1.cpp
  - make vec2ex1c 等でコンパイル
    - ファイル名を変える場合は Makefile の書き換えが必要

演習1について(構造体とデータ操作)

- 2次元ベクトルを表す型

```
struct vector2 {
        double x;
        double y;
};
```

- 2次元ベクトルの加法演算

```
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;
};
```

- 2次元ベクトルの加法演算

```
struct vector2 vec0, vec1, vec2;
vec0 = addv2(vec1, vec2);
```

- vector2 は，構造体ではなくベクトルとして扱おう

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

・・・ばらして使うのは、ダメ

演習2について(クラス)

- 構造体の問題点
  1. 構造体のメンバ変数をいつ、どこ、だれでもいじれる
  2. 他の型の意味の同じ操作には別の名前をつける必要がある

     例) addv2, addv3, addmatrix

     例)set_add(myset, anelement)
- クラス:データと関数をまとめたもの

```
class Vector2 {
    private:    外にみせない(デフォルト)
        double x;
        double y;
    public:     外からも使ってもらう
        Vector2(...)          { }    コンストラクタ(データを作るとき呼ばれる)
        Vector2(double x0, double y0); 引数付で変数宣言して初期化できる
        Vector2 add(Vector2 u);
        メンバ関数はメンバ変数にアクセス可, 第一引数は呼び出したデータ自身
};
```

※Cとの違い

- class/struct の型名使用に typedef は不要.

```
class Vector2 {
        …
};

int main(int argc, char * argv[]) {
        Vector2 v, u;
        …
```

演習2について(メンバ関数)

- メンバ変数の呼び出し方

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

メンバ関数はオブジェクトに作用する

- C言語で相当するものは

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

- メンバ関数ではC言語の関数より引数が1つ減る
- 2次元ベクトルの加法演算

```
Vector2 Vector2::add(Vector2 u) {
        Vector2 ret;
        ret.x = x + u.x;
        ret.y = y + u.y;
                メンバ関数からはメンバ変数にアクセス可
        return ret;
};
```

演習3について(コンストラクタ)

- メンバ変数を初期化
  - 関数の名前をクラス名と同じにする
  - publicで宣言する
  - 型指定はない(すなわち値は返さない)
  - 引数をとるのは自由
    - 引数のないコンストラクタ:デフォルトコンストラクタ
    - 代入やコピーをする:コピーコンストラクタ
      - Vector2 b=a;　　// 但し、aは初期化済み
- コンストラクタを作る習慣を！

演習3について(コンストラクタ):続き

```
Vector2 v, u(2, 3), w(u);
```

```
public:
    Vector2(){ }

    Vector2(double x0, double y0)
    {
        x = x0;
        y = y0;
        printf("引数あり");
    }

    Vector2(Vector2 &v){
        .....
    }
```

関数の引数(値渡し)のときも呼ばれる

コンストラクタ（補足）

- クラスが生成されたときに自動的に呼び出される
  - 引数の有無などによって適切に呼び出される

```
class Vector2 {
  private:
    double x;
    double y;
  public:
    Vector2(){ }
    Vector2(double x0, double y0)
        { x = x0;  y = y0; }
    Vector2(const Vector2 & v)
        { x = v.x;  y = v.y; }
    void print()
      { printf("( %f %f )", x, y); }
};
```

```
int main() {
  Vector2 v1;
  Vector2 v2(2.4,  5.5);
  Vector2 v3=v2;
  v1.print();
  v2.print();
  v3.print();
};
```

```
%./a.out
( ?? ?? )
( 2.4 5.5)
( 2.4 5.5)
```

演習4について

- 2次正方行列のクラス
  - 実現には2×2の2次元配列を
- メンバ関数
  - add, sub, mul, print, scan
  - Vector2 を参考にすればよい

```
Class Matrix22{
    private:
        x[2][2];
    public:
        .....
```

演習5について(ポリモルフィズム)

- ポリモルフィズム
  - 関数名が同じでも引数が違えば別の関数
  - 人間にとって分かりやすい
    - 例えば，足し算なら全てにaddと名付けて良い
    - 変数の足し算: add(double i)
    - ベクトルの足し算: add(Vector2 u)
    - 行列の足し算: add(Matrix a)
  - 異なる意味や機能に同じ名前をつけると混乱のもと

- Vector2のaddとMatrixのaddは別物
  - プログラムを混ぜ合わせても正しく動くことを確認する

本日の達成目標

- 演習5まで終了し、レポートを書きはじめる.
  - できたかどうか、TAに確認してもらうこと
  - すんだら、次回以降の内容を自習しても可
  - 終わらなかった場合、自分で時間をみつけてすませておくこと.

- 次回の最初は，4節(クラスで集合を表す)以降の説明を行います

レポートについて

- 実行結果（出力）をつけるだけでは報告になりません！
  - 実験装置（プログラム）と実験を簡潔に説明し，予想を立てて，結果から確かめること．
- なぜ，そうなるのかをプログラムと実行結果を照らし合わせて説明すること

知能情報工学実験演習II
C++演習（2）

下薗／中村

TA：佐藤誠／古野雅大

集合の実現

- トランプカードの手はカードの集合
- 集合は列、連結リスト、木などで表せる
  - どのようなデータ構造がよいかは問題ごとに異なる
- ここでは配列を使って集合を表す
  - 動的データ構造より簡単（オブジェクトの生成・消滅に気をつけなくてすむ）
  - トランプのカードは多くない（全部札をもっても52～3枚）

サンプルプログラム

- トランプのクラス
  - cardset.h
  - cardset.cc
  - cardsetex1.cc
- ばば抜きのクラス
  - babastate.h
  - babastate.cc
  - babanuki1.cc

トランプカード型Card

- スートと番号の対で表す

```
class Card {
private:
        int suit;          // スート
        int number;        // 番号
public:
        Card(void)      { }                 デフォルトコンストラクタ
        void set(int st, int num)           スートと番号をセット
        bool equal(Card tgt);               同じか否かの判定
        int gnumber(void);                  アクセサ
        bool scan(void);                    入力（スートは文字列で）
};                                          エラーのとき返り値はtrue
```

enumは名前・記号をコード化する

- 番号が0から自動的に振られる

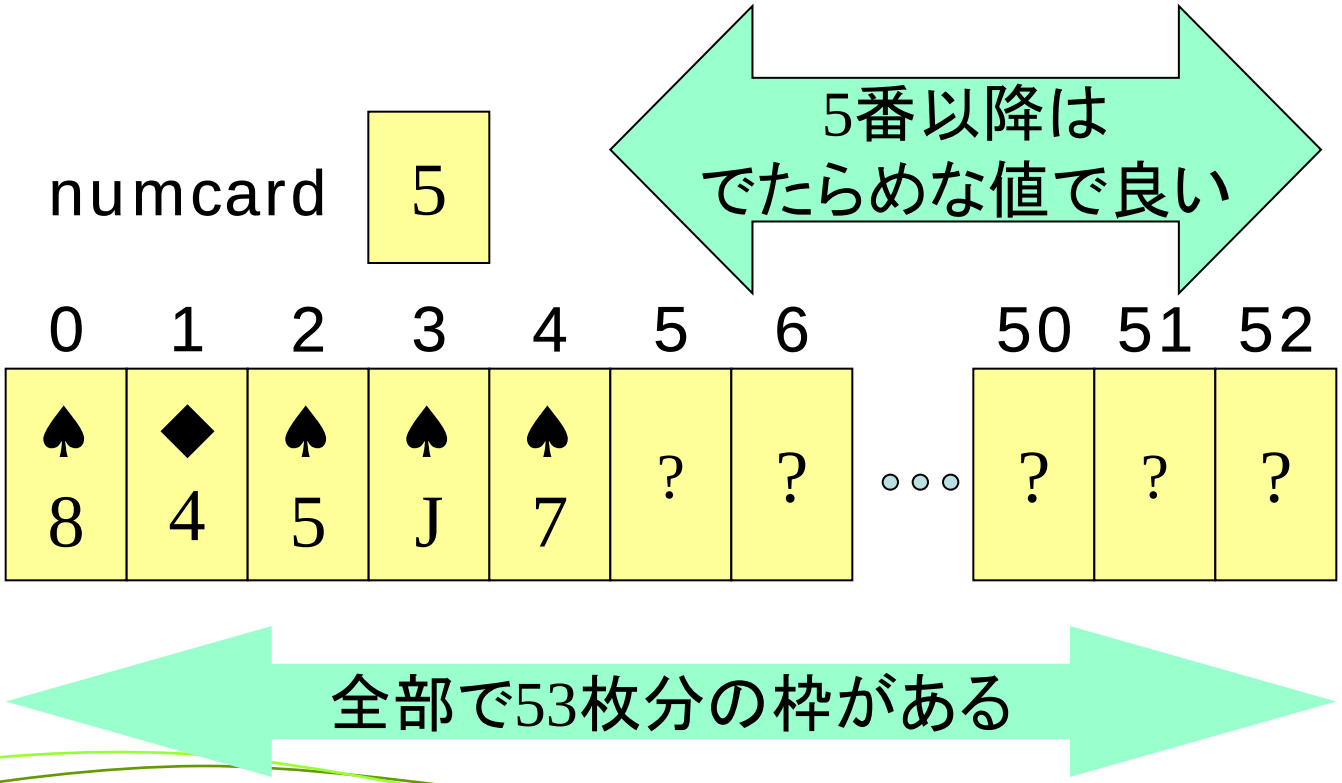
```
enum {
        SUIT_SPADE,
        SUIT_DIAMOND,
        SUIT_HEART,
        SUIT_CLUB,
        SUIT_JOKER
};
```

- 番号を陽に指定することも出来る

```
enum {
        SUIT_SPADE = 1,
        ...
        SUIT_JOKER = 8,
```

トランプカード集合型CardSet

- 配列の0番からnumcard – 1番までにカードのデータが入る



トランプカード集合型CardSet

- カードは1デッキ（52枚と Joker1枚）しか使わないとする

```
class CardSet {
public:
        static const int maxnumcard = 53;   カードの総数
private:
        int numcard;                         現在のカード数
        Card cdat[maxnumcard];               カードのデータ
        int locate(…);                       場所探し
public:
        CardSet(void)     { makeempty(); }       デフォルトコンストラクタ
        bool isempty(void);                  空集合か否かの判定
        bool insert(Card newcard);           カードを1枚入れる
        bool remove(Card target);            targetのカードを1枚除く
        bool remove(int num);                数字がnumのカードを1枚除く
};
```

演習6(トランプ＆トランプカード型)

- ソースファイル ~sin/C++

(1) プログラムをコンパイルし，動作を確かめよ

- main() はカードとして不正な値が入力されるまでカードをよみ続け，最後にカードの集合を出力

(2) 関数removeを追加せよ

※ 実現後に関数 CardSet::pickup() 内のコメントははずす

- 作成した main() では remove 手続も試せ

演習6(2)のヒント:removeの実現法

- CardSet::locate()で除くカードの場所を探す
  - 探し方は2種類用意されている
- 見つからなかったらtrueを返す
- 見つかったら配列をうまく直して，numcardも更新する

クラスを元により大きなクラスを作る

- 手持ちのカード型をもとに，ババ抜きの途中の状態を表す型を作る

ババ抜き状態型BabaState

- 全員の手を記憶

```
class BabaState {
puclic:
        const int numplayer = 5;          プレーヤ数
private:
        CardSet hand[numplayer];          各プレーヤの持ち手
public:
        BabaState(void)           { reset(); }デフォルトコンストラクタ
        void reset(void);         最初にカードを配った状態にする
        bool move(int from, int to);      fromからtoへカードの移動
};
```

演習7(ババ抜き状態型)

- ソースファイル ~sin/C++

(1) BabaState::move() を実装せよ

演習7のヒント:moveの実現法

① from番のプレーヤの手からカードを取る
② to番のプレーヤの手の中で同じ番号のカードを探す
③ 同じ番号のカードが見つかれば，２枚とも捨てる
④ 同じ番号のカードが見つからなければ，to番のプレーヤーの手に加える

習うより慣れろ

- 次の課題のうち少なくとも1つに取り組む
  - 演習8：Cardクラスを別の方法で実現せよ
  - 演習9：BabaStateクラスを別の方法で実現せよ
  - 発展演習：ババ抜きを人間が関与できるようにせよ

作り方の鉄則

- いきなり考えなしにメンバ変数を操作するな！
- 不便だからとprivateをpublicにするのは最低！
- 適切なクラスから必要な操作ができるようにメンバ関数を作れ！

演習8・ヒント

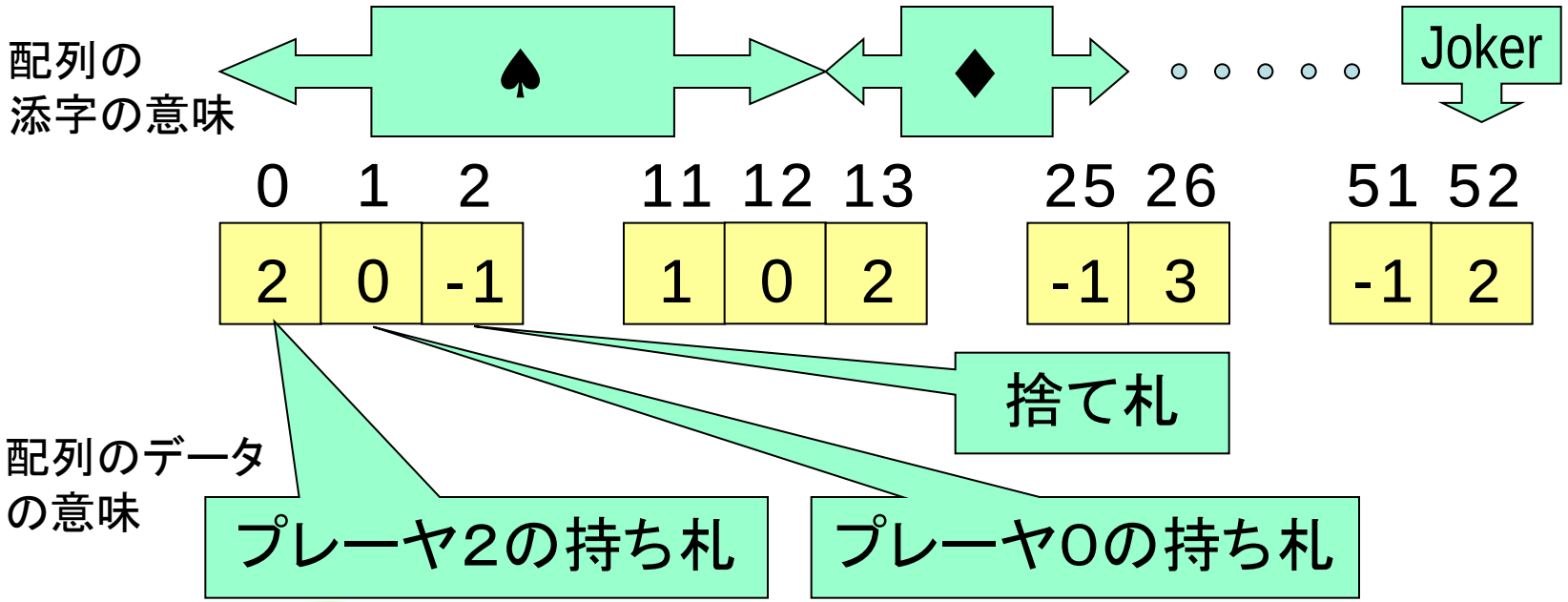
- Cardクラスのメンバ変数をint １つにせよ.
  - たとえば，すべてのメンバ関数が13をダイヤの1として扱えば，全体は矛盾なく動作する.
- Cardクラスのメンバ関数をこれまでとまったく同じ外部仕様（インターフェース：関数の引数の与え方と返り値の使い方）にせよ.
  - Cardクラスを入れ替えてもそのまま動くはず.

演習8・ヒント

- 相手の手から「？枚目のカードを取る」は，CardSet::pickup()で可能である．
  - CardSet::pickup() をメインから直接アクセスするのは×
- 「何枚目か」を指定してカードを移動するメンバ関数をBabaStateクラスに作れ．
- 仮想プレーヤはランダムに，自分は「何枚目か」を入力しカードを取れるようにせよ．
- 相手の現在の枚数が知りたければ適切なメンバ関数を作れ．

演習9・ヒント

- BabaStateクラスは，CardSetクラスを用いずに以下のようにしても実現できる.



ノルマ

- 今週のノルマ
  - 演習7まで終了すること
- 来週のノルマ
  - 最後まで終了すること(レポート提出を含む)
- 各ノルマを満たせば，そのあとは自由
  - TAなどに確認してもらうこと
  - 次回以降の内容を自習・実装してもよい
  - レポートの準備をしてもよい
  - 帰ることも妨げない

知能情報工学実験演習II
C++演習（3）

下薗／中村

TA：佐藤誠／古野雅大

C＋＋演習　レポートと提出

1. 実験3回目の時間終了時に，
   佐藤TA，古野TAいずれかに規定要素を確認してもらい，表紙に確認の日付と時間，担当TA名の記入をうける．未完成の場合も含む．
2. 担当TAがレポートを預かれば，提出．
3. 再提出は，同じ担当TAに，時間や場所の約束をとりつけ行う．表紙は1で記入を受けたものであること．確認のたびに日時を記入する．

- 初回の提出期限：
  次週の最初の（授業）日・・・来週月曜日

レポートの規定要素(書き方1)

- 用紙: A4, 縦置き, 横書き
- 余白: 上下 25～35mm, 左右 22 ～ 30mm
- 行数, 行字数, 文字サイズ:
  30行/ページ 程度(行間1.25 ～ 1.5 行),
  40～45文字/一行 程度(文字大きさ 11 ～ 12pt), 表題は 18pt 程度
- 表紙にテーマ名, 学生番号と氏名, 実験日(3回分)

レポートの規定要素（書き方2）

- 文字スタイル：
  本文は明朝体，タイトルや節見出しはゴシック
  英数文字は半角．プログラムリストやコンソール出力は等幅（非プロポーショナル）体（Courier, Lucida, Consoals など）
- ページ番号：
  フッタ中央またはヘッダ右端に通しで
- 左上をステープラ等でとめる

レポートの規定要素(報告内容)

- 行った演習の結果，演習の問いに対する答えが示されていること.
  - プログラムのソースコードは電子的に別提出し，レポートではそれを前提に引用は必要最小限にとどめる
  - 複数の演習の結果を一組のソースコード等で示す場合は，わかるように記述する
- コンストラクタのはたらきと動作，およびポリモルフィズムについて，自身の実験と結果によって考察すること.

レポートの規定要素（実装結果）

- 演習7まで，および 8 または 9 のヘッダ／ソースコードのファイルを電子的に提出
  - コンパイルエラーがないこと，正しく動作することを確認のうえ，すべてのファイルを
  - ~sin/Studex2014/[学生番号]
  - ディレクトリ　にコピーする

悪質なレポートについて

- 間違いに気づかない，自分で説明できないなど，意味を理解せずに写す行為（通称「コピペ」）やその疑いが否定できない場合，再提出や即不合格
- 演習7までを完了していない，演習8以降の取り組みが不十分，など授業時間中に演習を行ったと確認できないレポートは，再提出や即不合格
- 電子提出のプログラムについても同様

おまけ：プログラミングを学ぶコツ

- 一番楽に作れる方法を，まずよく考えること
  - アイディアが実現可能かどうかを調べたり聞いたり
- 書くプログラム，文書の作成は，カット＆ペースト，検索と置換で効率的に
  - 生きた化石 emacs エディタにもカット／コピー＆ペースト，検索 search，置換 replace はある
- デバッガ（gdb）がすこしでも使えると便利
  - 一行ごとに実行，変数の内容を表示，が可能
  - できれば eclipse など高機能エディタやデバッガが統合された開発環境を．

おまけ：Makefile の使い方

- コンパイルのコマンドを登録する

  Makefile に，たとえば以下のエントリーを追加する

```
myexample:
        c++ -Wall -g mycardset.cc mycardset_example.cc -o mycommand
```

  と，main 関数がある mycardset_example.cc ，その関連ファイル mycardset.h, mycardset.cc を

```
make myexample
```

  で mycommand 実行ファイルとしてコンパイルする.

- なお Makefile に登録済みのエントリーは，.cc ファイルを別々に細かくコンパイルをするようになっています