

知能情報工学実験演習II

C++演習

下園／中村

TA：原田、井上

C++演習

演習の目的と内容

- 前半：C++ 演習
 - C言語の復習も兼ねる
 - データの抽象化を中心に
- 後半：グループ演習
 - 前半使ったプログラムを再利用しつつ、グループでカードゲームの思考ルーチンを作成

※この資料は以下の場所にもあります

– /home/i/sin/C++/C++2012.pdf

レポートの作成、提出

- C++ 演習

- 各自が課題をこなし、レポートを作成し提出
- 3回目、終了後～翌日正午までに提出

※詳細は来週

- グループ演習

- 最終回の翌週水曜までに提出
- 詳細は後半開始後に説明します
- レポートはグループで作成するが、各自で個人評価票を提出

知能情報工学実験演習II

C++演習（第1回）

C++演習

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訓練。
 - シンプルさを追及する。
 - データ型の「意味」を意識して使う。
- C言語でのデータ型への意味づけ
 - 型に自分で名前をつけられる
typedef
 - 構造をもつデータ型を定義できる
struct（構造体）, union（共用体）

第1回目の内容（2）

- C++言語での型への意味づけサポート
 - struct と class 二種類の構造体宣言
データ隠ぺいが利用可能
 - データのライフサイクル（誕生、消滅）
の考え方を実現
 - 型の定義に操作関数を含めることができ、
同名の関数を型により区別可能

そのほか、列挙型 enum、定数宣言 const、名前空間・・・などなど

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

- /home/i/sin/C++
 - 演習1
 - Makefile
 - vector2c.h
 - vector2c.c
 - vec2ex1c.c
 - 演習2以降
 - vector2p.h
 - vector2p.cc
 - vec2ex1p.cc
 - make vec2ex1c もしくは make vec2ex1p 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);
```

- 操作関数を作る手間を惜しむな

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

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にはいつ、どこ、だれでもアクセスできる
 2. 異なる対象の類似の操作にすべて別の名前をつける必要がある
例) addv2, addv3, addmatrix
例) set_add(myset, anelement)

- クラスと構造体：データと操作関数をまとめたもの

```
class Vector2 {  
    private:   メンバ変数は普通private  
        double x;  
        double y;  
    public:   メンバ関数は普通public  
        Vector2(...) { }   コンストラクタ (データを作るとき実行)  
        Vector2(double x0, double y0); 引数付コンストラクタ  
        Vector2 add(Vector2 u);  
                                メンバ関数からはメンバ変数にアクセス可  
                                メンバ関数はオブジェクトに作用する  
};
```

※Cとの違い

- Struct/class 型を型名として使う場合に typedef 宣言が不要.

```
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
※ 演算子 + の定義を書くこともできる

- 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

public:

```
Vector2() { }
```

```
Vector2(double x0, double y0)
```

```
{
```

```
    x = x0;
```

```
    y = y0;
```

```
    printf("引数あり");
```

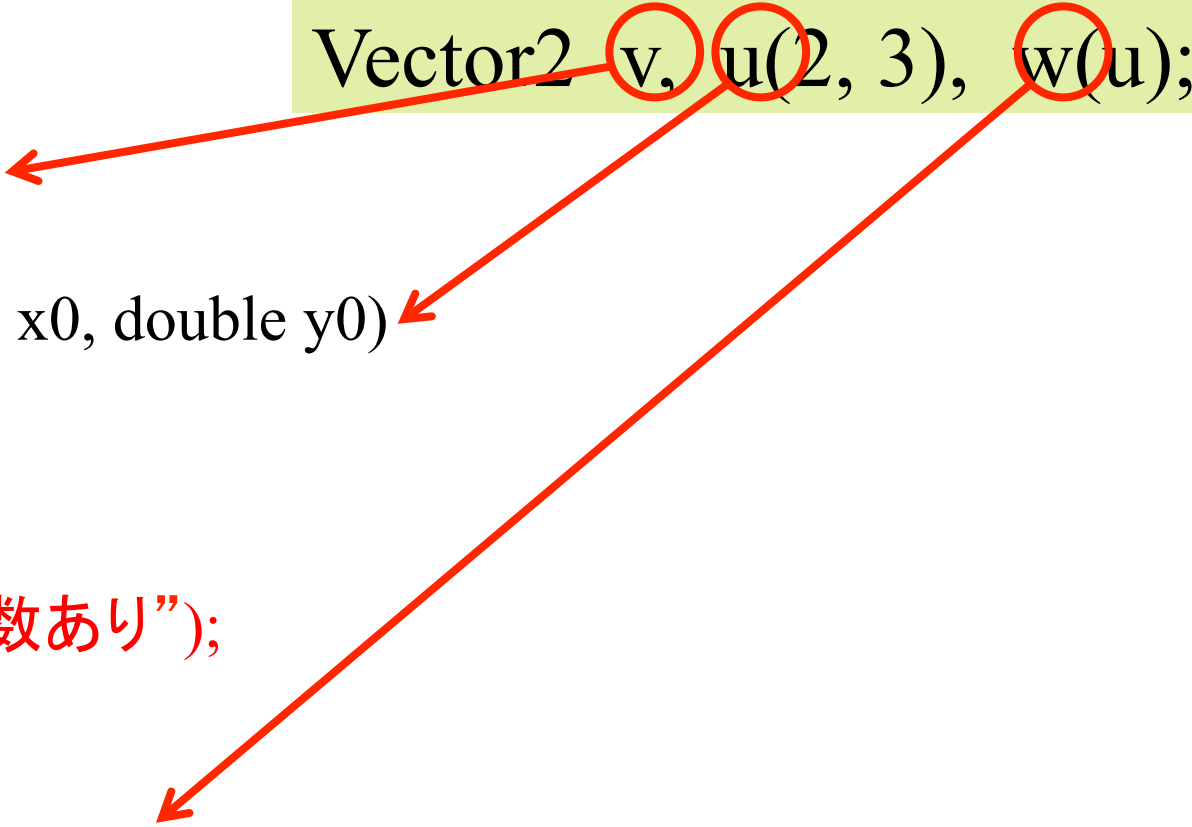
```
}
```

```
Vector2(Vector2 &v) {
```

```
    ....
```

```
} 関数の引数(値渡し)のときも呼ばれる
```

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



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まで終われば，そのあとは自由
 - TA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
 - みつからないよう帰ることは妨げない
- 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）を示すだけではダメ

%a.out

デフォルトコンストラクタが呼ばれました

コピーコンストラクタが呼ばれました

...

以上の結果から、正しくコンストラクタが動作しているといえる

— いえない。

そもそも、なにを正しいと言っているのか。。

- 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）

下園／中村

TA：原田、井上

C++演習

集合の実現

- トランプカードの手はカードの集合として表せる
- 集合≠リスト (列) ≠線形リスト
 - どのようなデータ構造が良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
 - 動的データ構造は難しいことも一つの理由

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)    { }  
    void set(int st, int num)  
    bool equal(Card tgt);  
    int gnumber(void);  
    bool scan(void);  
};
```

デフォルトコンストラクタ
スートと番号をセット
同じか否かの判定
アクセサ
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

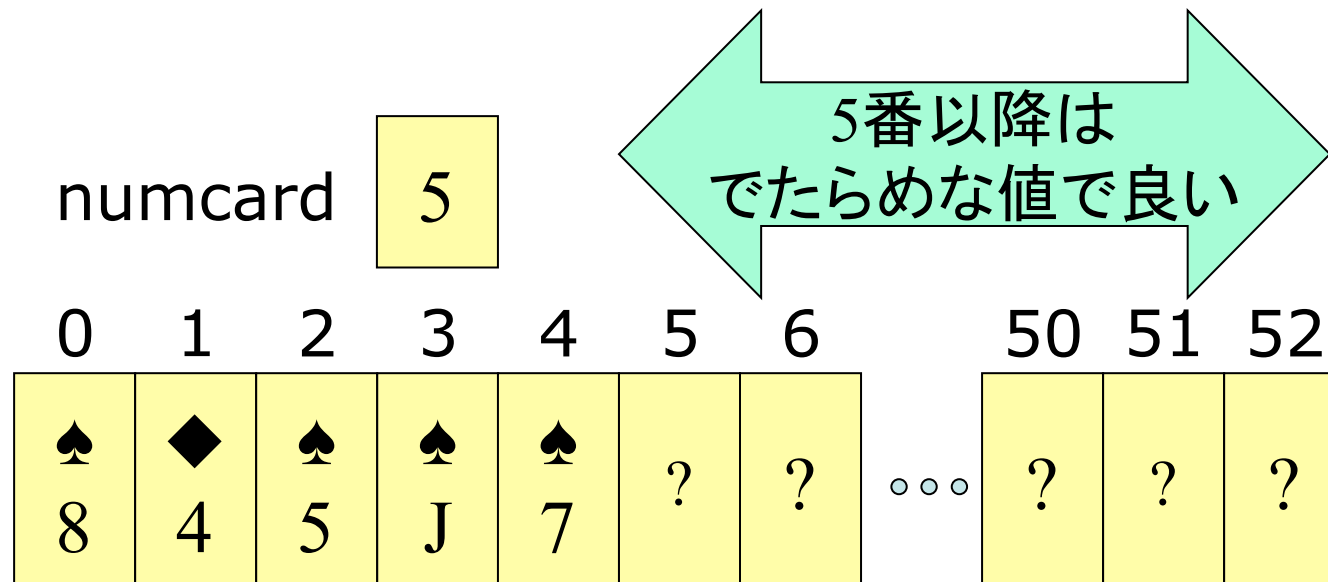
- 番号が0から自動的に振られる

```
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

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（Joker 1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 /home/i/shimada/C++
 - (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
 - main()はおかしなカードが入力されるまで, カードを入力し続け, 最後にカードの集合を出力する
 - (2) 関数removeを追加せよ(実現後に関数 `CardSet::pickup()` 内のコメントを外せ)
 - 新しいmain()ではremove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，ババ抜きの中の状態を表す型を作る

ババ抜き状態型BabaState

- 全員の手を覚えておく

```
class BabaState {  
public:  
    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 /user/i/shimada/C++
(1) BabaState::move()を実現せよ

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ければ, 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番のプレーヤの手に加える

習うより慣れる

- 次の課題のうち少なくとも1つに取り組む
 - 演習8：ババ抜きを人間が関与できるようにせよ
 - 演習9：Cardクラスの実現方法を変更せよ
 - 演習10：BabaStateクラスの実現方法を変更せよ

作り方の鉄則

- いきなり考えなしにメンバ変数进行操作するな！
- 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

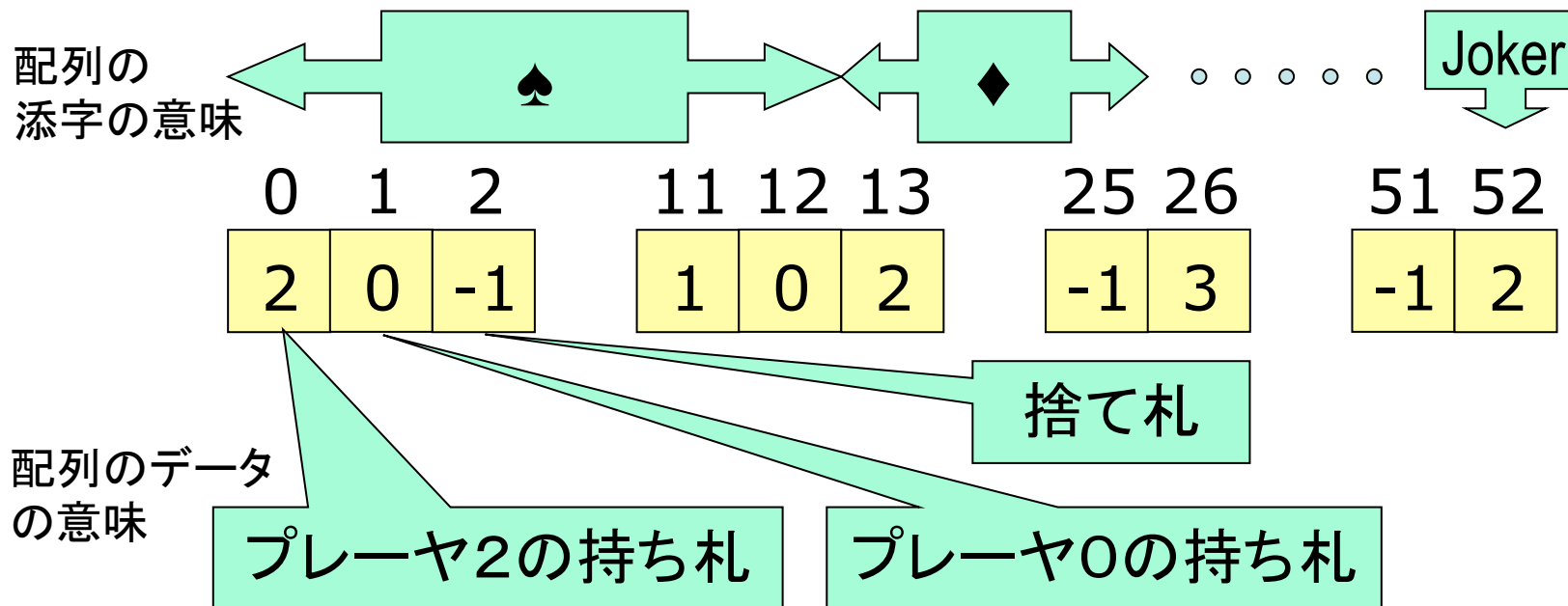
- 相手の手から「？枚目のカードを取る」は、`CardSet::pickup()`で可能である。
 - `CardSet::pickup()` をメインから直接アクセスするのは×
- 「何枚目か」を指定してカードを移動するメンバ関数を `BabaState` クラスに作れ。
- 仮想プレイヤーはランダムに、自分は「何枚目か」を入力しカードを取れるようにせよ。
- 相手の現在の枚数が知りたければ適切なメンバ関数を作れ。

演習9・ヒント

- 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10・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

演習3について(コンストラクタ)：続き

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
```

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

function(Vector2
&i)

a = i->x;

a = i.x;

レポートについて

- 個人演習のレポート
 - 締切： 月 日（水曜日） 17:00
 - 場所： E レポートボックス
- レポートに不備がある場合は再提出を指示する。
- 提出遅れ、再提出で指示に従っていない場合、「不可」とすることがある。

レポートの書式

- 以下の書式を守らない場合は、それだけで再レポートの対象となる.
- 紙での提出
 - 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.
 - 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, レポート提出日を明記すること
 - コンストラクタおよびポリモルフィズムについて, 自身の実行結果をもとにA4サイズ5枚以内で説明しなさい
 - テキストの丸写しはそれだけで再レポートの対象となる
- プログラムの提出
 - 演習8~10までのうち, 行ったものを電子的に提出しなさい
 - コンパイル, 実行可能なもののみ
 - 正しく動作しないものは評価しない