情報工学演習I
第１１回

C++の演習3（関数のオーバロード）

順番を入れ替えました

授業の予定（後半）

#	月日	内容	担当者
7	11月13日	C言語の演習4 (ポインタの演算，列挙型)	内海
8	11月20日	C言語の演習課題	内海
9	11月27日	C++の演習1（クラス）	岩村
10	12月 4日	C++の演習2（クラスの継承）	岩村 (代理：谷川)
11	12月11日	C++の演習3 (関数のオーバロード)	岩村
12	12月18日	C++の演習4（インライン展開）	岩村
13	1月 8日	C++の演習5（仮想関数）	岩村
14	1月15日	C++の演習課題	谷川
15	1月22日	総合演習	谷川

今日の内容

- 第8回演習課題の解説
- 関数のオーバロード
- デフォルト引数
- コピーコンストラクタ

第8回演習課題の解説

第８回演習課題解説（１）

- ２から１０００までの整数のうち，素数のみを表示するプログラムを作れ

- 素数とは？
  - 1 と自分自身以外に正の約数を持たない、1 でない自然数（正の整数）のこと
- N が素数である
  - 2,3,…,N でN を割ったとき、N のみで割り切れる

第８回演習課題解説（２）

- キーボードから入力された整数が素数である場合，素数であることを出力するプログラムを作れ．

- 素数の判別方法は課題１と同じ
- キーボード入力された整数を利用

  キーボード入力された整数を変数に代入する関数

  scanf(“%d”, &n);

& を忘れないように

第８回演習課題解説（３）

- char型の2次元配列を以下のように初期化する.

  char moji[][256]={ “king”, “kind”, “kinder”, “kindergarten”};

  このとき，これらの文字列を辞書に出てくる順にソートし，出力せよ．ただし，ソートのアルゴリズムを用いてソートすること.

- ソートは第２回演習課題－３のものを利用
- 文字列を比較する関数strcmp の利用

  int strcmp(const char *s1, const char *s2);
  - s1==s2: 0
  - s1 ＞ s2(s1 の方が後に辞書に出てくる):正の値
  - s1 ＜ s2(s2 の方が後に辞書に出てくる):負の値

第８回演習課題解説（４）

- クラスにN人の人がいて，クラス内に同じ誕生日の人が2人以上いる確率を求め，N が何人よりも大きくなると，その確率が50%以上になるか求めるプログラムを作れ.

- （求める確率）
  ＝１－（N人のクラス内に同じ誕生日の人がいない確率）
- N人のクラス内に同じ誕生日の人がいない確率

$$\frac{364}{365}\times\frac{363}{365}\times\cdots\times\frac{365-(N-1)}{365}$$

- N=2,3,…と計算し、求める確率が50%以上になるまで続ける
- ２３人以上で５０％以上となる

関数のオーバロード

関数のオーバーロード

ex12_overload.cc

```
#include <iostream>
using namespace std;

unsigned int
hiku(unsigned int x,
unsigned int y) {
  return x-y;
}

int hiku(int x, int y) {
  return x-y;
}

int main() {
  unsigned int a=60;
  unsigned int b=200;

  cout << "In case of unsigned int: "
<< hiku(a, b) << endl;
  cout << "In case of int: " <<
hiku((int)a, (int)b) << endl;

  return 0;
}
```

unsigned int型

unsigned int型の呼び出し

int型の定義

int型の呼び出し

関数のオーバーロード

- C++では、同じ名前を持つ関数を複数定義できるようになった

```
unsigned int hiku(unsigned int x, unsigned int y) {
```

```
int hiku(int x, int y) {
```

引数で区別

- C言語の場合は、同じ名前の関数を定義すると怒られる

関数のオーバーロード

- ex3_w_class2.cc に出てきたコンストラクタとデフォルトコンストラクタも関数のオーバーロード

```
// コンストラクタ
  Account(string _name, int _balance) {
    name = _name; // 名前を初期化
    balance = _balance; // 残高を初期化
  }
```

```
// デフォルトコンストラクタの例
  Account() {
}
```

デフォルト引数

球の体積の計算

ex13_default_arg.cc

```
#include <iostream>
#include <math.h> // πを使うため
using namespace std;

double sph_vol(double r=1.0) { // 球の体積
  return 4.0 / 3.0 * M_PI * r * r * r;
}

int main() {
  cout << "With argument (r=2.0): " << sph_vol(2.0) << endl;
  cout << "With argument (r=1.0): " << sph_vol(1.0) << endl;
  cout << "Without argument: " << sph_vol() << endl;

  return 0;
}
```

デフォルト引数

- 引数を指定しないときの引数の値が指定できる

```
double sph_vol(double r=1.0) { // 球の体積
```

引数を指定した場合

```
  cout << "With argument (r=2.0): " << sph_vol(2.0) << endl;
  cout << "With argument (r=1.0): " << sph_vol(1.0) << endl;
  cout << "Without argument: " << sph_vol() << endl;
```

デフォルト引数

- デフォルト引数の後に普通の引数は取れない

```
double sph_vol(double r=1.0, int n) { // 駄目な例
```

コピーコンストラクタ

オブジェクトのコピー

ex14_copy_constructor1.cc

```
#include <iostream>
using namespace std;

class test {
public:
  int val;
};

int main() {
  test a; // オブジェクトaを初期化
  a.val = 10; // a.valに10を代入
  cout << "a.val = " << a.val << endl;
  test b = a; // オブジェクトbを，aを代入して初期化
  a.val = 20; // コピー後，a.valに20を代入してみる
  cout << "b.val = " << b.val << endl;

  a.val = 10; // a.valに10を代入
  test c; // オブジェクトcを初期化
  c = a; // cにaを代入
  a.val = 30; // コピー後，a.valに30を代入してみる
  cout << "c.val = " << c.val << endl;

  return 0;
}
```

実行例

```
$ ./a.exe
a.val = 10
b.val = 10
c.val = 10
```

オブジェクトaがbとcにコピーされた

コピー後にa.valの値を変更してもb.valとc.valには反映されない

オブジェクトのコピー

- int型の場合

  int a=3;

  int b = a;

- オブジェクトの場合
  - Case 1:

    test a;

    a.val = 10;

    test b=a;

  - Case 2:

    test a; test b;

    a.val = 10;

    b=a;

クラスtest

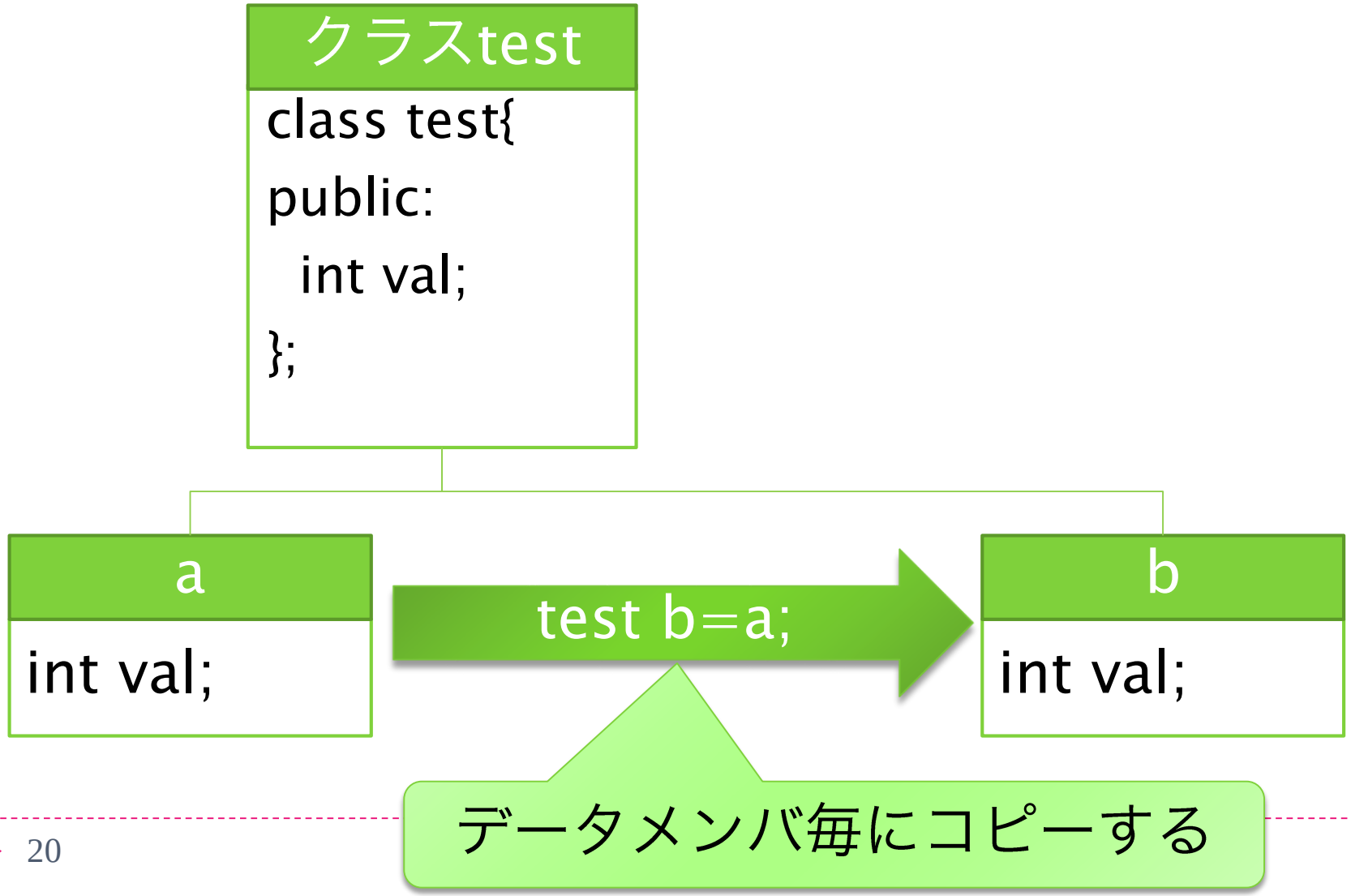
```
class test {
public:
  int val;
};
```

オブジェクトbの定義とコピーが同時

オブジェクトbの定義とコピーが別

オブジェクトのコピー

- C++でオブジェクトをコピーする仕組み



オブジェクトのコピーがうまくいかない例

ex15_copy_constructor2.cc

ex11_constructor.ccがベース

```
#include <iostream>
using namespace std;

// オブジェクト作成時に動的にメモリを確保し，破棄時にメモリを開放するクラス
class dyn_mem {
public:
  int *array; // 配列用のポインタ
  int size; // 配列の大きさ

  dyn_mem() { // デフォルトコンストラクタ
    size = 0; // 配列のサイズが指定されないときは，配列のメモリを確保しない
  }
```

オブジェクトのコピーがうまくいかない例 続き

ex15_copy_constructor2.cc

```
 dyn_mem(int _size) { // コンストラクタ
    size = _size;
    array = new int[size]; // 配列のメモリ確保
  }


 ~dyn_mem() { // デストラクタ
    if (size!=0) { // 配列にメモリが確保されている場合のみ, メモリを開放する
      delete [] array;
      size = 0;
    }
  }
};
```

オブジェクトのコピーがうまくいかない例 続き

ex15_copy_constructor2.cc

```
int main() {
 dyn_mem dm1(3); // オブジェクトの作成(大きさが3の配列のメモリを確保)
  dm1.array[0] = 10;
  dm1.array[1] = 3;
  dm1.array[2] = 8;

  dyn_mem dm2(3); // オブジェクトの作成(大きさが3の配列のメモリを確保)
  dm2 = dm1; // オブジェクト dm1 を dm2 にコピー

  for (int i=0; i<dm2.size; i++) { // dm2の配列を表示
    cout << "dm2.array[" << i << "] = " << dm2.array[i] << endl;
  }
```

オブジェクトのコピーがうまくいかない例 続き

ex15_copy_constructor2.cc

```
  cout << "----------" << endl;

  dm1.array[0] = -1; // dm1 のメンバの値を変更してみる
  cout << "dm1.array[0] = " << dm1.array[0] << endl;
  cout << "dm2.array[0] = " << dm2.array[0] << endl;
  return 0;
}
```

実行例

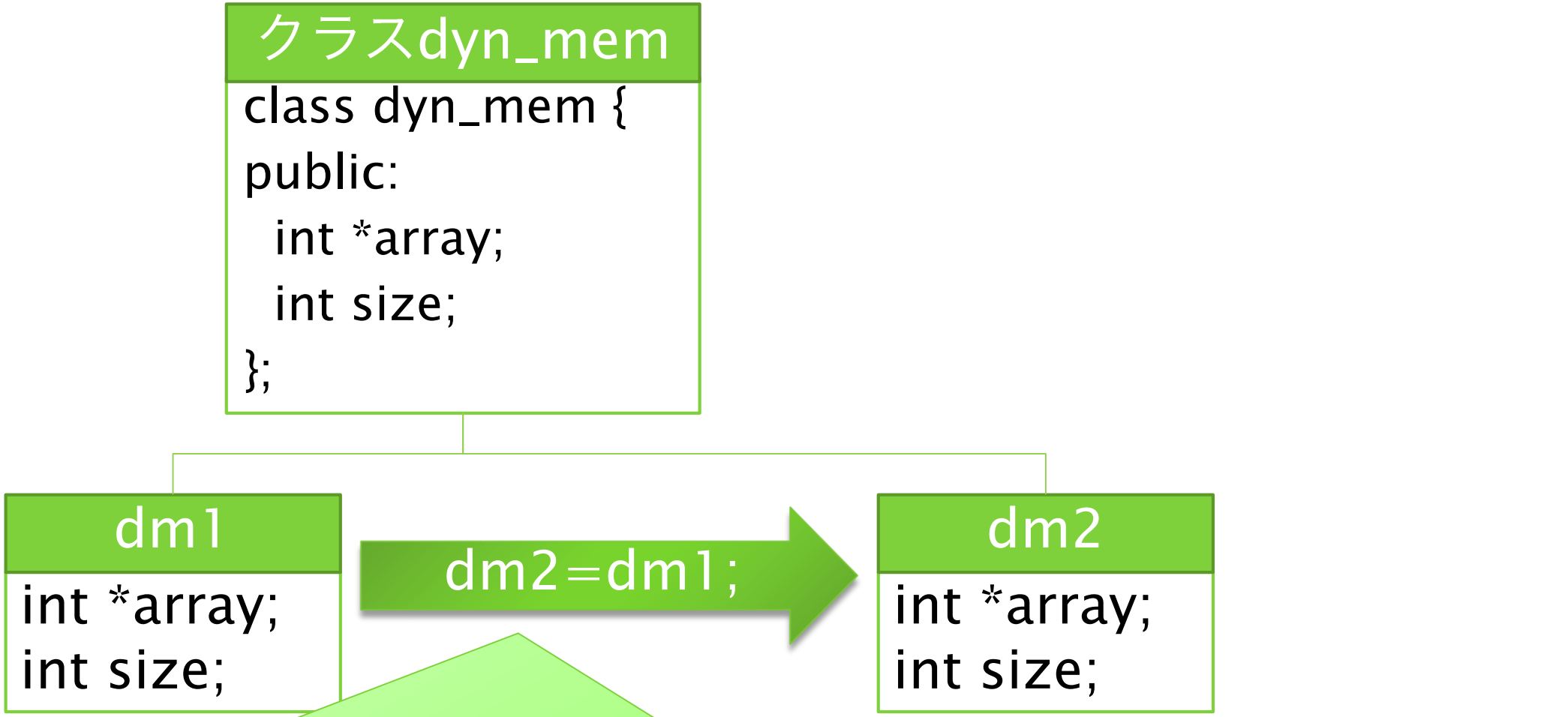
```
$ ./a.exe
dm2.array[0] = 10
dm2.array[1] = 3
dm2.array[2] = 8
----------
dm1.array[0] = -1
dm2.array[0] = -1
Aborted (コアダンプ)
```

dm1.valの配列の値を変えると、dm2.valの配列の値も変わった（オブジェクトは別だけど、配列は同じ）

既に解放したメモリをもう一度解放しようとしたのでエラーが起きた（dm1とdm2で配列を共有しているので、同じメモリが２つのデストラクタで解放された）

何が起きているのか？

- C++でオブジェクトをコピーする仕組み



- データメンバ毎にコピーする
（ただし浅いコピー：ポインタは同じアドレスを指すだけ）

オブジェクトのコピーがうまくいく例

ex16_copy_constructor3.cc

ex11_constructor.ccがベース

```
#include <iostream>
using namespace std;

// オブジェクト作成時に動的にメモリを確保し，破棄時にメモリを開放するクラス
class dyn_mem {
public:
  int *array; // 配列用のポインタ
  int size; // 配列の大きさ

  dyn_mem() { // デフォルトコンストラクタ
    size = 0; // 配列のサイズが指定されないときは，配列のメモリを確保しない
  }
```

オブジェクトのコピーがうまくいく例

続き

ex16_copy_constructor3.cc

```
dyn_mem(int _size) { // コンストラクタ
  size = _size;
  array = new int[size]; // 配列のメモリ確保
}

dyn_mem(dyn_mem &obj) { // コピーコンストラクタ
  size = obj.size; // サイズをコピー
  array = new int[size]; // 配列のメモリ確保
  for (int i=0; i<size; i++) { // 配列をコピー
    array[i] = obj.array[i];
  }
}
```

この部分が新しく追加された

オブジェクトのコピーがうまくいく例

続き

ex16_copy_constructor3.cc

```
 dyn_mem(int _size) { // コンストラクタ
    size = _size;
    array = new int[size]; // 配列のメモリ確保
  }


 ~dyn_mem() { // デストラクタ
    if (size!=0) { // 配列にメモリが確保されている場合のみ, メモリを開放する
      delete [] array;
      size = 0;
    }
  }
};
```

オブジェクトのコピーがうまくいく例

続き

ex16_copy_constructor3.cc

```
int main() {
 dyn_mem dm1(3); // オブジェクトの作成(大きさが3の配列のメモリを確保)
  dm1.array[0] = 10;
  dm1.array[1] = 3;
  dm1.array[2] = 8;

  dyn_mem dm2 = dm1; // dm1 の中身をコピーした dm2 を作成

  for (int i=0; i<dm2.size; i++) { // dm2の配列を表示
    cout << "dm2.array[" << i << "] = " << dm2.array[i] << endl;
  }
```

この部分が変更された

オブジェクトのコピーがうまくいく例

続き

ex16_copy_constructor3.cc

```
  cout << "----------" << endl;

  dm1.array[0] = -1; // dm1 のメンバの値を変更してみる
  cout << "dm1.array[0] = " << dm1.array[0] << endl;
  cout << "dm2.array[0] = " << dm2.array[0] << endl;
  return 0;
}
```

実行例

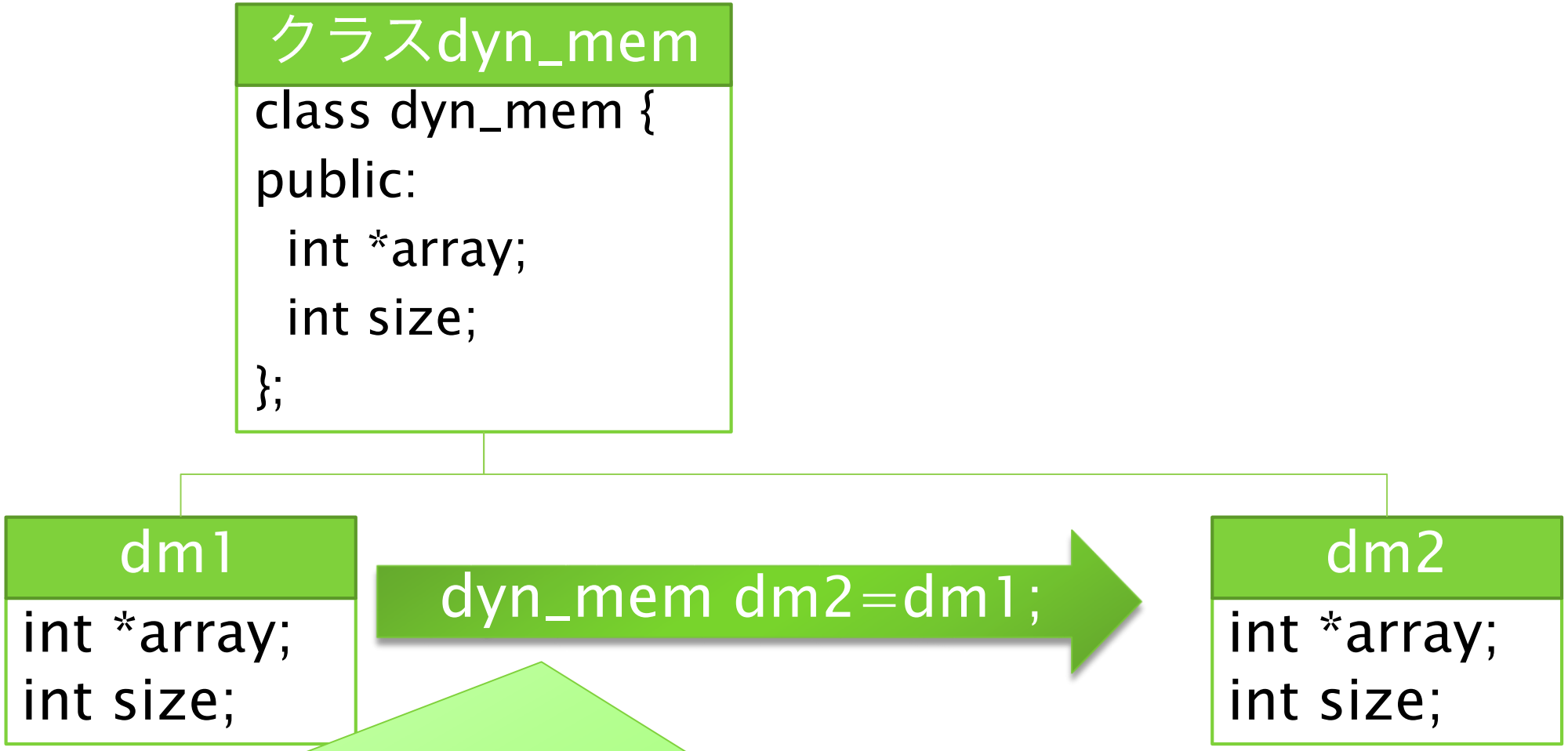
```
$ ./a.exe
dm2.array[0] = 10
dm2.array[1] = 3
dm2.array[2] = 8
----------
dm1.array[0] = -1
dm2.array[0] = 10
```

dm1.valの配列の値を変えても、
dm2.valの配列の値は変わらない
（別オブジェクトで配列も別）

メモリ確保と解放の対応が取れているため、エラーが起きなくなった

何が起きているのか？

- C++でオブジェクトをコピーする仕組み



コピーコンストラクタ

- オブジェクト（データメンバ）のコピー方法の定義

クラス名 (クラス名 &obj)
という書式

```
dyn_mem(dyn_mem &obj) { // コピーコンストラクタ
  size = obj.size; // サイズをコピー
  array = new int[size]; // 配列のメモリ確保
  for (int i=0; i<size; i++) { // 配列をコピー
    array[i] = obj.array[i];
  }
}
```

自分で配列のメモリを確保して、
配列の要素毎にコピー

コピーコンストラクタが呼ばれる場合、呼ばれない場合（初期化と代入）

- 初期化
  - dyn_mem dm1(3);
  - dyn_mem dm2(dm1);
  - dyn_mem dm2 = dm1;
- 代入
  - dm2 = dm1;

コピーコンストラクタは
初期化の時だけ呼ばれる

代入では呼ばれないので注意！

演習課題

第１０回演習課題（４）

前回の積み残し
第11回のところに提出
（アップロード）する

- 4. 以下のプログラムを作成する
  - 以下の仕様を満たすクラスを作れ
    - オブジェクト作成時に整数kを引き数とし、newを使って(k+1)*(k+1)の2次元配列を確保する
    - オブジェクト破棄時にdeleteでメモリを解放する
    - 任意の座標(x,y)に値を代入できる
    - 任意の座標(x,y)の値を表示できる

第１０回演習課題（４）続き

- プログラムを実行すると、キーボードから整数kを入力することを求められる
- このkを引数として、前述のクラスのオブジェクトを作る
- (0,0)から(k−1,k−1)までには、乱数で適当な値を入れる
- 列毎の和を(k,0)から(k,k−1)に入れる
- 行毎の和を(0,k)から(k−1,k)に入れる
- (0,0)から(k−1,k−1)までの合計を(k,k)に入れる
- (0,0)から(k,k)までを出力した後、オブジェクトを破棄する

k=4のとき

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

行毎の和

合計

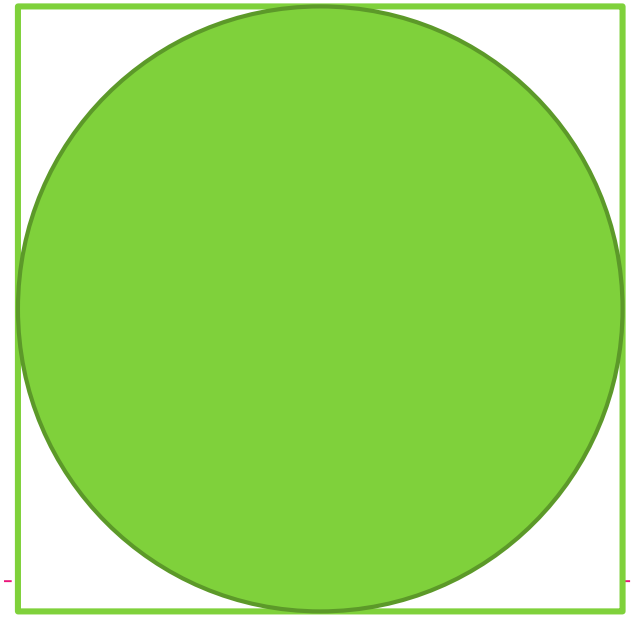
列毎の和

第１１回演習課題（１）

- １. 乱数を使って円周率πを求めよ。
  - 例えば、x座標とy座標を乱数で生成して、点を発生させる
  - 1x1の正方形と直径１の円を考えて、発生させた点の数と、円の内側にできた点の数を比べると、円周率が求まる
  - 発生させた点の数と求まった円周率の関係を図示するとなお良い

使った手法をTAに分かるように説明する

何かを参考にした場合は、その旨を述べる。書いてない場合は０点



第１１回演習課題（2）

- 2. 以下の条件を満たすプログラムを作成せよ。
  - 引数として自然数を１つとり、その数が完全数であるかどうかを判定する関数を持つ。
  - 1から100以上の適当な値までの自然数が完全数かどうかを調べて、完全数を表示する。

完全数

その数自身を除く約数の和が、その数自身と等しい自然数
（例：6＝1＋2＋3、
28＝1＋2＋4＋7＋14）

提出に関して

- 提出するもの
  - ソースファイル(.ccまたは.cpp ファイル)
    - ファイル名はkadai1211_学籍番号_課題番号.cc（.cpp）
    - (Visual Studioの場合)
      ファイル名はkadai1211_学籍番号_課題番号_v.cc
      （.cpp）
  - 実行結果の出力と講義に関するコメント
    - .txt ファイルで、学籍番号、氏名を含む
    - ファイル名はreport1211_学籍番号.txt とする

提出に関して（続き）

- 提出期限
  - 1月8日（水）　００：００
- 提出方法
  - 授業支援システムから提出
- 注意点
  - ファイル名の命名規則が間違っているものは採点しない
  - コンパイルの通らないものは採点しない