プログラミング AII

(知能情報システムプログラム2年)

(協創経営プログラム　　3年)

(創生学部知能情報システム領域学習科目パッケージ2年)

0 ガイダンス

0–1 受講に当っての留意事項

旧カリキュラムにおける位置付け

- 平成28年度以前に入学した情報工学科受講生については、
  (合格したら)「プログラミングI」に読み替えられる。

必要な予備知識

- 1年次第3~4タームの「プログラミング基礎I, II」と
  前タームの「プログラミング AI」を既に履修して
  (ある程度の理解をして)いることを前提に話を進める。

授業の進め方

- 講義と演習／実習を交互に行う。
  ⇨ 基本的には、3限は講義、4限は演習／実習。

- C++プログラムの書き方の詳細は参考書等に書かれているので、授業では細かい話はしない。

  ⇨

  授業に出席するだけではこの授業の単位を取れないことを自覚して下さい。
  ⇨ 予習，質問

0–2 達成目標

- オブジェクト指向の考え方を理解する。

そのために、
実際にC++言語を使ってオブジェクト指向プログラミングに慣れる。

知能情報システムプログラムにおける学習・教育目標との対応:

対応	プログラムの到達目標
	（１）知識・理解
	・・・・・・・・・・・・・・・・・・・・・・・・・・・・
○	c) コンピュータのソフトウェアに関する基礎的知識を修得する。
	・・・・・・・・・・・・・・・・・・・・・・・・・・・・
	（２）当該分野固有の能力
	・・・・・・・・・・・・・・・・・・・・・・・・・・・・
○	c) プログラム等の要求条件を理解し、 プログラム設計等の作業スケジュールを立て、 プログラム作成等を計画通りに実行できる。
	（３）汎用的能力
	・・・・・・・・・・・・・・・・・・・・・・・・・・・・
	（４）態度・姿勢
	・・・・・・・・・・・・・・・・・・・・・・・・・・・・

0–3 教科書、参考書

教科書：

講義ノート等をpdfの形でWebに配置

⇨ 各自でdownloadを行い必要な箇所を印刷

参考書：

⇨ 講義ノート参照

C++言語に関するもの

- I.ポール「(Information&Computing 62) Cプログラマのための C++ —オブジェクト指向プログラミングに向けて」
  (1992年, サイエンス社, 絶版)
- I.Pohl「C++ for C programmers 3rd Edition」
  (1999年, Addison-Wesley,$44.99)

- 柴田望洋「Cプログラマのための C++入門」
  (1992年, ソフトバンク, 絶版;
  1999年,「新装版 Cプログラマのための C++入門」,
  ソフトバンク, 絶版)
- 塚越一雄「(Software Technology 25) 決定版 はじめての C++」
  (1999, 技術評論社, 2680円＋税)
- 柴田望洋「新版 明解C++ 入門編」
  (2009年, ソフトバンククリエイティブ, 2700円＋税)
- 柴田望洋「新版 明解C++ 中級編」
  (2014年, ソフトバンククリエイティブ, 2700円＋税)
- H. シルト「独習C++ 第4版」 (2012年, 翔泳社, 3200円＋税)
- B. ストラウストラップ「プログラミング言語C++ 第4版」
  (2015年, ソフトバンククリエイティブ, 8800円＋税)
- B. ストラウストラップ「C++のエッセンス」
  (2015年, ソフトバンククリエイティブ, 2200円＋税)
  ...ストラウストラップ(2015, 第4版) 第I部(1~5章) とほぼ同じ

. . . . . .

オブジェクト指向に関するもの

- 平澤章「オブジェクト指向でなぜつくるのか」
  (2004年,日経BP社,2400円＋税)
- 山田隆太&(株) 豆蔵「豆蔵セミナーライブオンテキスト1 わかるオブジェクト指向」
  (2005年,技術評論社,2480円＋税)
- T.A.バッド「オブジェクト指向プログラミング入門 第2版」
  (2002年,ピアソン, 4800円＋税)
- L.Cardelli&P.Wegner「On Understanding Types, Data Abstraction, and Polymorphism」
  (ACM Computing Surveys, Vol.17, No.4, pp.471–522, 1985)

. . . . . .

0–4 授業予定

	3限	4限
1回	クラス定義以前の基本事項に関して、C 言語との違いを理解する。	実習課題1: C言語との違い、新しいプログラミング環境に慣れるための課題に取り組み、LaTeXでレポートを完成させる。 (g++コンパイラ, ストリーム入出力, 等)
2回		実習課題2: 引き続き、C言語との違い、新しいプログラミング環境に慣れるための課題に取り組み、LaTeXでレポートを完成させる。 (暗黙の実引数, 関数の多重定義, 参照宣言, 空き領域演算子, string型, 等)
3回	クラスを定義してそのインスタンスを生成して使う、という考え方を理解する。	実習課題3: クラスを定義してそのインスタンスを生成して使う、という方式に慣れるための課題に取り組み、LaTeXでレポートを完成させる。

4回	何をクラスとして定義すべきか考える。 ●演算子の多重定義	実習課題4: 定義するクラスの選択／設計を特に注意深く行なってもらうための課題に取り組み、LaTeX でレポートを完成させる。
5回	●継承 (既定義クラスの拡張), ●多態性の実現, 抽象クラス	
6回	●Makefileを用いた分割コンパイル	実習課題 5: 継承, 多態性, 抽象クラスについての理解を深めるための課題に取り組み、LaTeX でレポートを完成させる。
7回	●型パラメータ付きのクラス定義, ●ジェネリックプログラミング, ●標準テンプレートライブラリ (STL), ●例外処理, ●オブジェクト指向のまとめ	
8回	ターム末試験	実習課題6: ジェネリックプログラミングや例外処理についての理解を深めるための課題に取り組み、LaTeX でレポートを完成させる。

1 C++プログラミング事始め

1–1 C++言語の歴史

1979年秋　**Bjarne Stroustrup(AT&Tベル研)**が「クラス付きのC **(C with classes)**」言語の設計作業を開始。

目的：　事象駆動型のシミュレーションを行い、マルチプロセッサとLANのためのUNIXカーネルサービスの分散を行えるようにする。

設計方針：　C言語(ハードウェアを直接的に処理可)に**Simula**(シミュレーションを記述可) スタイルのクラス機能を加え、更に型チェック等を改善。

最初の機能：　クラス、派生クラス、**public/private**によるアクセス制御、コンストラクタとデストラクタ、引数型チェックと暗黙の型変換機能をもつ関数宣言、……

1983年夏 Rick Mascittiが「C++」という名前を発案。
（追加機能：　仮想関数、関数と演算子の多重定義、参照、ストリームI/O、複素数ライブラリ、.....）

1984年 名前を「クラス付きのC (C with classes)」から「C++」に変更。

1985年10月 C++言語の最初の商用リリース。

1986年 B.Stroustrup「The C++ Programming Language」(Addison-Wesley)

1990年 M.A.Ellis&B.Stroustrup「The Annotated C++ Reference Manual」(通称ARM C++; Addison-Wesley)

—

1991年 **B.Stroustrup「The C++ Programming Language, 2nd edition」(Addison-Wesley)**

(追加機能： 総称型、例外処理の機構、.....)

1997年 **B.Stroustrup「The C++ Programming Language, 3rd edition」(Addison-Wesley)**

(追加機能： 名前空間、dynamic_cast、テンプレートに関する数多くの改良、汎用的なコンテナとアルゴリズムによるSTLフレームワーク、.....)

1998年 **ISO** による C++ の標準規格。(通称**C++98**)

2011年 ISOによる C++ の新しい標準規格。(通称**C++11**)

1–2 C++言語の特徴

- **1979年頃のC言語とSimula67言語のクラスの概念を土台にしている。**

  クラス定義の機構を用いて
  問題領域に固有の抽象データ型を実現できる。
  ⇨ 問題領域を適切にモデル化できる。
  ⇨ 分り易いプログラム

  例外処理の機構がある。
  ⇨ 本来の処理の流れの明瞭さを損なわずに、
  エラー処理をきちんと行える。
  ・・・・・・・・・

- 汎用プログラミング言語だが、(C言語と同様に)システムプログラミング用言語の性格も強い。(**Stroustrup,2015,第4版,1.2節**)

- C言語のプログラムはほぼそのままC++のプログラムとしても働く。

> C言語プログラマへの提言 (Stroustrup,2015)：
> 機能が追加されたC言語と考えない様に。
>
> C++の恩恵を受けるには、C言語の場合とは異なる設計/実装のスタイルが必要だ。

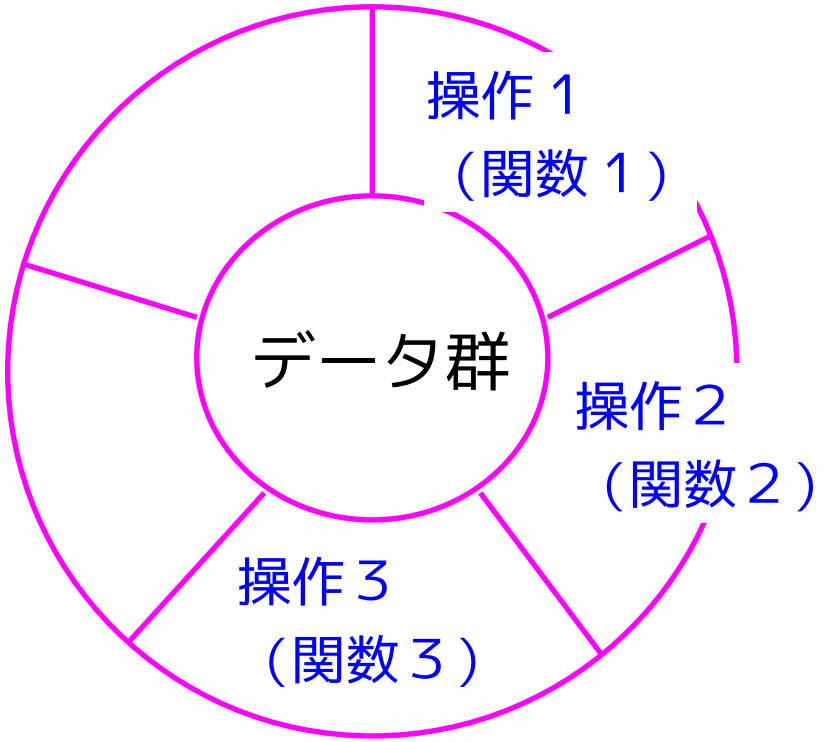
- オブジェクト指向プログラミングをサポートする。

  ⇨ 既定義クラスの拡張 (継承) により、
  コードの再利用が促進

- **STL**(Standard Template Library) が用意され、
  多くの標準的なデータ構造や関連アルゴリズムを利用できる。

  (利用の際は、パラメータ付きデータ型を用いた総称的プログラミングを行うことになる。)

1–3 オブジェクト指向の考え方、利点



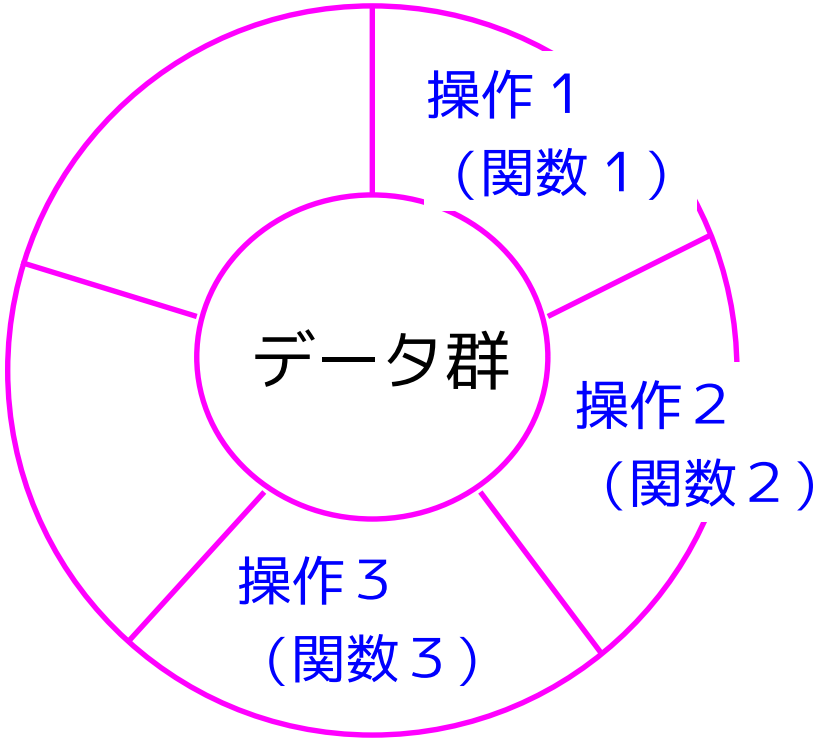
オブジェクト指向の考え方：

- 関連するデータ群と操作(関数)を
  1つにまとめてカプセル化し、
  ソフトウェア部品として扱う。
  (構造体を拡張したもので、
  オブジェクトと呼ぶ。)

- C++プログラム上では 、構造体の場合と同じ様に、まず
  ① オブジェクトの構成要素等を記述した「クラス」を定義し
  (i.e. オブジェクトの型枠／設計図を作り)、
  ② その型枠／クラス定義に沿ったオブジェクトを必要なだけ用意して
  使う、
  という作業を行うことになる。

—

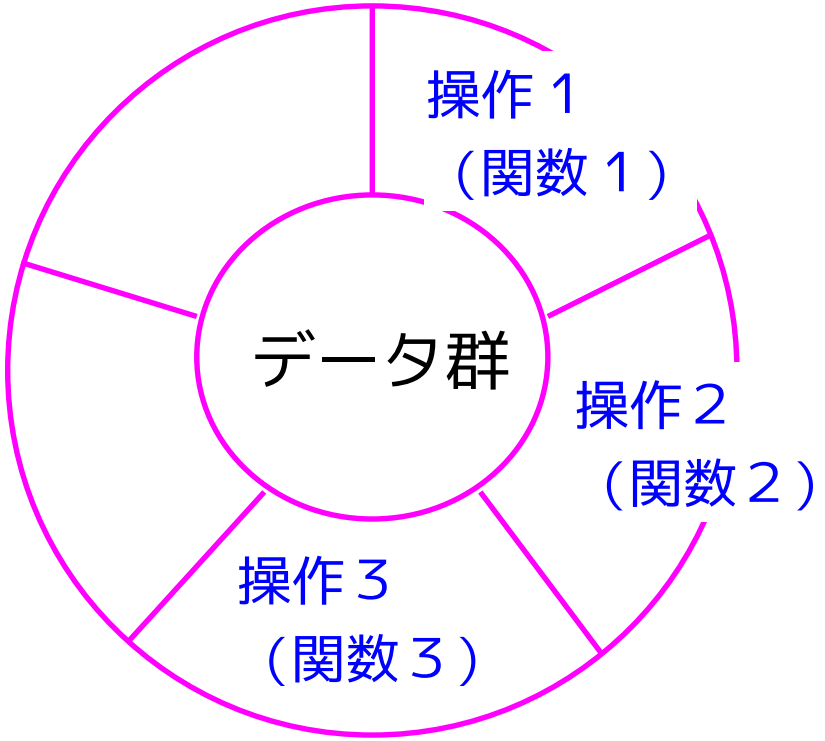
- オブジェクト内のデータについては、外部からのアクセスを制限 (i.e. データ隠蔽) する。



⇨ オブジェクト毎の独立性が高くなり、作成・保守が容易になる。

⇨ 隠蔽されている内部の実装方法 (e.g. データ構造) については、他のオブジェクトとは独立に自由に改良を進めることができる。

- プログラミング言語内に、
  既定義クラスを拡張するための機構
  を用意



⇨ 既定義クラスの拡張(継承)
を行うことにより、
コードの再利用が促進

- 既定義クラスの拡張を繰り返すと、クラス間の階層構造

⇨ プログラミング言語内に、
この階層構造を利用して
類似したオブジェクトを統合的に扱うための機構
(⟶多態性をもったコードを可能にする)
を用意する。

オブジェクト指向の難点、利点(まとめ):

- (難点) 「適切なクラスを設計する」stepも必要
  ⇨ C言語と比べてプログラミング作業は複雑に

しかし、 クラスの設計／定義を適切に行えば 、

- (利点) ソフトウェア部品の独立性／モジュール性が高くなる
  ⇨ 保守を行い易く

- (利点) コードの再利用が促進される。

- (利点) 類似したオブジェクトの統合的な扱いが可能になる。

⇨ 大きなソフトウェアを構築する時は、オブジェクト指向は特に有効。

1–4 Linuxの下でのC++プログラミング作業

拡張子：次のようなものが用いられている。

`.C` `.cc` `.cxx` `.cpp` `.c`

⇨ この授業では `.cpp` を用いる。

C++言語のコンパイラ：統合開発環境もあるが、
この授業では
コマンドライン上で動作する **GNU C++** コンパイラを用いる。

- コマンド名 ... g++
- 使い方 ... gcc コマンドとほぼ同じ

例えば 、コマンドライン上で

g++ 半角空白で区切られた C++ソースファイル や .o ファイル の列

とすると、カレントディレクトリ上に `a.out`

補足 (g++とgccの関係)：

- 昔は C++コンパイラは
  ① C++のソースをCのソースに変換し、
  ② 変換結果をCコンパイラでコンパイルする、
  という作業を行うトランスレータとして実現されていた。

- 昔は g++コマンドは
  C++のソースをコンパイルするための引数を付けて gcc を呼び出す、
  という形で実装されていたこともあったみたい。

- man gcc と打ち込んで現れる電子マニュアルによると、
  ◇ C と C++ のコンパイラは統合されています。

- ウィキペディアによると、GCCは
  昔は ... C言語のコンパイラ
  今は ... 多言語を対象にしたコンパイラ群(コレクション)

C言語標準ライブラリ関数の利用：

C言語の標準ライブラリ関数を必要に応じて使うことができる。
但し、

```
#include <c対応するヘッダファイルの拡張子を除いた名前>
```

例えば 、C言語の `scanf()` や `printf()` を用いたい場合は、
C++プログラムの前の方に

```
#include <cstdio>
```

1–5 簡単なプログラム例

C++プログラムの雰囲気に慣れるため

例1. 1 (決められた文字列の出力)　会話画面に「Welcome to C++ World!」と表示するだけのCプログラム

```
[motoki@x205a]$ cat -n printWelcomeToCppWorld.c
     1 /* 「Welcome to C++ World!」と出力するCプログラム */
     2 #include <stdio.h>
     3
     4 void printMessage(char *message);
     5
     6 int main(void)
     7 {
     8   printMessage("Welcome to C++ World!");
     9   return 0;
    10 }
```

```
11
12 /* 引数で与えられた文字列を出力 */
13 void printMessage(char *message)
14 {
15   printf("%s\n", message);
16 }
[motoki@x205a]$  gcc printWelcomeToCppWorld.c
[motoki@x205a]$  ./a.out
Welcome to C++ World!
[motoki@x205a]$
```

少しだけ手直しすれば、C++プログラムとして動作

```
[motoki@x205a]$  cat -n printWelcomeToCppWorld2.cpp
 1 /* 「Welcome to C++ World!」と出力するC++プログラム *
 2 #include <cstdio>
 3
 4 void printMessage(const char *message);
 5
```

.h なし

警告を回避

```
 6 int main(void)
 7 {
 8   printMessage("Welcome to C++ World!");
 9   return 0;
10 }
11
12 /* 引数で与えられた文字列を出力 */
13 void printMessage(const char *message)
14 {
15   printf("%s\n", message);
16 }
[motoki@x205a]$  g++ printWelcomeToCppWorld2.cpp
[motoki@x205a]$  ./a.out
Welcome to C++ World!
[motoki@x205a]$
```

例1. 2 (ストリームI/O,名前空間)　1つ前の例1.1のC++プログラムと同等の、もっとC++言語の書き方に沿ったC++プログラム

```
[motoki@x205a]$  cat -n printWelcomeToCppWorld3.cpp
     1 /* 「Welcome to C++ World!」と出力するC++プログラム */
     2 #include <iostream>  // ストリームI/Oを使うので
     3 using namespace std; // 名前を特定する際のデフォルトの
     4                      //  名前空間をstd(標準ライブラリ内
                            //              の名前を含む) と指定
     5 void printMessage(const char* message);データ型を明確に
     6
     7 int main()                                          void 不要
     8 {
     9   printMessage("Welcome to C++ World!");
                                                  return 0; 不要
    10 }
    11
    12 /* 引数で与えられた文字列を出力 */
```

```
13 void printMessage(const char* message)
14 {   ↓標準出力ストリーム
15    cout << message << endl;                    挿入演算子
16 }                              ↑改行コードを出すマニピュレータ
[motoki@x205a]$  g++ printWelcomeToCppWorld3.cpp
[motoki@x205a]$  ./a.out
Welcome to C++ World!
[motoki@x205a]$
```

例1. 3 (関数の暗黙の実引数, inline修飾子)　例1.2のC++プログラムについては、関数printMessage()の機能を少しだけ拡張できる。

```
[motoki@x205a]$  cat -n printWelcomeToCppWorld4.cpp
     1 /* 「Welcome to C++ World!」と出力するC++プログラム */
     2 #include <iostream>  // ストリームI/Oを使うので
     3 using namespace std;
     4    ↓呼出し場所へのコード埋込指示
     5 inline void printMessage(const char* message
                                        ="ProgrammingAII");
                                          デフォルト値の設定
                                             (使う前に)
     6
     7 int main()
     8 {
     9   printMessage("Welcome to C++ World!");
    10   printMessage();
    11 }
    12
```

```
13 /* 引数で与えられた文字列を出力 */
14 inline void printMessage(const char* message)
15 {
16   cout << message << endl;
17 }
[motoki@x205a]$  g++ printWelcomeToCppWorld4.cpp
[motoki@x205a]$  ./a.out
Welcome to C++ World!
ProgrammingAII
[motoki@x205a]$
```

例1. 4 (操作方法も要素に含む構造体)　C++言語では構造体の中に「データの操作方法」である関数を含ませることができる。

具体例 として、複素数の

実部データ real ,

虚部データ imag ,

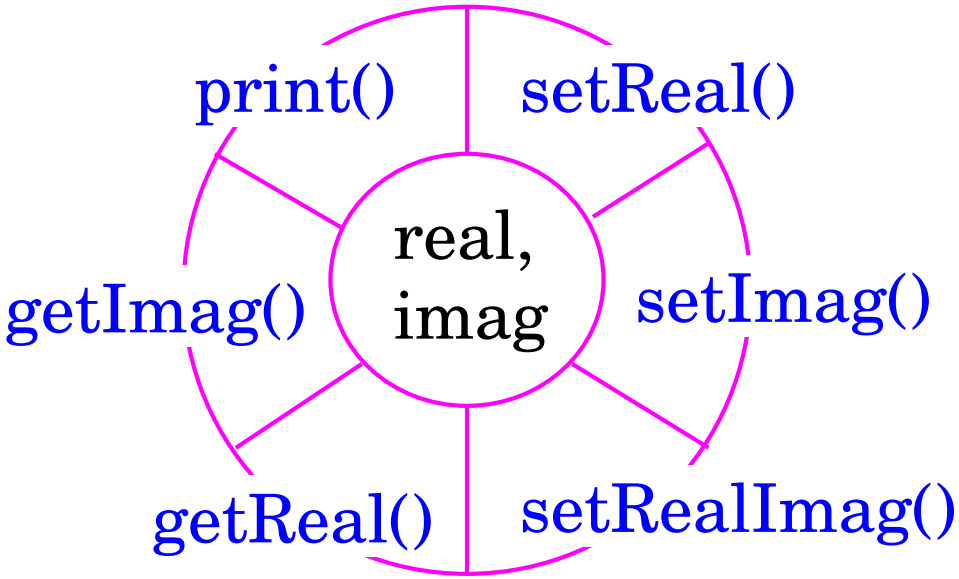
データ操作のための関数群

setReal(), setImag(),

setRealImag(),

getReal(), getImag(),

print()



を構成要素とする構造体を定義し、使用テスト

```
[motoki@x205a]$  cat -n testComplexNumberStruct1.cpp
     1 /* 操作方法も要素に含む構造体ComplexNumberを定義しテスト...
     2 #include <iostream>  // ストリームI/Oを使うので
     3 using namespace std;
     4
```

```
 5 struct ComplexNumber {
 6   double real;
 7   double imag;                          ↓自己構造体へのポインタ
 8   void setReal(double real){ this->real = real; }
 9   void setImag(double imag){ this->imag = imag; }
10   void setRealImag(double real, double imag){
11     this->real = real;
12     this->imag = imag;
13   }
14   double getReal(){ return real; }
15   double getImag(){ return imag; }
16   void print() const { 関数本体内で構造体要素の変更なし
17     cout << "(" << real << ")+("
                  << imag << ")i" << endl;
18   }
19 };
```

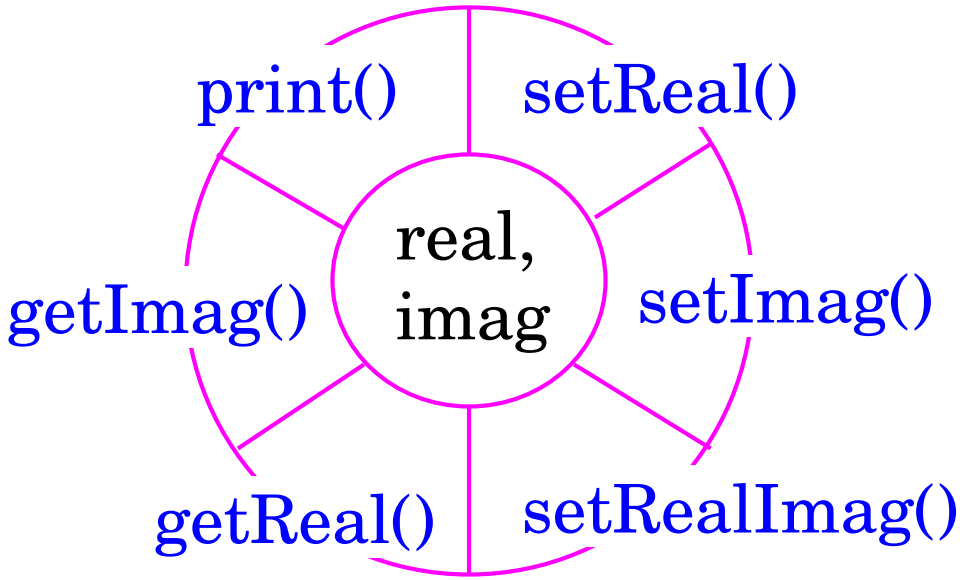
```
20
21 int main()
22 {
23   ComplexNumber z;
24
25   z.setRealImag(9.5, -3);
26   z.print();
27 }
```

struct 不要

構造体内の関数呼び出し

```
[motoki@x205a]$  g++ testComplexNumberStruct1.cpp
[motoki@x205a]$  ./a.out
(9.5)+(-3)i
[motoki@x205a]$
```

例1. 5 (アクセス指定子private, public)　例1.4のC++プログラムについては、構造体ComplexNumber内の要素への外部からのアクセスを制限しデータ隠蔽を進めることができる。

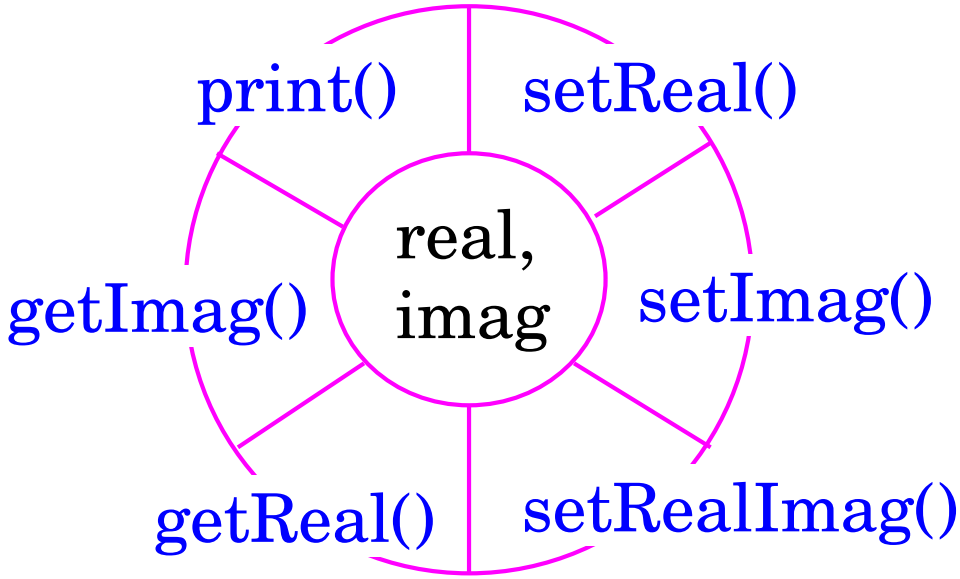



```
[motoki@x205a]$  cat -n testComplexNumberStruct2.cpp
     1 /* 操作方法も要素に含む構造体ComplexNumberを定義しテスト...
     2 #include <iostream>  // ストリームI/Oを使うので
     3 using namespace std;
     4
```

```
 5 struct ComplexNumber {
 6 private:                                  以降、外部からのアクセス禁止
 7   double real;
 8   double imag;
 9 public:                                   以降、外部からのアクセス許可
10   void setReal(double real){ this->real = real; }
11   void setImag(double imag){ this->imag = imag; }
12   void setRealImag(double real, double imag){
13     this->real = real;
14     this->imag = imag;
15   }
16   double getReal(){ return real; }
17   double getImag(){ return imag; }
18   void print() const {
19     cout << "(" << real << ")+("
                  << imag << ")i" << endl;
```

```
    20   }
    21 };
    22
    23 int main()
    24 {
    25   ComplexNumber z;
    26
    27   z.setRealImag(9.5, -3);
    28   z.print();
    29 }
[motoki@x205a]$  g++ testComplexNumberStruct2.cpp
[motoki@x205a]$  ./a.out
(9.5)+(-3)i
[motoki@x205a]$
```

例1. 6 (クラス定義)　例1.5のC++プログラム中で定義した
オブジェクトの設計図／型枠を
structの代わりにキーワードclassを用いて定義することもできる。
(キーワードclassを用いて定義される設計図／型枠のことをクラス)




```
[motoki@x205a]$  cat -n testComplexNumberClass.cpp
     1 /* クラスComplexNumberを定義しテストするC++プログラム */
     2 #include <iostream>  // ストリームI/Oを使うので
     3 using namespace std;
     4
```

```
 5 class ComplexNumber {
 6   double real;
 7   double imag;
 8 public:
 9   void setReal(double real){ this->real = real; }
10   void setImag(double imag){ this->imag = imag; }
11   void setRealImag(double real, double imag){
12     this->real = real;
13     this->imag = imag;
14   }
15   double getReal(){ return real; }
16   double getImag(){ return imag; }
17   void print() const {
18     cout << "(" << real << ")+(" << imag << ")i" << end
19   }
20 };
21
```

デフォルトでは private

```
    22 int main()
    23 {
    24   ComplexNumber z;
    25
    26   z.setRealImag(9.5, -3);
    27   z.print();
    28 }
[motoki@x205a]$  g++ testComplexNumberClass.cpp
[motoki@x205a]$  ./a.out
(9.5)+(-3)i
[motoki@x205a]$
```