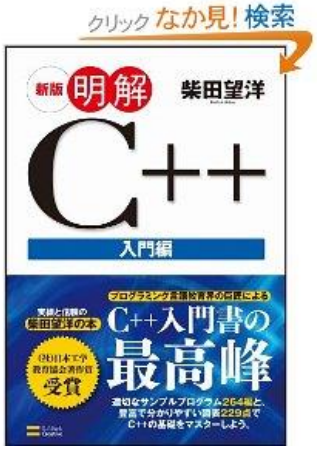
情報工学演習I
第１０回

C++の演習2（クラスの継承）

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（仮想関数）	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 (関数のオーバロード)	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

参考書（追加）



- 参考書
  - (D)「新版 明解C++　入門編」
    - 柴田 望洋 (著)

先週紹介しましたが、継承が入っていないので、お勧めできないと思いました

今日の内容

- 第7回演習課題の解説
- クラスの継承
- newとdelete
  - 配列の動的メモリ確保
  - オブジェクトのメモリ確保
- コンストラクタ、デストラクタ

第7回演習課題の解説

第7回演習課題（１）

1. 0から99までの値をとる乱数を1000個発生させ，発生した乱数の0から99までの頻度を計算し，出力するプログラムを作成せよ.頻度の計算は要素が１００個ある配列を使って行うこと．また，配列の値の変更は，ポインタ演算をして行うこと.

- 頻度：0が２回，１が２回出現

- 配列を用意して，乱数の値に応じて頻度を計算
- 配列のアクセスの方法
  - × hist[r]++;
  - ○ (*(hist + r))++; など，[]演算子を使わない

第7回演習課題（2）

2. 授業支援システムにあるtext.txt をダウンロードし，このテキストファイルから整数のデータを読みこんで動的に確保したメモリに格納し，値の平均，分散，中央値，最頻値を求めるプログラムを作成せよ．ファイルの形式は，はじめの１行目に整数の数が書かれており，次の行以降は空白区切りの整数のデータが列挙されている．

- text.txt の1行目の値を読み込む
- 1行目の値をもとにメモリをmalloc で確保し，値を読み込む
- 平均値などの計算は第5回演習課題の課題２の関数を利用

第7回演習課題（3）

3. 曜日を列挙型で表現し，switch 文を使って列挙型の曜日ごとに，自己の時間割を示すプログラムを作成せよ.

- enum 型で曜日を以下のように定義
- enum week { sunday, monday, tuesday, wednesday, thursday, friday, saturday };
- enum 型変数を分岐の式として利用

第7回演習課題（４）

4. ３次多項式　$x^3 - 168x^2 + 8621x - 129978$ を $(x-a)(x-b)(x-c)$の形に因数分解したときのa, b, c の値を計算し，出力するプログラムを作成せよ.
ただし0<a<b<c とし，a, b, c は整数となる.

- 3次多項式が0 になるx がa, b, c となる
- x=1, 2,… と代入し，3次多項式が0 になるx を小さいものからa, b, c とすればよい

クラスの継承

電卓プログラムの作成

- 以下では、電卓の振る舞いをするクラスを作って、電卓を実現
  - 加減算対応型計算機（ex6_calculator1.cc）
    - クラスCalculator1を定義
      - 足し算と引き算に対応
      - 数値のクリアができる
      - 現在の値を返すことができる
  - 四則演算対応型計算機（ex7_calculator2.cc）
    - クラスCalculator2を定義
      - クラスCalculator1の機能に加えて、かけ算と割り算に対応

クラスの継承

クラスの継承とは

- 既存のクラスの機能を引き継ぎ、一部のみを変更できる機能
  - 新しいデータメンバの追加
  - 新しいメンバ関数の追加
  - 既存のデータメンバの上書き
  - 既存のメンバ関数の上書き
- メリット
  - 過去に作ったクラスのインタフェース（関数の引数、戻り値の仕様）を変更することなく、機能を追加できる

プログラムの変更を最小限にできる

クラスの継承とは

- 用語
  - 基本クラス
    - 継承されるクラス（この例ではCalculator1）
  - 派生クラス
    - 継承するクラス（この例ではCalculator2）

計算機（加減算のみ）

ex6_calculator1.cc

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

class Calculator1 {
private:
  double val; // 計算機内部で記憶している値

public:
  Calculator1() { // コンストラクタ
    clear(); // 値のクリア
  }
  void clear() { // 値のクリア
    val = 0;
  }
  double add(double val2) { // 足す
    val += val2;
    return val;
  }
  double sub(double val2) { // 引く
    val -= val2;
    return val;
  }
  // 現在の値を返す
  double get_val() {
    return val;
  }
};
```

コンストラクタの中で関数を呼び出すこともできる

計算機（加減算のみ） 続き

ex6_calculator1.cc

```
int main() {
  string ope; // コマンド入力用
  double n; // 数値入力用
  Calculator1 calc; // 計算機のオブジェクトを作る


  cout << "Input an operation and a number separated by space (ex. ¥"+ 10¥" to add 10)." << endl
       << "Input ¥"=¥" to show the current value." << endl
       << "Input ¥"clear¥" to clear." << endl
       << "Press Ctrl+c to quit." << endl;
  cout << "Current value: "  << calc.get_val() << endl;
```

計算機（加減算のみ） 続き

ex6_calculator1.cc

```
while(1) { // 無限ループ

  cin >> ope; // キーボードからコマンドを入力する

  if (ope=="+") { // コマンドが "+" か調べる
    cin >> n; // キーボードから数字を入力する
    cout << "Result: "  << calc.add(n) << endl;
    continue; // 無限ループの最初に戻る
  }
  if (ope=="-") { // コマンドが "-" か調べる
    cin >> n; // キーボードから数字を入力する
    cout << "Result: "  << calc.sub(n) << endl;
    continue; // 無限ループの最初に戻る
  }
```

計算機（加減算のみ） 続き

ex6_calculator1.cc

```
 if (ope=="=") { // コマンドが "=" か調べる
      cout << "Current value: "  << calc.get_val() << endl;
      continue; // 無限ループの最初に戻る
    }
    if (ope=="clear") { // コマンドが "clear" か調べる
      calc.clear();
      cout << "Result: "  << calc.get_val() << endl;
      continue; // 無限ループの最初に戻る
    }
    cout << "Invalid command! ignored." << endl;
  }

  return 0;
}
```

実行例

```
$ ./a.exe
Input an operation and a number separated by space (ex. "+ 10" to add 10).
Input "=" to show the current value.
Input "clear" to clear.
Press Ctrl+c to quit.
Current value: 0
+ 10
Result: 10
+ 100
Result: 110
```

緑色の文字はユーザーが入力したものを表す

実行例（続き）

```
- -50
Result: 160
- 1000
Result: -840
clear
Result: 0
+ 10
Result: 10
=
Current value: 10
```

計算機（四則演算対応）

ex7_calculator2.cc

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

class Calculator1 {
protected: // 継承したクラスと共有するために，protectedにする
  double val; // 計算機内部で記憶している値

public:
  Calculator1() { // コンストラクタ
    clear(); // 値のクリア
  }
  void clear() { // 値のクリア
    val = 0;
  }
  double add(double val2) { // 足す
    val += val2;
    return val;
  }
  double sub(double val2) { // 引く
    val -= val2;
    return val;
  }
  // 現在の値を返す
  double get_val() {
    return val;
  }
};
```

ここだけ変更

クラスCalculation1はほぼそのまま

続き

計算機（四則演算対応）

ex7_calculator2.cc

```
class Calculator2 : public Calculator1 {
public:
  // 掛ける
  double mul(double val2) {
    val *= val2;
    return val;
  }
  // 掛ける
  double div(double val2) {
    val /= val2;
    return val;
  }
};
```

クラスCalculation1を継承した
クラスCalculation2を新たに追加

続き

計算機（四則演算対応）

ex7_calculator2.cc

```
int main() {
  string ope; // コマンド入力用
  double n; // 数値入力用
  Calculator2 calc; // 計算機のオブジェクトを作る
```

クラスCalculator1のオブジェクトから
クラスCalculator2のオブジェクトに変更

```
  cout << "Input an operation and a number separated by space (ex. ¥"+ 10¥" to add 10)." << endl
       << "Input ¥"=¥" to show the current value." << endl
       << "Input ¥"clear¥" to clear." << endl
       << "Press Ctrl+c to quit." << endl;
  cout << "Current value: "  << calc.get_val() << endl;
```

続き

計算機（四則演算対応）

ex7_calculator2.cc

```
while(1) { // 無限ループ

  cin >> ope; // キーボードからコマンドを入力する

  if (ope=="+") { // コマンドが "+" か調べる
    cin >> n; // キーボードから数字を入力する
    cout << "Result: "  << calc.add(n) << endl;
    continue; // 無限ループの最初に戻る
  }
  if (ope=="-") { // コマンドが "-" か調べる
    cin >> n; // キーボードから数字を入力する
    cout << "Result: "  << calc.sub(n) << endl;
    continue; // 無限ループの最初に戻る
  }
```

続き

計算機（四則演算対応）

ex7_calculator2.cc

```
if (ope=="*") { // コマンドが "*" か調べる
    cin >> n; // キーボードから数字を入力する
    cout << "Result: "  << calc.mul(n) << endl;
    continue; // 無限ループの最初に戻る
  }
  if (ope=="/") { // コマンドが "/" か調べる
    cin >> n; // キーボードから数字を入力する
    cout << "Result: "  << calc.div(n) << endl;
    continue; // 無限ループの最初に戻る
```

クラスCalculation2で新たに加わった
かけ算、割り算の処理を追加

続き

計算機（四則演算対応）

ex7_calculator2.cc

```
 if (ope=="=") { // コマンドが "=" か調べる
      cout << "Current value: "  << calc.get_val() << endl;
      continue; // 無限ループの最初に戻る
    }
    if (ope=="clear") { // コマンドが "clear" か調べる
      calc.clear();
      cout << "Result: "  << calc.get_val() << endl;
      continue; // 無限ループの最初に戻る
    }
    cout << "Invalid command! ignored." << endl;
  }

  return 0;
}
```

クラスの継承で気をつけること、その１

- 基本クラスのメンバが派生クラスからアクセスできること

ex6_calculator1.cc

```
class Calculator1 {
private:
  double val; // 計算機内部で記憶している値
```

派生クラスからアクセスできるようにprotectedに変更

ex7_calculator2.cc

```
class Calculator1 {
protected: // 継承したクラスと共有するために，protectedにする
  double val; // 計算機内部で記憶している値
```

クラスの継承で気をつけること、その１ 続き

- 基本クラスのメンバのアクセス制御
  - public
    - クラス外からアクセス可能
  - private
    - クラス外からアクセス可能
  - protected
    - 基本的にはprivateと同じだが、派生クラスからはアクセス可能

クラスの継承で気をつけること、その２

- 基本クラスのアクセス制御

ex7_calculator2.cc

```
class Calculator2 : public Calculator1 {
```

クラスCalculation1を継承したクラスCalculation2を新たに追加

  - この「public」の部分によって、継承の挙動が変わる
    - public: 基本クラスの公開メンバが派生クラスの公開メンバとして扱われる
    - private:基本クラスの公開メンバが派生クラスの非公開メンバとして扱われる
  - 何も指定しないときは、privateとして扱われる
  - わかりやすさのために、明示的に指定する方がいい

newとdelete
（配列の動的メモリ確保）

配列の動的メモリ確保・解放

- C言語の場合
  - mallocでメモリの確保
  - freeでメモリの解放

メモリの容量をsizeof演算子で計算しないといけないし、キャストも必要。
面倒だった

ex8_malloc.c

```
#include <stdlib.h> /* mallocを使うときのお約束 */

int main() {
  int *n;
  n = (int *)malloc(sizeof(int)*10); /* int型の変数10個分のメモリ確保 */
  free(n); /* 確保したメモリの開放 */
  return(0);
}
```

配列の動的メモリ確保・解放

- C++の場合
  - newでメモリの確保
    - 配列の大きさが１のときは、
      n=new int;
      とすれば良い
  - delete []でメモリの解放

sizeof演算子もキャストも不要

重要：配列のメモリ確保のときは、[]が必要

ex9_new_array.cc

```
int main() {
  int *n;
  n = new int[10]; // int型の変数10個分のメモリ確保
  delete [] n; // 確保したメモリの開放
  return(0);
}
```

newとdelete
（オブジェクトのメモリ確保）

ex10_new_obj.cc

```
#include <string> // stringを使うために必要
#include <iostream> // 入出力に必要
using namespace std; // お約束

class Account {
public:
  string name; // 名前
  int balance; // 残高
};

int main() {
  Account *suzukip;
  suzukip = new Account(); // Account型のオブジェクトを作成

  suzukip->name = "鈴木龍一"; // 鈴木さんの名前
  suzukip->balance = 123000; // 鈴木さんの残高
  cout << suzukip->name << "様の残高は" << suzukip->balance << "円です。" << endl;

  delete suzukip; // 確保したメモリの開放
  return 0;
}
```

クラスのメモリ確保

- newでメモリの確保
  - まず、クラスのポインタを宣言
    Account *suzukip;
  - newによるオブジェクトの作成
    suzukip = new Account();
- クラスのポインタ
  - メンバへのアクセスは”.”の代わりに、”->”を使う
    - ○ suzukip->balance
    - ○ (*suzukip).balance
    - × suzukip.balance
- deleteでメモリの解放

これはコンストラクタに渡す引数がない場合

構造体と同じ

ポインタに*を付けると、Account*型からAccount型になるので

重要：オブジェクトのときは、[]をつけない

コンストラクタ、デストラクタ

コンストラクタとデストラクタ

- クラスの特別なメンバ関数
  - コンストラクタ
    - オブジェクトが作られるときに呼ばれる関数
  - デストラクタ
    - オブジェクトが破棄されるときに呼ばれる関数

クラス内でのメモリ確保例

ex11_constructor.cc

```
#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
  }
```

確保していない配列のメモリ開放を避けるために，sizeを0にしておく

続き

クラス内でのメモリ確保例

ex11_constructor.cc

```
 dyn_mem(int _size) { // コンストラクタ
    cout << "Constructor is called." << endl;
    size = _size;
    cout << "Memory is allocated." << endl;
    array = new int[size]; // 配列のメモリ確保
  }
  ~dyn_mem() { // デストラクタ
    cout << "Destructor is called." << endl;
    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
      cout << "Memory is released." << endl;
      delete [] array;
      size = 0;
    }
  }
};
```

続き

クラス内でのメモリ確保例

ex11_constructor.cc

```
int main() {

  {
    cout << "A" << endl;
    dyn_mem dm(10); // オブジェクトの作成(大きさが10の配列のメモリを確保)
    cout << "B" << endl;
  }
  cout << "C" << endl;

  return 0;
}
```

ブロック

ブロック内で宣言された変数（オブジェクト）は、ブロックが終わると破棄される

実行例

```
$ ./a.exe
--A--
Constructor is called.
Memory is allocated.
--B--
Destructor is called.
Memory is released.
--C--
```

オブジェクトが作られて、メモリが確保された

オブジェクトが破棄されて、メモリが解放された

演習課題

第１０回演習課題（１）

- 1. 以下の仕様を満たすプログラムを作れ
  - 以下の仕様を満たすクラスを持つ
    - 第９回演習課題（１）で作成したクラスを継承する
    - 数学、理科、英語に加えて、国語の点数を保存することができる
    - 合計４教科の点数を変更（上書き）できる関数と照会できる関数がある
    - 合計４教科の点数の平均を計算して返す関数がある
  - 上記のクラスを用いて、２人分のデータ（名前、４教科の点数）を順次コマンドラインから入力できる
  - 全員分の情報を入力した後、一人ずつ名前と平均点を表示する

第１０回演習課題（2）

- 2. 以下の様に分割コンパイルを実現せよ
  - 第９回演習課題（２）で作成したプログラムを以下の3つに分割する
    - main関数を含むファイル
    - クラスのヘッダファイル
    - クラスのメンバ関数の定義を含むファイル
  - コンパイルするコマンドをテキストファイルに書く
    - 例：

      g++ -c ex5_main.cc
      g++ -c ex5_coordinate.cc
      g++ ex5_main.o ex5_coordinate.o
  - ソースファイル（３つ）とコンパイルするコマンドを提出

第１０回演習課題（３）

- 3. 以下のプログラムを作成する
  - 引数を2つ取る
    - １つはファイル名
    - もう一つは、キーワード
  - ファイルの中にキーワードが何回出現するかを数える

第１０回演習課題（４）

コンストラクタ、デストラクタを説明した場合のみ

- 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

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kadai1204_学籍番号_課題番号.cc（.cpp）
    - (Visual Studioの場合)
      ファイル名はkadai1204_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report1204_学籍番号.txt とする

提出に関して（続き）

- 提出期限
  - 12月18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