2015年

C++レクチャー 第一回

TeraPadは捨てよう

- TeraPadとJCPadはクソです
- 補完機能や折り畳み機能がある

  イケてるエディタを使おう

- そろそろIDEを使うのもいいと思うよ

- Windows, Mac, Linux対応
  Sublime Text（使ってます）, Atom, Vim, Emacs など

- Windows限定
  秀丸, EmEditor, Notepad++ など

- ↑に挙げたおすすめの中で特におすすめは
  Sublime Text と Atom です

- なお、どのエディタ（TeraPad含め）も
  設定いじったりプラグイン入れたりしてなんぼです

- 学校のPCでもzipでDLされるのとかポータブル版みたいな
  インストール不要（レジストリとか使わない）ソフトなら
  再起動しても設定リセットされずに使用可能だから、
  TeraPadとJCPadはとっとと卒業しよう

フォントも変えよう

- Windows標準ならConsolas
- Mac標準ならMenlo

- Rictyとか入れよう
  https://github.com/yascentur/Ricty とか
  ググったりもしてみてね

C++とは

- Cを＋＋した言語
- オブジェクト指向（マルチパラダイム）
- コンパイラ言語
- 強い静的型付け
- 処理が速い、多機能で複雑、難しい

変数宣言

- 関数外や関数の先頭以外でも可能
- スコープ（有効範囲）は宣言されたブロック内のみ
- C言語でもコンパイラがC99以降に対応していれば可能
- C99については↓とか見てみて

  http://www.buildinsider.net/language/clang/01

```
int main() { //引数のvoidは省略可能
    double d1 = 0.1;
    for(int i = 0; i < 10; i++) { //宣言可能
        double d2 = i; //ブロック内で宣言可能
    }
    double d3 = 0.0;
    d3 = d2 + 0.5; //エラー
    return 0;
}
```

bool型

- bool - 真と偽の二種類のみ

  1byte

  javaでいうboolean型

- C言語でもC99対応コンパイラなら使用可能

  （デフォルトでは_Bool）

  stdbool.hをインクルードすればboolでおｋ

- ちなみにCの規格は現時点では2011年のC11が最新です

```
#include <iostream> //標準入出力

int main() {
    using std::cout; /* std名前空間の関数coutを
            main関数内で名前空間省力して使用可能に */

    bool isBool = false;
    int n = 0;

    while(!isBool) {
        cout << n++ << " ";
        cout << ++n << " ";
        isBool = n>10;
    }
    return 0;
}
```

```
#include <iostream>

bool sample() {
    int x = 0;
    return x == 1;
}

int main() {
   if( sample() ) {
        std::cout << "返り値はtrue";
    } else {
        std::cout << "返り値はfalse";
    }
    return 0;
}
```

string

- 文字列のクラス
- JavaでいうStringクラス
- クラスだから文字列処理のメソッドが
  たくさんあるよ

```cpp
#include <iostream>
#include <string> //stringを使うために必要

int main() {
    using namespace std;

    string rao("raou-geh"); // rao = "〜"; でも可
    string str1(5, 'a');
    string str2(rao, 2);
    string str3(rao); // str3 = rao でも可
    string str4(rao, 4, 3);

    cout << str1 << endl
         << str2 << endl
         << str3 << endl
         << str4 << endl;

    return 0;
}
```

```
#include <iostream>
#include <string>

int main() {
    using namespace std;
    string str = "rao-";
    str += "umai"; // +=演算子で結合可能

    if(str == "rao-umai") { // ==で比較可能
        cout << "ラ王は旨い" << endl;
    }

    return 0;
}
```

namespace（名前空間）

- Javaのパッケージみたいなもの
- 変数名や関数名の衝突が起こらないようにする
- 大規模開発で有用
- std名前空間の std は standard の略

```
#include <iostream>
using namespace std; //std名前空間をグローバルにする

namespace A {
    void foo() { cout << "名前空間Aのfoo" << endl; }
    namespace C {
        void foo() { cout << "名前空間Aの中の名前空間Cのfoo" <<
endl; }
    }
}
namespace B {
    void foo() { cout << "名前空間Bのfoo" << endl; }
}

int main() {
    using A::C::foo; /* A名前空間のC名前空間のfooをmani関数内で省略
                        して使用可能にする ::はスコープ解決演算子 */
    A::foo();
    B::foo();
    foo();
    return 0;
}
```

参照渡し

- 値渡し
  変数のコピーを渡すため、関数内で引数に変更を加えても
  呼び出し元では変化なし

- 参照渡し（説明間違ってるかも）
  参照（アドレスを指すもの）を渡すため呼び出し元も変わる
  コピーを作らないため引数がサイズの大きな
  オブジェクトでも処理に負担がかからない

- 参照はポインタの一種（説明間違ってるかも）
  参照 - 参照先のオブジェクトを指し示すオブジェクト
  ポインタ - オブジェクトのアドレス

```
// &を付けると参照になる
// * 同様 & を付ける位置は型の後か変数の前どちらでもいい
void swap(int& x, int &y) {
    int tmp = x;
    x = y;
    y = tmp;
}

int main() {
    using namespace std;
    int a = 1;
    int b = 2;
    cout << "a=" << a << ", b=" << b << endl;
    swap(a, b); // &はいらない
    cout << "a=" << a << ", b=" << b << endl;
}
```

クラス

- 実際のモノを抽象化したもの
- オブジェクト指向の要
- クラス設計（デザインパターン）は非常に重要
- C++ではクラスの変数をメンバ変数、
  関数をメンバ関数と呼ぶのが一般的

アクセス指定子

- private, protected, public の三種類
- public - アクセス制限なし
- protected - 継承先まで
- private - 自クラスのみ
- Javaでいうdefalut（無指定）は無い

```
class Animal {
//無指定でも private になるけど書いたほうがいいかな
/* アクセス指定子を書いた後は、他のアクセス指定子を書くまでの
   全てのメンバにそのアクセス指定子が適用される                    */
private:
    int age;
    double weight;
    string name;
    string kind;

public:
    Animal() :
        age(0), weight(0.0), name("未決定"), kind("不明") {
        cout << "Animalのデフォルトコンストラクタ" << endl;
    }
    Animal(int a, double w, string n, string k) :
        age(a), weight(w), name(n), kind(k) {
        cout << "Animalの引数コンストラクタ" << endl;
    }
}; // ;が必要
```

メンバ関数の書き方

```
class Hoge {
    int a;
public:
    //C++では基本的にクラス内は宣言のみ
    Hoge();
    int func();
};

//クラス外に処理を実装
Hoge::Hoge : a(1) {
}
int Hoge::func() {
    //処理
}
```

Animalクラスのコンストラクタの

処理をクラス外に書こう！

```
class Animal {
public:
    Animal();
    Animal(int a, double w, string n, string k);
};

/* : を使った書き方は
    コンストラクタ初期化子（メンバイニシャライザ）という
    コンストラクタでの初期化には
    特別な理由がない限りメンバイニシャライザを使おう！  */
```

```
Animal::Animal() :
        age(0), weight(0.0), name("未決定"), kind("不明") {
        cout << "Animalのデフォルトコンストラクタ" << endl;
}
Animal::Animal(int a, double w, string n, string k) :
        age(a), weight(w), name(n), kind(k) {
    cout << "Animalの引数コンストラクタ" << endl;
}
```

const修飾子

- 主に安全性を高めるために使う
  またコンパイラが最適化しやすくなるらしい

- 変数、メンバ関数、ポインタ、オブジェクトで意味が異なる

- const変数と引数は定数（初期化のみ可能で変更不可）
  constメンバ変数は代入不可能なため、
  メンバイニシャライザで初期化する

- constメンバ関数はメンバ変数の変更が不可能になる

- **積極的に使おう！**

```
class Rao {
    const int hage; //定数
    int tall; //普通の変数
public:
    Rao() : hage(99) { //イニシャライザで初期化
//      hage = 0; 代入のためエラー
        tall = 194; //当然できる
    }
    int getHage() const { return hage; }
//  void setHage(h) { hage = h; }//定数を変更しようとしているためエラー

    int getTall() const { return tall; }
    void setTall(int t) { tall = t; } //当然OK
//  void setTall(int t) const { tall = t; } constメンバ関数でメンバ変数を変更しようとしているためエラー
};

/* オブジェクトのコピーを作らないために参照渡しをしたいが、値を変えなかったり
   変えられないようにしたい場合には const 付けて参照渡しすればおｋ          */
void func(const Rao& rao) {
    cout << rao.getHage() << endl << "ラ王の身長は" << rao.getTall() << endl;
//  rao.setTall(184); const引数のオブジェクト（のメンバ変数）を変更しようとしているためエラー
}
```

カプセル化

- メンバ変数などを外部から
  見えないように隠蔽し安全性を高める

- アクセス指定子や
  セッター、ゲッターなどで実現

- さっきのAnimalクラスに全メンバ変数のセッターとゲッターを追加しよう
- 「このレクチャー」ではセッターとゲッターとコンストラクタはクラス内に書いていいよ（インライン関数）
- 各メンバ変数を表示するメンバ関数 showData も追加しよう
- main関数で実行して動作を確かめよう

```
class Animal {
public:
    void showData() const;
    int getAge() const { return age; }
    void setAge(int a) { age = a; }
    double getWeight() const { return weight; }
    void setWeight(int w) { weight = w; }
    string getName() const { return name; }
};

void Animal::showData() const {
    cout << name << "は" << kind << "、" << age <<
"歳で" << weight << "kg " << endl;
}
```

```
int main() {
    Animal anim1(3, 6.5, "太郎丸", "犬"); /* オブジェクト生成
                オーバーロードされたコンストラクタが呼ばれる */
    anim1.showData();
//  cout << anim.name << endl; //エラー

    Animal anim2; /* オブジェクト生成
                    デフォルトコンストラクタが呼ばれる */
    anim2.showData();
    return 0;
}
```

継承

- is-a 関係
  犬（子クラス）は動物（親クラス）である

- has-a 関係
  集約（クラスが他のクラスで構成されている）

- 親クラスBを継承した子クラスAのオブジェクトの型はAであると同時にBでもある（例外あり）

- 継承の仕方に３種類あり

継承

- privateな継承

  子クラス側からは親クラス全てのメンバがprivateに見える

  子クラスは親クラスの型ではない

- protectedな継承

  子クラス側からは親クラスのprivateはそのまま

  protectedとpublicがprotectedに見える

  まず使わないから忘れておｋ

- publicな継承 - そのまま

```
class Super {
    int a;
public:
    Super() : a(0) {};
    Super(int _a) : a(_a) {};
};

class Sub : public Super {
    //親クラスのメンバを保持
    //Subの機能追加
public:
/*  親クラスのコンストラクタが先に呼ばれた後に
    子クラスのコンストラクタが呼ばれる          */
    Sub() {};
    Sub(int _a) : Super(_a) {};
};
```

- Animalクラスのメンバ変数をprotectedに変更し、Animalクラスからpublicな継承をしてDogクラスを作成しよう

- 犬種を表す文字列の定数 variety を追加しよう

- ワンワンと表示するメンバ関数 bark を追加しよう

```
class Dog : public Animal {
    const string variety;

public:
    void bark() const;
    Dog() :
        Animal(0, 0.0, "未決定", "犬"), variety("不明") {
        cout << "Dogクラスのデフォルトコンストラクタ" << endl;
    }
    Dog(int a, double w, string n, string v) :
        Animal(a, w, n, "犬"), variety(v) {
        cout << "Dogクラスの引数コンストラクタ" << endl;
    }
    string getVariety() const { return variety;}
}; // ;を忘れないこと！

void Dog::bark() const {
    cout << "ワンワン" << endl << endl;
}
```

オーバーライド

- 親クラスのメンバ関数を子クラスで上書きして特化させる
- 引数、戻り値、constの有無は一緒で処理が異なるもの
- オーバー"ロード"は引数、constの有無が
  異なるかどうかで、戻り値の型の違いは関係ない
- オーバー"ライド"されるメンバ関数には
  virtual を付けて仮想関数にする

AnimalクラスのshowDataをDogクラスで
オーバーライドして、犬種も表示するようにしよう

```
class Animal {
public:
    virtual void showData() const;
}

class Dog : public Animal {
public:
    void showData() const;
};
```

```
void Dog::showData() const {
    cout << name << "は" << kind << "で種類は" << variety << endl;
    cout << age << "歳で" << weight << "kg " << endl << endl;
}

int main() {
    Dog dog(11, 9.5, "パー", "パグ");
    dog.showState();
    dog.bark();
}
```

特に覚えておいてほしいこと

- エディタ
- 参照渡し
- コンストラクタの書き方
  （メンバイニシャライザの使い方）
- const修飾子

課題

- なんかオリジナルのプログラムを作成
- 過去に作ったものをC++に書き直すとかでももちろんいいけど、
  printfとかのCの関数、charのポインタや配列などは**極力使用しない**こと
  （それで動かなくなるならまあ妥協しても構わないけどね）
  できたら構造体とかは関数とまとめてクラスにして（当然カプセル化）
- 時間なかったり面倒くさかったら真剣にやらずに手を抜いたものでいいけど、
  オリジナリティは作って欲しい
- STLや多重継承、テンプレートなど高度っぽいことを使ってくれると喜びます
- ソースファイルの先頭に名前をコメントで記載
- **コメントはできるだけ書いてください**
- **文字コードはUTF-8にしてください**（エラーや文字化けが起こる場合には
  添付する前に文字コード変更）

提出方法

- ソースファイル（.cppや.h）のみを
  添付したメール送って
- 件名は課題とわかるように適当で
- 誰かわかるようにしてね
- 感想やダメ出し、要望などあったら
  メール本文に是非どうぞ
- **提出期限 10/13(火) 23:59**