C++言語

NO.6

目次

- オーバーライド
  - メンバ関数を書き換える
  - 上書き前のメンバ関数の機能を使って、メンバ関数を書き換える
- C言語の復習
  - 構造体とポインタ
  - アロー演算子

クラス作成の復習

演習課題6-1

- クラス名はMedicalCheckとする
- 以下のメンバー変数を持つとする。全部protected:
  - ID　　整数型
  - height　　浮動小数点型(単位m)
  - weight　　浮動小数点型(単位Kg)
- 以下のメンバー関数を持つ。全部public
  - void setID(int id);　　メンバ変数IDに値を格納
  - int getID();　　メンバ変数IDの値を取得
  - void setHeight(double h);　　メンバ変数heightに値を格納
  - double getHeight();　　メンバ変数heightの値を取得
  - void setWeight(double w);　　メンバ変数weightに値を格納
  - double getWeight();　　メンバ変数weightの値を取得

（次のページに続く）

演習課題6-1（続き）

- 以下のメンバー関数も持つ。
  - double BMI();
    - BMIは、体重(Kg) / (身長(m)^2)で求まる
  - double StandardBodyWeight();
    - BMIが22になる体重を返す。

```
1   1.61   65.2↓
25.1534
57.0262
```

実行例1

```
2 1.51 49.2↓
21.578
50.1622
```

実行例2

```
int main()
{
    int id;
    double h, w;
    cin >> id >> h >> w;
    MedicalCheck mc;
    mc.setID(id);
    mc.setHeight(h);
    mc.setWeight(w);
    cout << mc.BMI() << endl;
    cout << mc.StandardBodyWeight() << endl;
    return 0;
}
```

main関数

上書き定義（オーバーライド）

上書き定義（オーバーライド。再定義）

- 既存のメソッドの機能を拡張したいことがある
  - 高機能にする
  - 別の機能を割り当てる
  - エラーチェックを追加する

継承先で、上書き定義する

- 同じ名前の関数（引数と返り値も同じ）を定義すると上書きできる

上書き定義の例

```
class MedicalCheck {
protected:
    ･･･省略･･･
public:
    ･･･省略･･･
    double StandardBodyWeight() {
        ･･･秘密･･･
    }
};
```

継承

```
class MedicalCheck2 : public Medical Check {
public:
    double StandardBodyWeight() {
        ･･･省略･･･
    }
};
```

継承先で同じ関数を定義

```
int main()
{
    MedicalCheck   mc1;
    MedicalCheck2 mc2;
    ･･･省略･･･
    double sbw1 = mc1.StandardBodyWeight();
    double sbw2 = mc2.StandardBodyWeight();
   ･･･省略･･･
};
```

演習問題6-2(6-2)

- 演習問題6-1で定義したクラスMedicalCheckを継承したクラスMedicalCheck2を定義しなさい
- MedicalCheck2は、以下のメンバ変数を持つ
  - double rohrer();　　ローレル指数を求める
    - ローレル指数は、体重(Kg) / 身長(cm) ^{3} × 10^7で求まる（単位に注意）
  - double StandardBodyWeight();
    - ローレル指数が130になる体重を返す。
    - 上書き定義（オーバーライド。再定義）

- （次のスライドに続く）

演習問題6-2(6-2)（続き）

```
int main()
{
    int id;
    double h, w;

    cin >> id >> h >> w;
    MedicalCheck mc;
    mc.setID(id);
    mc.setHeight(h);
    mc.setWeight(w);
    cout << mc.BMI() << endl;
    cout << mc.StandardBodyWeight() << endl;

    MedicalCheck2 mc2;
    mc2.setID(id);
    mc2.setHeight(h);
    mc2.setWeight(w);
    cout << mc2.rohrer() << endl;
    cout << mc2.StandardBodyWeight() << endl;

    return 0;
}
```

```
1  1.61  65.2 ↓
25.1534
57.0262
156.232
54.2527
```

実行例1

```
2  1.51  49.2 ↓
21.578
50.1622
142.901
44.7584
```

実行例2

上書き前のメンバ関数を使う

- 上書き定義するときに、上書き前のメンバ関数を使いたいときがある
  - 例：単純に使いたいとき
  - 例：値を格納するメンバ関数を拡張して、エラーチェックした後値を格納するメンバ関数を作る
    - 値の格納は、上書き前のメンバ関数を使う
- 「クラス名::メンバ関数名」で参照可能
  - 例：「MedicalCheck::StandardBodyWeight(v)」

例：自然数しか格納できないスタックの定義

```
class stack {
    int buf[10];
protected:
    int idx;
public:
    stack() {  ・・・省略・・・ }
    int pop() {  ・・・省略・・・ }
    void push(int v) {
        ・・・省略・・・
    }
};
```

継承先で
isEmpty()作成のため、
protectedにしておく

stackクラスでpush()が
定義されている

```
class nstack : public stack {
public:
    void push(int v) {
        if (v > 0) {
            stack::push(v);
        }
    }
};
```

stackクラスのpush()を
呼んでいる

「push()」なら、nstackの
pushの呼び出しになる
（再起呼び出し）

演習問題6-3

- 前のスライドの足りない部分を補って、動くようにしなさい
- 更に、nstackに以下のメンバ関数を追加しなさい
  - int isEmpty();
    - スタックが空ならば1を返す
    - スタックに値が入っていたら0を返す

（次のスライドに続く）

演習問題6-3(つづき)

```
int main()
{
    nstack a;
    int n, v;

    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> v;
        a.push(v);
    }
    while ( ! a.isEmpty() ) {
        cout << a.pop() << endl;
    }
    return 0;
}
```

```
3 10 20 30 ↓
30
20
10
```

実行例1

```
4  11  22  -33  44↓
44
22
11
```

実行例2

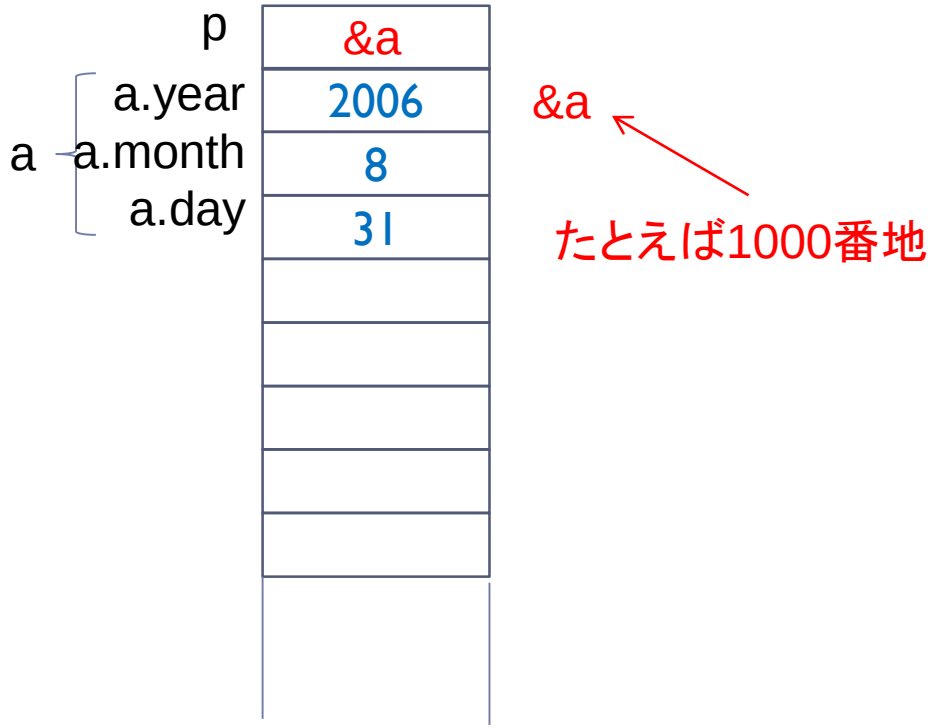
構造体とポインタ

ポインタ

- アドレス
  - 全ての変数は、格納している場所（アドレス）がある
  - 変数の前に「＆」をつけると、そのアドレスを知ることが出来る
- ポインタ変数
  - アドレスを格納することが出来る変数
- ポインタ変数を使うと、指し示すことが出来る
  - 変数の前に「*」をつけると、指し示した先を操作できる

構造体とポインタ（1）

- 「&a」でアドレスを取得できる
- ポインタ変数に「&a」（アドレス）を格納できる




```
#include <stdio.h>

struct Date {
  int year;
  int month;
  int day;
};
int main(void)
{
  struct Date *p;
  struct Date a;

  p = &a;
  return 0;
}
```

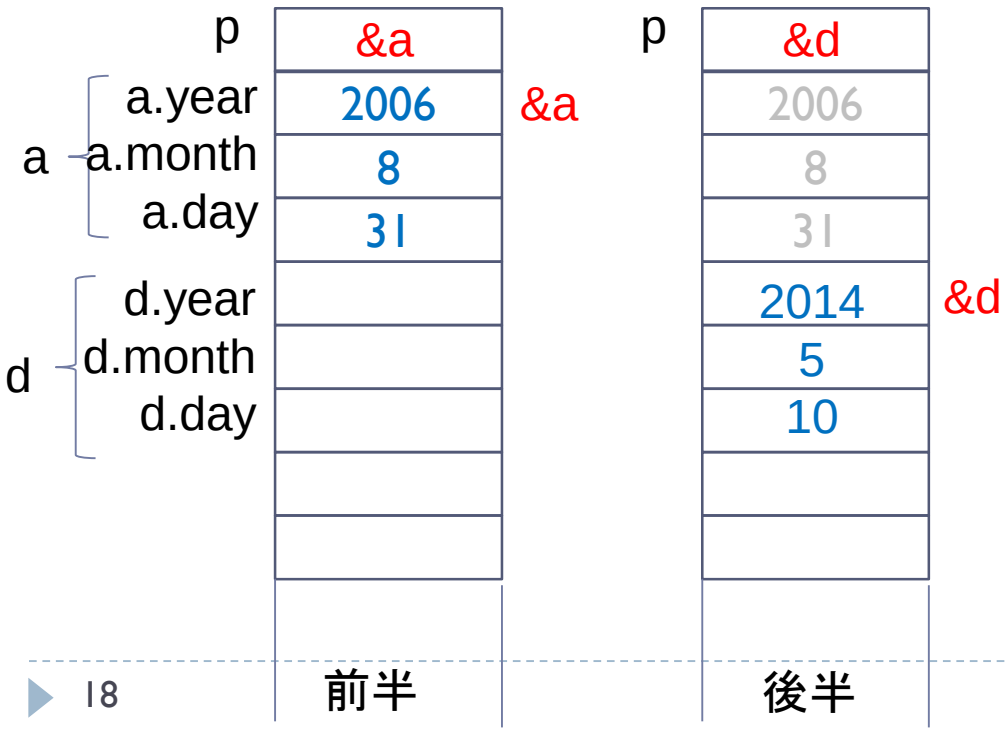
構造体の定義

構造体のポインタ定義

構造体の変数定義

構造体とポインタ

- ポインタ変数の前に「*」が付くと、その指し示した先になる
  - 「p = &a」のとき、「*p」は「a」と同じ
  - 「p = &d」のとき、「*p」は「d」と同じ




```
int main(void)
{
  struct Date *p;
  struct Date a, d;

  p = &a;
  (*p).year = 2006;
  (*p).month = 8;
  (*p).day = 31;

  p = &d;
  (*p).year = 2014;
  (*p).month = 5;
  (*p).day = 10;

  return 0;
}
```

アロー演算子

- 「(*p).year」と書くのは煩雑
  - 「p->year」と書ける
  - 「pが指し示している先のyear」と言う意味

```
int main(void)
{
  struct Date *p;
  struct Date a, d;

  p = &a;
  p->year = 2006;
  p->month = 8;
  p->day = 31;

  p = &d;
  p->year = 2014;
  p->month = 5;
  p->day = 10;

  return 0;
}
```

演習問題6-4

- 右のプログラムのコメント部分および下線部分を埋めて、正しく動くようにしなさい

（次のスライドに続く）

```
#include <iostream>
using namespace std;
struct date {
    int year;
    int month;
    int day;
};

void set1(struct date p, int year, int month, int day)
{
    p.year = year;    p.month = month;   p.day = day;
}

void set2(struct date *p, int year, int month, int day)
{
    // 構造体に値を設定するプログラムを書く
}

void set3(struct date &p, int year, int month, int day)
{
    // 構造体に値を設定するプログラムを書く
}
```

演習問題6-4（続き）

```
void print(struct date a)
{
    //  表示するプログラム
}
int main()
{
    struct date a;
    set2(___, 2012, 5, 10);
    print(a);
    set3(___, 2013, 6, 11);
    print(a);
    set1(___, 2014, 7, 12);
    print(a);
    return 0;
}
```

```
2012/5/10
2013/6/11
2013/6/11
```

実行例

ヒント：
・下線部に何を入れるか良く考えてください
（ポインタの復習、参照の復習）
（set?()の順番に注意）