静的メンバ、参照、定数化

static（静的）メンバ

- 同じクラスから生成された全オブジェクト間で共有するためのメンバ変数とメンバ関数
  - オブジェクト毎に個別に記憶する必要がない場合には，メンバ変数を静的メンバとして宣言する
  - 各オブジェクトのメンバ変数に依存しない処理は，メンバ関数を静的メンバとして宣言する
  - static をメンバ変数/関数の宣言前に付ける

static の使用例

```
class Health {
private:
    char  *name;
    float  height, weight;
    static char *heightUnit, *weightUnit; //身長と体重の使用単位を登録
public:
    Health (char *n, float h, float w); {
        name = n; height = h; weight = w;
    }
    static char* getUnit(char type) { // 身長または体重の登録単位を返す
        if (type == 'h') return heightUnit;
        else if (type == 'w') return weightUnit;
    }
    static void setUnit(char type, char* unit) {
    ...
};
```

static 使用上の注意(1)

- static なメンバ関数内で，通常のメンバ変数/関数は扱えない
  - オブジェクトを引数で渡す形式であれば使用できる

例:

✕
```
static int getBMI () {
    return weight / (height * height);
}
```

◯
```
static int getBMI (Health *data) {
    return data->weight/(data->height * data->height);
}
```

ただし，この様な場合は Health の通常のメンバ関数とする方が適切

static 使用上の注意（2）

- オブジェクトからはメンバ関数を呼び出せない

  Health data;

  × char *hUnit = data.getUnit ('h');

  – クラス名を接頭辞に付けて呼び出す

    char* hUnit = Health::getUnit ('h');

  – Health クラス内では、接頭辞を省略できる

    char* hUnit = getUnit ('h');

変数の参照渡し

参照とは？

- 変数と同じ実体を渡す際に用いる
- 変数値の複製ではなく，変数への参照先を関数に渡す
  - C言語では，引数の変数に値が複製されていた
  - C言語では，変数への参照はアドレスの値を引数の値に複製していた
  - C＋＋では型名の後に＆をつけると，参照（アドレス）が渡される
    - 引数はポインタではなく，通常の変数の様に扱われる
    - 参照で渡された場合に，元の変数の値を書き変えられる

参照の使用例

```
void swap(int& x, int& y) {// C の場合 int *x,*y
    int tmp = x; // C の場合, tmp = *x;
    x = y;          // *x = *y;
    y = tmp;        // *y = tmp;
}
int main() {
    int a = 1, b = 2;
    swap(a, b);
    printf(“a = %d, b = %d¥n”,a,b); //a = 2, b = 1
}
```

参照の使用方法

- 変数の値の複製ではないので，サイズの大きなクラス変数（構造体）を引数として渡す場合に，計算効率が良い
  - 先頭アドレスの値のみが複製される
- 引数で渡した変数の値が関数内で変更される可能性があるので，注意が必要
  - const 宣言を用いて，値の書き換えを禁止する事は可能である（後で，説明する）
- ポインタを使えない演算子のオーバーロードで多用される（後で，説明する）

参照に関する注意事項

- 参照は実体(オブジェクト)を別名化する機能なので、変数宣言のみはエラーとなる

  × int &a;

  o int &a = n; // int n; が存在する場合

- ポインタへの参照は不可

  × int& *ary = new int [10];

  × int& ary[] = new int [10]

  o int *ary = new int [10];

const 修飾子による定数化

const 修飾子とは？

- 変数が定数である事を宣言する
  - 値は初期化のみ許可され，更新は禁止される
- 変数や関数の意味を明確化する
- コンパイラによる最適化を促進する
- 型宣言の前に const キーワードを宣言する
- 関数の引数に対しても使用できる
  - 参照渡し（&）の際に，値の更新を禁止できる
- 関数の処理全体に対しても指定できる
  - 関数内のオブジェクトは全て，値を変更できなくなる

const の使用例

```
class Health {
private:
    const char  *name; // 名前は部分的に変更できない
    float  height, weight;
    const float averageBMI = 20.0; // BMI の標準値
public:
    Health (const char *n, float h, float w); {
        name = n; height = h; weight = w;
    }
    float getBMI() const; // このメソッド内では、
                          // メンバ変数の値を更新しない
};
```

文字列の const 修飾子の意味

- const char * として宣言された変数は、その変数自体には再代入が可能である。

```
const char *name = "TUT"; // 初期化
 name = "CS"; // ○ 再代入可能
```

- ただし、文字列の「編集」はできない

```
name[1] = 'O'; // × コンパイラエラーとなる
```