From structure to Class

What's "Object"

- Function is a processing unit for data
  - Function and data are deeply related to each other
- Class is a package of functions and data
  - Structure is a package of data
  - Unifying processes (member functions) and their targets (member variables) makes a program easy to be understood
- Programming is an Object-Oriented if it makes the processes for the instances of class (or objects)

Class definition (member variable)

```
struct Health {
    char  *name; /* 対象者の氏名 */
    float  height, weight; /* 身長・体重 */
    float  BMI; /* 計算した肥満度 */
};
```

```
class Health {
public: // 以下のメンバを公開
    char  *name; // メンバ変数
    float  height, weight;
    // 肥満度は省略（関数で値を返す）
};
```

where "//" denotes the beginning of a comment line

Class definition (member function)

```
class Health {
public:
    char  *name; // メンバ変数
    float  height, weight; // メンバ変数

    float getBMI () { // BMI の値を返す
        return weight / (height * height);
    }
};
```

Constructor

```
class Health {
public:
    char  *name;
    float  height, weight;
    Health () { // 引数無し
        name = null; height = weight = 0.0;
    }
    Health (char *n, float h, float w) { // 引数有り
        name = n; height = h; weight = w;
    }
    float    getBMI () {...
```

Corresponding constructor is automatically selected by the data type of parameters

Instantiation of class

Set member variables after instantiation

```
Health health; // 引数無しの構築子
health.name = “taro”;
health.height = 1.7; health.weight = 50;
printf (“BMI = %f¥n”, health.getBMI());
```

Set member variables in instantiation

```
Health health (“taro”, 1.7, 50);
printf (“BMI = %f¥n”, health.getBMI());
```

Dynamical instantiation with new operator

```
Health *health = new Health (“taro”,1.7, 50);
printf (“BMI = %f¥n”, health->getBMI());
```

Example of class (point on 2D coordinates)

```
class Point {
public: // 以下のメンバを公開
    int x, y; // メンバ変数
    Point () { x = y = 0; } // 引数無し構築子
    Point (int _x, int _y) { // 引数有り
        x = _x; y = _y:
    }
    Point* median (Point p) { // 中点を生成して返す
        return new Point ((x+p.x) / 2, (y+p.y) / 2);
    }
};
```

Instantiation of Point class

Set member variables after instantiation

```
Point p0, p1; // 引数無し構築子
p0.x = 1; p0.y = 2; p1.x = 3; p1.y = 6;
Point *cp = p0.median (p1);
printf (“median = %d, %d¥n”, cp->x, cp->y);
```

Dynamical instantiation with new operator

```
Point *p0 = new Point (1, 2);
Point *p1 = new Point (3, 6);
Point *cp = p0->median (*p1);
// 2, 4 が出力される
printf (“median = %d, %d¥n”, cp->x, cp->y);
```

Hide member variables

```
class Health {
private:
    char  *name;
    float  height, weight;

public:
    Health (char *n, float h, float w); {
        name = n; height = h; weight = w;
    }
    float getBMI () { // BMI 値を計算して返す
        return weight / (height * height);
    }
};
```

all members below this statement cannot be referred

all members below this statement can be referred

Effect of private statement

Health health (); // 引数無しの構築子を呼び出す

× health.name = “taro”;

× health.height = 1.7;

× health.weight = 50;

// Below statement is only executable

○ printf (“BMI = %f¥n”, health.getBMI());

Why hide member variables?

- Member variables represent inner states of object, and member functions are made for updating them from other classes
- Set permission for referring and updating the inner states
  - Permission for reference: make a get[VariableName]() member function
  - Permission for updating: make a set[VariableName]() member function
- Supplementary processes can be added in these functions
  - Reference: Instant calculation of non-assigned variables; getBMI()
  - Update: Check and default replacement for invalid values

Member functions for updating

```
class Health {
private:
    char  *name;
    float  height, weight;
public:
    void setHeight (float h) {
        if (h > 0. && h < 3.) // ３m 以上の人はいない
            height = h;
    }
    void setWeight (float w) {
        if (w > 0. && w < 300.) // 300kg 以上は除外
            weight = w;
    }
};
```

Member functions for reference

```
class Health {
private:
    char  *name;
    float  height, weight;
public:
    char* getName () {
        return name;
    }
    float getHeight () {
        return height;
    }
    // prohibit referencing weight values from other classes
};
```

Separation of header and source files

Example of a header file（Point.h）

```
class Point {
public:
      int x, y;
      Point ();
      Point (int _x, int _y);
      Point* median (Point p);
};
```

Member function has no description enclosed by brackets { }

Example of a source file（Point.cpp）

```
Point::Point () { x = y = 0; }
Point::Point (int _x, int _y) {
          x = _x; y = _y:
}
Point* Point::median (Point p) {
          return new Point ((x+p.x) / 2, (y+p.y) / 2);
}
```

prefix of a class name （Point:: is required for every functions

Summary

- Class := Structure + member functions
  - Functions of the same name are discriminated by their parameters
  - Header (.h) for definition, and source (.cpp) for functional description
  - Usually, only header file is disclosed for sharing information

- Constructor is made for instantiation of a class
  - The same name to the class, and is opened as public
  - Can have parameters (or omit them)
  - new operator for dynamical instantiation（like malloc in C++）

- Member function is a process for the instance of its class
  - Member variables are made for holding inner stated
  - Member variables can be hidden from other classes for safety
  - Member functions such as getter and setter are made for controlling the permission of member variables