

Pair

สมชาย ประสิทธิ์จุตระกูล
นักที นิภานันท์

Class

- class คือ การกำหนด “ประเภท” ของตัวแปรใหม่แบบหนึ่ง โดย data structure ต่าง ๆ ที่เราสร้างขึ้นนั้นก็จะเป็น class เช่นเดียวกัน
- ใน class จะประกอบด้วย “คำอธิบาย” ส่วนประกอบต่าง ๆ ของ class ซึ่งเรียกว่า member โดยมีอยู่ 2 ประเภท
 - Data member เป็นตัวกำหนด “ข้อมูล” ที่ class นั้นเก็บไว้
 - Function member เป็นตัวกำหนด “การทำงาน” ของ class นั้น

Class (ต่อ)

- ตัวอย่างเช่น เราอยากจะสร้างตัวแปรที่เก็บข้อมูล “นิสิต” ซึ่งประกอบด้วย ชื่อ, ID, และ GPA

```
class Student {  
public:  
    string name;  
    string ID;  
    float GPA;  
};
```

- หลังจากประกาศ class และ เราสามารถสร้างตัวแปรจาก class นั้นได้
 - ตัวแปรของ class เรียกว่า object
 - แต่ละ object มี member แยกกัน

```
int main() {  
    Student a;  
    Student b;  
  
    a.name = "Nattree";  
    a.ID = "4030241021"  
    a.GPA = 2.95;  
    b.name = "Gold";  
    b.GPA = 4.00;  
  
    cout << a.GPA << endl;  
    cout << b.GPA << endl;  
}
```

Class (ต่อ)

- เราสามารถกำหนดการทำงานของ class ผ่าน member function ได้

```
class Student {  
public:  
    string name;  
    string ID;  
    float GPA;  
    bool first_honor() {  
        return GPA >= 3.60;  
    }  
    int get_year() {  
        return stoi(ID.substr(0,2));  
    }  
    bool same_year(Student b) {  
        return this->get_year() ==  
            b.get_year();  
    }  
};
```

```
int main() {  
    Student a,b,c;  
  
    a.ID = "4030241021";  
    b.ID = "4030000021";  
    c.id = "5770123421";  
  
    cout << a.same_year(b)  
<< endl;  
    cout << a.same_year(c)  
<< endl;  
}
```

อยากรอเรียกใช้
member ของ
ตัวเอง ใช้ this->

#include

```
class a {  
};
```

a.h

```
#include "a.h"  
  
class b {  
};
```

b.h

```
#include "a.h"  
  
class c {  
};
```

c.h

```
#include "b.h"  
#include "c.h"  
  
int main() {  
}
```

main.cpp

```
class a {
```

```
};
```

```
class b {
```

```
};
```

```
class a {
```

```
};
```

```
class c {
```

```
};
```

```
int main() {
```

```
}
```

ผิดเพระมีการ
นิยาม a ซ้ำ

main.cpp

```
#ifndef A_H  
#define A_H  
class a {  
};  
#endif
```

a.h

```
#include "a.h"  
  
class b {  
};
```

b.h

```
#include "a.h"  
  
class c {  
};
```

c.h

```
#include "b.h"  
#include "c.h"  
  
int main() {  
}
```

main.cpp

#include

```
#ifndef A_H  
#define A_H  
class a {  
};  
#endif  
class b {  
};
```

```
#ifndef A_H  
#define A_H  
class a {  
};  
#endif  
class c {  
};
```

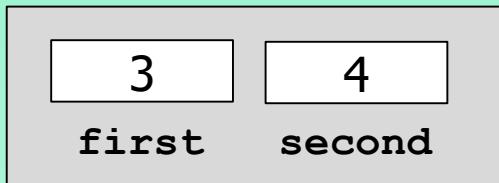
```
int main() {  
}
```

มี A_H แล้ว
จึงไม่สนใจ
บริเวณนี้

main.cpp

pair: เก็บข้อมูล 2 ก้อน

a



b



c



```
pair<int,int> a  
pair<string,int> b  
pair<float,vector<int>> c
```

pair.h

```
#ifndef _CP_PAIR_INCLUDED_
#define _CP_PAIR_INCLUDED_

namespace CP {

template <typename T1, typename T2>
class pair {
public:
    ...

protected:
    ...

private:
    ...

};

} // namespace CP

#endif
```

กำหนด
namespace
ให้ไม่ซ้ำกับ
ของ stl

overview

```
template <typename T1,typename T2>
class pair{
public:
    T1 first;
    T2 second;
    //----- constructor -----
    pair() { ... }
    pair(pair<T1,T2> other) { ... }
    pair(T1 _first,T2 _second) { ... }
    //----- operator -----
    pair<T1,T2>& operator= { ... }
    bool operator==(pair<T1,T2> &other) { ... }
    bool operator<(pair<T1,T2> &other) { ... }
};
```

Template

- เป็นการประกาศประเภทของตัวแปร โดยให้ผู้ใช้ class เป็นคนกำหนด
- ระบุโดย keyword template <typename...>

```
template <typename T1, typename T2>
class pair{
public:
    T1 first;
    T2 second;
    ...
};
```

```
pair<int,float> a;
//มี a.first เป็น int
//มี a.second เป็น float

pair<std::string,float> b;
//มี b.first เป็น std::string
//มี b.second เป็น float
```

Constructor

- มีหน้าที่กำหนดค่าเริ่มต้นของ member ของ class และเตรียมการอื่น ๆ ที่จำเป็น
 - ตัวแปรที่เป็น member
- จะถูกเรียกโดยอัตโนมัติเมื่อมีการสร้าง object ของ class นั้น ๆ
- Constructor ที่ความมีคือ
 - default constructor
 - copy constructor
- ประการศ constructor โดยใช้ชื่อของ function เป็นชื่อเดียวกับ class
- constructor มีได้หลายตัว (ด้วยการ overloading)

Copy Constructor of CP::pair

- เป็น constructor ที่สร้าง object ใหม่ให้มีค่าเหมือน object ที่รับมาเป็น parameter
- parameter จะต้องเป็น type เดียวกับ class

```
template <typename T1,typename T2>
class pair{
public:
    T1 first;  T2 second;

    // copy constructor
    pair(const pair<T1,T2>& a) {
        first = a.first;
        second = a.second;
    }
    ...
};
```

ทำการ assign ค่าให้ first กับ second ให้เหมือนกับ first, second ของ a

Default Constructor of CP::pair

- เป็น constructor ที่สร้าง object ใหม่ในกรณีปกติ
- ไม่มี parameter

```
template <typename T1,typename T2>
class pair{
public:
    T1 first;  T2 second;

    // default constructor
    pair() {
        first = T1();
        second = T2();
    }
    ...
};
```

ทำการ assign ค่าให้ first กับ second ให้ เป็นค่า default (โดย ใช้ default constructor ของ T1 และ T2)

Initializer List

- เนื่องจากการกำหนดค่าตัวแปรมีการใช้บออยใน constructor จึงมีการกำหนด syntax พิเศษขึ้นมา เพื่อให้เขียนง่ายขึ้น

```
template <typename T1, typename T2>
class pair{
public:
    T1 first;    T2 second;

    // default constructor
    pair() : first(), second() {
    }
    ...
};
```

ทำการกำหนดค่าให้ first กับ second โดยระบุค่าลงในไป วงเล็บ (ถ้าไม่ระบุ แปลว่าให้ใช้ default constructor ของตัว แปรดังกล่าว)

constructor เพิ่มเติม

- กำหนดค่าให้ first, second ตามที่ผู้ใช้กำหนด

```
template <typename T1,typename T2>
class pair{
public:
    T1   first;   T2   second;

    // custom constructor
    pair(const T1& a,const T2& b)
        : first(a), second(b) {
    }

    ...
};

};
```

ประกาศ parameter
เป็น const ...& เพื่อให้
“เร็ว” (โดยใช้ &) และ
ห้ามแก้ไข (โดยใช้
const)

ตัวอย่างการใช้งาน constructor

```
template <typename T1,typename T2>
class pair
{
public:
    T1 first; T2 second;

    pair(const pair<T1,T2>& a) {
        first = a.first;
        second = a.second;
        cout << "copy" << endl;
    }

    pair() : first(), second() {
        cout << "default" << endl;
    }

    pair(const T1 &a,const T2 &b)
        : first(a), second(b) {
        cout << "custom" << endl;
    }
}
```

```
int main() {
    //default
    CP::pair<int,int> a;
    cout << "a = " << a.first << "," <<
    a.second << endl;
    a.first = 1;
    a.second = 2;
    cout << "a = " << a.first << "," <<
    a.second << endl;

    //copy
    CP::pair<int,int> b(a);
    cout << "b = " << b.first << "," <<
    b.second << endl;

    //custom
    CP::pair<int,int> d(10,20);
    cout << "d = " << d.first << "," <<
    d.second << endl;
}
```

Function Overloading

- การเรียก constructor นั้น function ที่ถูกเรียกจะขึ้นอยู่กับ parameter ที่ส่งให้
 - method ที่เรียกตอนสร้างตัวแปรมี parameter แบบใด ก็จะเรียก constructor ที่มี parameter ตรงกัน
 - ถ้าหา method ที่มี parameter ตรงกันไม่ได้จะเกิด compilation error
 - ถ้าใน class method ที่มี parameter ตรงกันจะเกิด compilation error เช่นเดียวกัน
 - ลองทำดู
- วิธีการที่ทำให้ function ที่มีชื่อเดียวกันแต่มี parameter ต่างกันใช้งานร่วมกันได้เรียกว่า function overloading

Operator Overloading

- เราสามารถเปลี่ยนการทำงานของ operator ต่าง ๆ ของ class ได้ โดยประกาศ function สำหรับ operator ดังกล่าว
- เช่น มี object x1, x2 เป็น instance ของ class A และต้องการให้เราสามารถเขียน $X1 + X2$ ได้ ผลลัพธ์เป็น instance ของ class A เช่นกัน

```
class A
{
public:
    int value;
    A operator+(const A& x2) {
        A result;
        result.value =
            value * x2.value + 20;
        return result;
    }
};
```

```
int main() {
    A x1,x2,x3;
    x1.value = 3;
    x2.value = 6;
    x3 = x1 + x2;
    // จะได้ x3.value เป็น (3*6+20)
}
```

Operator Overloading ของ pair

- มี 3 operator ที่โดน overloaded คือ `=`, `==`, `<`
 - `operator=` คือ assignment operator เอาไว้ assign ค่า
 - `operator==` คือ equality operator เอาไว้ตรวจสอบความเท่ากัน
 - `operator<` คือ comparison operator เพื่อเอาไว้เปรียบเทียบว่า pair ใดมีค่ามาก, น้อยกว่ากัน

operator=

```
template <typename T1,typename T2>
class pair{
public:
    T1   first;   T2   second;
    ...
pair<T1,T2>& operator=(const pair<T1,T2>& other) {
    first = other.first;
    second = other.second;
    return *this;
}
...
};

};
```

operator==

```
template <typename T1,typename T2>
class pair{
public:
    T1   first;   T2   second;
    ...
    bool operator==(const pair<T1,T2> &other) {
        return (first == other.first &&
                second == other.second);
    }
    ...
};
```

Operator<

```
template <typename T1,typename T2>
class pair{
public:
    T1 first;    T2 second;
    ...
    bool operator<(const pair<T1,T2> &other) const {
        return ((first < other.first) ||
                (first == other.first &&
                 second < other.second)
            );
    }
    ...
};
```

ตัวอย่างการใช้งาน

```
int main() {
    CP::pair<int,std::string> a(10,"vishnu");
    CP::pair<int,std::string> b(a);

    // use equality operator
    cout << "a == b? " << (a == b ? "YES" : "NO") << endl;

    CP::pair<int,std::string> c(999,"asdf");
    // use assignment operator
    c = a;
    cout << "a == c? " << (a == c ? "YES" : "NO") << endl;

    c.second = "abc";
    cout << "a == c? " << (a == c ? "YES" : "NO") << endl;
}
```

ตัวอย่างการใช้งาน

```
int main() {  
    std::vector<CP::pair<int, string>> v;  
    v.push_back( CP::pair<int, string>(10, "asdf"));  
    v.push_back( CP::pair<int, string>(10, "zzz"));  
    v.push_back( CP::pair<int, string>(10, "ddd"));  
    v.push_back( CP::pair<int, string>(5, "asdf"));  
    v.push_back( CP::pair<int, string>(3, "X"));  
    v.push_back( CP::pair<int, string>(1, "asdf"));  
  
    sort(v.begin(), v.end());  
  
    for (size_t i = 0; i < v.size(); i++) {  
        cout << v[i].first << " " << v[i].second << endl;  
    }  
    return true;  
}
```

Sort เรียงข้อมูลโดย
เปรียบเทียบข้อมูลจาก
operator<

ลองแก้ไข operator< ให้ sort และเรียงตาม
second แทนได้หรือไม่?

ลองให้เรียงจาก “มากไปน้อย” แทนได้หรือไม่