Phill-CPP-0816

Polymorphism 多型化

- 繼承: 另外一個類別 屬性方法 → 有點像
- 實作: 兄弟們不同的實作, methods 名稱相同, behavior完全不同

```
#include <iostream>
using namespace std;
class Person{
  public:
    void run(){
      cout << " I can run!" << endl;
    }
};

class Salesman : public Person{
  public:
    void run(){ //overriding
      cout << " I can visit customer quickly" << endl;
    }
};

class Superman : public Person{
  public:
    void run(){
      cout << " I can run to fly very FAST!!!" << endl;
    }
};


int main()
{
    Person p1;
    Salesman p2;
    Superman p3;

    p1.run();
    p2.run();
    p3.run();
    return 0;
}
```

充分來使用 virtual

- 宣告一個 method 虛擬 → 衍生class 可以完全覆蓋跟重新定義 → base class 在宣告時 明明白白定義自己將來要被覆蓋掉
- 運用共同相似性, 可以定義相同的 pointer 來裝盛, 使用共同的 dereference 來call

```
#include <iostream>
using namespace std;
class Person{
  public:
    virtual void run(){ //設成 virtual function, 實現 polymorphism
      cout << " I can run!" << endl;
    }
};

class Salesman : public Person{
  public:
    void run() override { //overriding
      cout << " I can visit customer quickly" << endl;
    }
};

class Superman : public Person{
  public:
    void run() override {
      cout << " I can run to fly very FAST!!!" << endl;
    }
};


int main()
{
    Person* persons[3];

    persons[0] = new Person();  //new -> 動態產生
    persons[1] = new Salesman();
    persons[2] = new Superman();

    for( int i=0; i<3; i++){ //多型化應用
      persons[i] -> run();
      delete persons[i];
    }
    return 0;
}
```

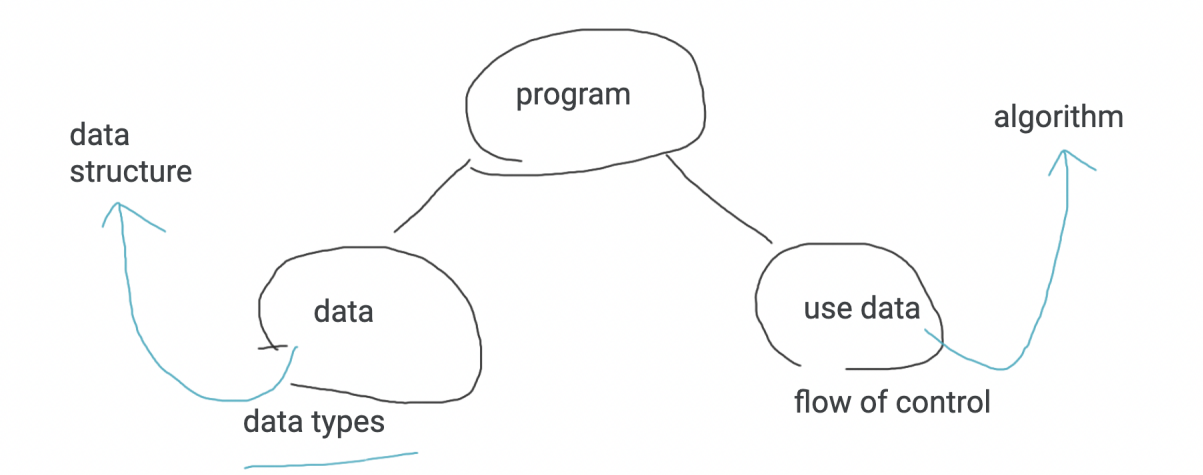
例子

Shape (draw(), move() …. OOA)

→ triangle, circle, square

→ Shape* things[]

→ for : things → draw(), things→ move() → 一個形狀 多種不同的行為 polymorphism



Computer → Efficiency

評估

→ 時間用了多少 (~response time)

→ 空間用了多少 (~memory usage)

→ 耗能

定義 問題

- 描述問題：定義問題 參數(輸入), 範圍(參數數值), 參數之間關聯性
- 問題的解：正確答案(輸出), 滿足條件(限制時間條件, 最佳化的限制, 花代價最少)
- 問題實際的例子：一到多組的參數 → 一到多組正確解; 測資

→ 演算法去解決問題

→ 評估演算法的複雜度