Chương 2. Trừu tượng hóa dữ liệu (data abstraction)

Nhắc lại các kiến thức trong C/C++

1. Hàm/ Khai báo hàm trong NNLT
2. Các cấu trúc điều khiển
3. Các toán tử
4. Các dạng dữ liệu đơn giản và phức hợp và biến dữ liệu
5. Phạm vi hoạt động của các biến
6. Kiểu dữ liệu con trỏ

Chương 2. Trừu tượng hóa dữ liệu

Trừu tượng hoá dữ liệu

1. Bản chất
2. Vai trò
3. Ví dụ
4. Sự khác nhau về trừu tượng hoá dữ liệu trong lập trình cấu trúc và lập trình hướng đối tượng
5. Bản chất của đối tượng
6. Mối quan hệ giữa các đối tượng
7. Khai báo lớp, sử dụng các đối tượng

1. Hàm/ Khai báo hàm trong NNLT

- Trong các ngôn ngữ lập trình sử dụng nguyên lý hàm hoặc khai báo nguyên mẫu của hàm (*function prototyping)*.
- Trong khai báo hàm chúng ta cần khai báo đầy đủ các thông tin sau:
  - Kiểu dữ liệu trả về từ hàm (có thể là rỗng)
  - Tên của hàm
  - Số lượng và dạng dữ liệu của các đối số tham gia vào hàm
- Ví dụ: `int translate(float x, float y, float z);`

2. Các cấu trúc điều khiển

(1) Cấu trúc if-else: có hai dạng: dạng thứ nhất: không có else và dạng thứ hai có else.

- Cấu trúc if không else:

```
if (expression)
    { statement; }
```

- Cấu trúc if có else:

```
if (expression)
    { statement;}
else
{statement;}
```

- Biểu thức phải là biểu thức logic.

C++: Giải phương trình bậc nhất

```
#include <iostream.h>
int main(){
  float a, b;
  cout << "Nhap cac he so.\n";
  cout << "He so a: "; cin >> a;
  cout << "He so b: "; cin >> b;
  if (!a)    // nhánh a==0
    if (!b)
     cout <<"PT co vo so nghiem";
    else
     cout << "PT vo nghiem";
  else // nhánh a!=0
    cout <<"PT co nghiem bang :"<< -b/a;
  return 0;
}
```


Java: Thiết lập tốc độ tàu

```
class Ship {
   ...
   public void setSpeed(double s){
    // Only change the speed if it is
    // not too high.
    if(s <= getMaximumSpeed()){
         speed = s;
    }
   }
   ...
}
```

2. Các cấu trúc điều khiển

(2) Cấu trúc switch: là cấu trúc lựa chọn cho phép lựa chọn sự 1 cách thực hiện trong nhiều cách đề xuất. Cấu trúc switch có dạng sau:

```
switch(selector) {
    case integral-value1 : statement; break;
    case integral-value2 : statement; break;
    case integral-value3 : statement; break;
    (...)
    default: statement;
}
```


2. Các cấu trúc điều khiển

- *Selector* là biểu thức thông thường phải trả về giá trị đếm được (số nguyên, ký tự, …).
- Thực hiện của cấu trúc **switch:** so sánh giá trị nhận được từ *selector* lần lượt với các giá trị đề xuất trong *integral value*.
- Từ khóa **default:**
- Từ khóa **break:**.

2. Các cấu trúc điều khiển

(3) Cấu **trúc vòng** lặp **while:** Cấu trúc này được sử dụng để mô tả một công việc nào đó sẽ lặp đi lặp lại nhiều lần.

```
while (expression)
        {statement;}
```

- Trong đó biểu thức *expression* là biểu thức lô-gic, xác định điều kiện có thực hiện công việc statement hay không.

2. Các cấu trúc điều khiển

(4) Cấu **trúc vòng** lặp **do while:** Cấu trúc này được sử dụng để mô tả một công việc nào đó sẽ lặp đi lặp lại nhiều lần.

```
do {
    statement;
  }
while(expression);
```

- Sự giống và khác nhau giữa while/do-while
- Tại sao cần có hai cấu trúc và khi nào sử dụng từng cấu trúc

In một dãy số: Java

```
// Print the numbers 1 to maximum.
public void printNumbers(int maximum){
    int nextToPrint = 1;

    while(nextToPrint <= maximum){
        System.out.print(nextToPrint+" ");
        // Get ready to print the next number.
        nextToPrint += 1;
    }
    System.out.println();
}
```


2. Các cấu trúc điều khiển

(5) Cấu trúc for: Đặc điểm của vòng lặp for là khi thựchiện có một biến đếm để đếm số lần đã thực hiện.

for (initialization; conditional; step)

{ statement; }

Quá trình thực hiện vòng lặp for được thực hiện như sau:

(1) Thựchiện khởi tạo.

(2) Kiểm tra điều kiện. Nếu điều kiện thỏa mãn thì thực hiện các lệnh ở thân của vòng lặp, nếu không thoả mãn thì thoát khỏi vòng lặp.

(3) Thực hiện các lệnh ở đếm của vòng lặp và lặp lạibước (2).

2. Các cấu trúc điều khiển

- Lưu ý: các phần khởi tạo (*initialization)*, kiểm tra điều kiện (*conditional),* và đếm (*step)* đều có thể là rỗng.

```
#include <iostream.h>
int main()
{
  for (int i = 0; i < 128; i = i + 1)
    if (i != 26)
      {cout << " value: " << i << " character: " << char(i)  <<
     endl;break;}
}
```


2. Các cấu trúc điều khiển

- **Các từ khóa break và continue**
- Trong các vòng lặp **while**, **do-while** và **for** để hỗ trợ cho khả năng lập trình mềm dẻo, có thể sử dụng các từ khóa **break** và **continue** để thay đổi trình tự thực hiện các lệnh trong thân vòng lặp.
- **break** sẽ cho phép thoát khỏi vòng lặp mà không thực hiện phần lệnh từ break cho tới khi kết thúc thân vòng lặp.
- **continue** cho phép dừng thực hiện phần còn lại của vòng lặp hiện thời và bắt đầu một chu kỳ lặp tiếp theo.

2. Các cấu trúc điều khiển

- **Đệ quy (Recursion):**
  - Recursion là một kỹ thuật hay và rất có lợi trong kỹ thuật lập trình.
  - Trong lập trình máy tính, đây là một lệnh của chương trình làm cho một mô đun hoặc chương trình con tự gọi lại chính nó.
  - Số lần thực hiện của các nội suy thường là không thể dự đoán trước.
- **Goto:** được chấp nhận trong C/C++
  - Sử dụng **goto** thông thường phá vỡ tính cấu trúc
  - Đánh giá khả năng lập trình kém của LTV.
  - Không cần sử dụng goto.

Ví dụ với Java

```
public class FactTest{

   public static int fact(int a){
       if(a <= 1) return(1);
      return(a * fact(a-1));
     }

   public static void main(String[] args){
      int x = 7;
      // println() converts int to string, adds \n
      // Can use '+' to concatente strings
        System.out.println("Fact of " + x + " is "
  +   fact(x));

      }
  }
```

3. Các toán tử (operator)

- **Khái niệm toán tử:** Chúng ta có thể coi rằng toán tử là một dạng hàm đặc biệt trong các ngôn ngữ lập trình. Toán tử có thể thao tác trên một hoặc nhiều hơn các biến dữ liệu (toán hạng) và trả về một giá trị mới.
- Thứ tự thựchiện các toán tử được quy định chặt chẽ
  - Toán tử số học
  - Toán tử so sánh
  - Toán tử lô-gich
- Sử dụng các () để thay đổi thứ tự thực hiện các toán tử.

3. Các toán tử (operator)

Các toán tử thông dụng:

- Toán tử gán (assignment) =
- Toán tử số học (arithmetic) +, -, *, /, %
- Toán tử xử lý bit (bitwise) &, |, ^, ~
- Toán tử dịch chuyển (shipt) <<, >>
- Toán tử một ngôi (unary) -, ++, --
- Các toán tử hợp kết hợp toán tử gán và các toán tử số học : +=, -=, *=, /=, %=, &=, |=, ^=
- Toán tử quan hệ <, >, <=, >=, ==, !=
- Các toán tử lô-gich: &&, !, ||
- Toán tử 3 ngôi var1=(logicexp)?exp1: exp2
- Các toán tử chuyển đổi dữ liệu (casting)

4. Các dạng dữ liệu đơn, phức hợp và biến dữ liệu

- Kiểu dữ liệu: các từ khóa mô tả phương pháp đăng ký bộ nhớ dùng để lưu trữ dữ liệu
- Kiểu dữ liệu được chia làm hai loại:
  - Các kiểu dữ liệu đã định nghĩa trong NNLT
  - Các kiểu dũ liệu do LTV xây dựng
- Các kiểu dữ liệu đơn trong C:
  - bool, char, int, float, double
  - các từ khóa xác định rõ hơn kích thước dữ liệu: short, long, signed, unsigned

4. Các dạng dữ liệu đơn, phức hợp và biến dữ liệu

- Các dạng dữ liệu phức hợp:
  - Mảng
    - Tên mảng
    - Kich thước mảng
    - Cách đánh chỉ số mảng
  - Bản ghi (cấu trúc)
    - Tên kiểu dữ liệu bản ghi
    - Tên các trường của bản ghi
    - Kiểu dữ liệu của các trường

5. Phạm vi hoạt động của các biến

- Phạm vi hoạt động (scope) của các biến cho phép xác định các nguyên lý của tạo biến, sử dụng biến và giải phóng biến
- Trong các ngôn ngữ lập trình phạm vi sử dụng các biến theo nguyên lý: trong phạm vi hàm/modul gần nhất (nearest brace)
- C++ cho phép định nghĩa các biến tại mọi điểm trong chương trình (on the fly)
- Phân loại: biến toàn cục (global), biến cục bộ, biến static
- Java: từ khóa static cho phạm vi toàn cục

Toán tử phạm vi :: (C++)

- Giúp phân biệt biến cục bộ và biến toàn cục cùng tên.

```
#include <stdio.h>
int counter = 50; // global variable
int main() {
for (register int counter = 1; // this refers to the
       counter < 10; // local variable
       counter++)
   {
       printf("%d\n", ::counter // global variable
                / // divided by
                counter); // local variable
   }
 return 0; }
```

- Trong Java không tồn tại khái niệm toán tử phạm vi

```
1  // Fig. 2.20: fig02_20.cpp
2  // Summation with for.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  // function main begins program execution
9  int main()
10 {
11    int sum = 0;                      // initialize sum
12
13    // sum even integers from 2 through 100
14    for ( int number = 2; number <= 100; number += 2 )
15       sum += number;                 // add number to sum
16
17    cout << "Sum is " << sum << endl;  // output sum
18    return 0;                         // successful termination
19
20 } // end function main
```

```
Sum is 2550
```


5. Con trỏ

- Con trỏ là một loại dữ liệu đặc biệt cho phép lưu trữ địa chỉ của các biến dữ liệu, con trỏ chỉ được phép trỏ tới biến dữ liệu thuộc kiểu nó đã khai báo
- Các phép toán liên quan tới con trỏ:

  * : lấy giá trị của biến dữ liệu nơi con trỏ trỏ tới

  &: lấy địa chỉ của biến dữ liệu
- Con trỏ và xử lý mảng
  - ++p: con trỏ p trỏ tới phần tử tiếp theo trong mảng
  - --p: con trỏ trỏ tới phần tử trước liền kề trong mảng

5. Con trỏ

- **Ví dụ 1 về sử dụng con trỏ và mảng:**

```
void main()
{ int a[10];
  int* ip = a;
  for (int i = 0; i < 10; i++)
  ip[i] = i * 10;
}
```

- **Ví dụ 2 về sử dụng con trỏ và mảng**

```
void func2 (int* a, int size)
{ for(int i = 0; i < size; i++)
    a[i] = i * i + i;
}
void main()
 { int a[5];
  func2(a, 5);
 }
```


5. Con trỏ

- **Ví dụ về sử dụng con trỏ và cấu trúc:**

```
struct strucexamp
 { char c;
    int i;
    float f;
   double d;
};
void main()
 { strucexamp s1, s2;
  strucexamp  *sp = &s1;
  sp->c = 'a';
  sp->i = 1;
  sp->f = 3.14;
  sp->d = 0.00093;

  sp = &s2; // pt khác
   sp->c = 'a';
   sp->i = 1;
   sp->f = 3.14;
   sp->d = 0.00093;
 }
```

C++: Cấp phát bộ nhớ động

- Toán tử cấp phát bộ nhớ động **new.** Có hai cách sử dụng new:
  - Cấp phát bộ nhớ cho một biến

    ```
    new type;
    ```

    ở đây new là từ khoá, còn type là kiểu dữ liệu; giá trị trả về là: một con trỏ chỉ đến vị trí tương ứng khi cấp phát thành công và NULL trong trường hợp trái lại.
  - Cấp phát một mảng động các phần tử

    ```
    new type[n]
    ```

    trong đó n là một biểu thức nguyên không âm nào đó; giá trị trả về là: Một con trỏ chỉ đến đầu vùng nhớ đủ để chứa n phần tử thuộc kiểu type

  NULL khi không còn đủ bộ nhớ để cấp phát

C++: thu hồi bộ nhớ động

- Toán tử giải phóng vùng nhớ động (heap) delete

  ```
  delete con_trỏ;
  delete [] con_trỏ;
  ```

- Trả lại vùng nhớ trỏ bởi `con_trỏ`
- Sau lệnh delete giá trị của `con_trỏ` không xác định

C++: new và delete

Ví dụ:

Với khai báo `int *adr;`

lệnh

`adr=new int;` **sẽ cấp phát một vùng nhớ cần thiết cho một giá trị kiểu int và gán địa chỉ cho adr.**

lệnh:

`delete adr;` **sẽ giải phóng vùng nhớ trỏ bởi adr;**

Lệnh

```
char *adc= new char[100];
```

sẽ cấp phát một vùng nhớ đủ để chứa 100 ký tự và đặt địa chỉ đầu vùng nhớ vào biến adc.

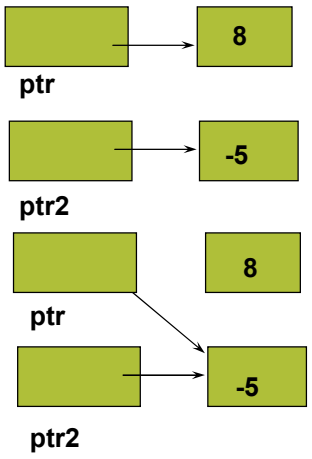
Lệnh `delete adc;` **sẽ giải phóng vùng nhớ này.**

Rò rỉ bộ nhớ

- Vấn đề: *mất* mọi con trỏ đến một vùng bộ nhớ được cấp phát. Khi đó, vùng bộ nhớ đó bịmất dấu, không thể trả lại cho heap được.

```
int* ptr = new int;
*ptr = 8;

int* ptr2 = new int;
*ptr2 = -5;
ptr = ptr2;
```

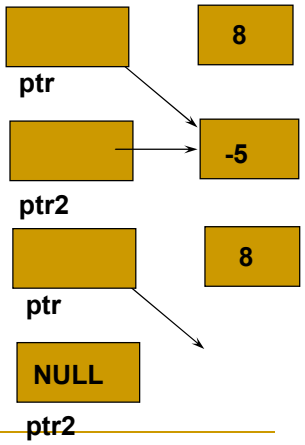


Con trỏ lạc

- Khi **delete ptr2**, ta cần chú ý không xoá vùng bộ nhớ mà một con trỏ ptr khác đang trỏ tới.

```
int* ptr = new int;
*ptr = 8;
int* ptr2 = new int;
*ptr2 = -5;
ptr = ptr2;

delete ptr2;    // ptr is left dangling
ptr2 = NULL;
```



Java

- Trong Java không có khái niệm con trỏ
- Sau này chúng ta sẽ thấy việc tác động lên bất cứ một biến (đối tượng) nào đều thông qua tham chiếu

Trừu tượng hoá dữ liệu là gì ?

Grady Booch định nghĩa về trừu tượng hóa:

- Sự trừu tượng hoá biểu thị những đặc tả thiết yếu của đối tượng để phân biệt nó với các đối tượng khác.
- Sự trừu tượng hoá thể hiện khả năng phân biệt ranh giới rõ ràng giữa các đối tượng và những tính chất đặc thù của chúng.
- Sự trừu tượng hóa phụ thuộc vào sự quan sát của của từng người.
- Sự trừu tượng hoá tốt phải nhấn mạnh được các chi tiết quan trọng và bỏ qua những tiểu tiết không quan trọng
- Mọi sự trừu tượng hoá đều có các đặc tính tĩnh và các đặc tính động.

2. Vai trò của trừu tượng hoá dữ liệu

- Sự trừu tượng hoá dữ liệu là một kỹ thuật mạnh mẽ giúp chúng ta giảm bớt sự phức tạp của bài toán.
- Khi không thể nắm bắt toàn bộ đối tượng phức tạp, chúng ta phải lựa chọn bỏ qua những chi tiết không cần thiết, thay vào đó bằng một mô hình đối tượng tổng quát lý tưởng.
- Xuất phát từ kỹ thuật trừu tượng hoá, người ta đã đi đến những kỹ thuật khác như môđun hoá phần mềm, hay sự phân cấp của hệ thống phần mềm (quan điểm chia để trị). Nhờ đó đã làm cho bài toán trở nên rõ ràng, dễ lưu trữ cũng như thực hiện.

3. Ví dụ

- Mô tả hoạt động của một ngăn xếp trong cấu trúc dữ liệu bằng mảng
- Các dữ liệu cần thiết cho ngăn xếp:
  - Kích thước của phần tử dữ liệu
  - Số lượng các dữ liệu
  - Chỉ số phần tử tiếp theo
- Các hoạt động với các ngăn xếp
  - Khởi tạo
  - Thêm một phần tử vào ngăn xếp
  - Đếm số phần tử của ngăn xếp
  - Xóa phần tử trong ngăn xếp

4. Trừu tượng hoá dữ liệu trong lập trình cấu trúc

- Khai báo cấu trúc thích hợp:

```
typedef struct CStashTag
{
    int         m_nSize;
    int         m_nQuantity;
    int         m_nNext;
} CStash;
```

Khai báo các thao tác thích hợp:

```
void initialize(CStash* s, int size);
void cleanup(CStash* s);
int add(CStash* s, const void* element);
void* fetch(CStash* s,int index);
int count(CStash* s);
void inflate(CStash* s, int increase);
```

4. Trừu tượng hoá trong lập trình hướng đối tượng

```
struct CStash {
//dữ liệu
    int        m_nSize;
    int
       m_nQuantity;
    int        m_nNext;
// Hàm
void initialize(int size);
void cleanup();
 int add(const void*
  element);
void *fetch(int index);
    int count();
void inflate(int
  increase);
};
```


4. Phân biệt trừu tượng hóa của hai cách tiếp cận

- Lập trình cấu trúc:
  - Dữ liệu riêng biệt
  - Hàm tác động lên cấu trúc dữ liệu
  - Chú ý tới đặc điểm khai báo hàm
- Lập trình hướng đối tượng
  - Dữ liệu và các hàm tác động cùng năm trong một cấu trúc lớp
  - Các hàm tác động lên dữ liệu của đối tượng của mình
  - Khai báo các hàm: đối tượng là ẩn (mặc định)
- Sự tiến hóa của trừu tượng hóa:

5. Bản chất của đối tượng

- Đối tượng là gì ?
- Đối tượng là sự biểu diễn của một thực thể, hoặc trong thế giới thực như bàn, ghế, con người, hoặc là những thực thể trừu tượng
- Đối tượng là là sự trừu tượng hoá có ranh giới rõ ràng và có ý nghĩa đối với ứng dụng.
- Từng đối tượng trong hệ thống bao giờ cũng có ba đặc tả:
  - Trạng thái
  - Hoạt động
  - Đặc điểm nhận dạng

5. Bản chất của đối tượng

- **Trạng thái của đối tượng**
  - Trạng thái của một đối tượng là một trong số những hoàn cảnh mà đối tượng có thể tồn tại. Thông thường, trạng thái của đối tượng thay đổi theo thời gian
  - Trạng thái của đối tượng được định nghĩa là tập tất cả các đặc tính, các giá trị của các đặc tính đó, cộng với mối quan hệ của đối tượng với các đối tượng khác.

5. Bản chất của đối tượng

- **Hoạt động của đối tượng**
  - Hoạt động của đối tượng xác định cách thức đối tượng đáp ứng các yêu cầu từ các đối tượng khác và đó là tất cả những gì đối tượng có thể làm. Hoạt động của đối tượng được thực hiện bởi một tập các thao tác cho đối tượng.
- **Đặc điểm nhận dạng**
  - Đặc điểm nhận dạng là một đặc tính của đối tượng cho phép phân biệt nó với các đối tượng khác.

Mối quan hệ giữa các đối tượng

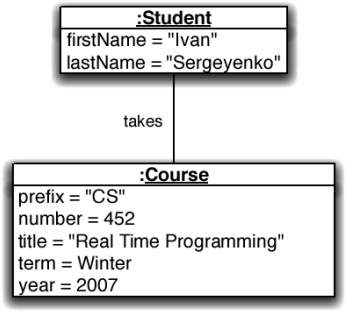
- **Mối quan hệ giữa các đối tượng**

  Toàn bộ hệ thống được xây dựng từ rất nhiều lớp và đối tượng. Hoạt động của hệ thống thu được thông qua sự phối hợp của các đối tượng trong hệ thống. Các mối quan hệ cung cấp các đường dẫn để các đối tượng tương tác với nhau. Có hai loại quan hệ giữa các đối tượng là :

  - *liên kết(link)*
  - *kết tập (aggregation)*

Mối quan hệ liên kết (link)

- **Mối** quan hệ liên kết là sự kết nối vật lý hoặc logic giữa các đối tượng. Một đối tượng phối hợp với các đối tượng khác thông qua các liên kết của nó với các đối tượng này. Nói một cách khác, một liên kết biểu diễn một liên hợp (association) xác định, trong đó một đối tượng(client) sử dụng những dịch vụ của đối tượng khác(supplier).



Mối quan hệ liên kết (link)

- Thông thường, thông điệp được truyền giữa hai đối tượng là một chiều, đôi khi có thể là cả hai chiều. Các thông điệp được khởi tạo ở phía client, sau đó được đưa tới supplier, còn dữ liệu có thể dịch chuyển theo cả hai chiều trên liên kết.

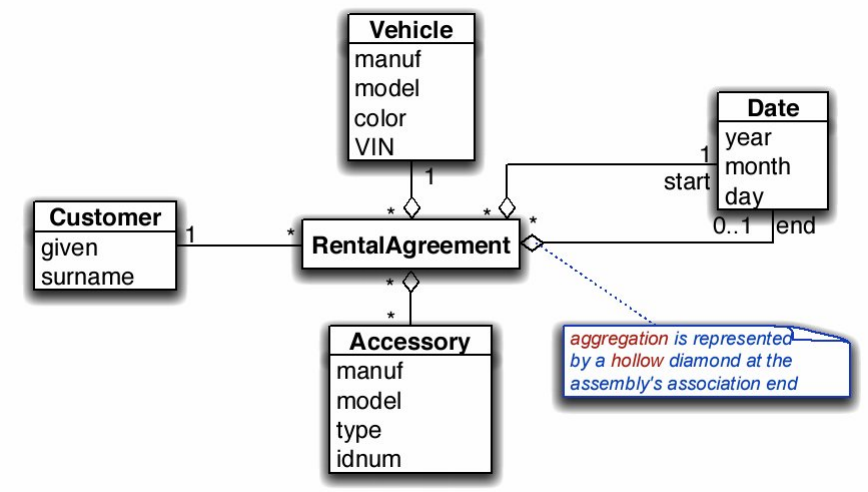
6. Mối quan hệ giữa các đối tượng

- Với mỗi liên kết, một đối tượng có thể có một trong ba vai trò :
  - *Actor*: Một đối tượng có thể hoạt động trên các đối tượng khác chứ không bị thao tác bởi các đối tượng khác.
  - *Server*: Một đối tượng không bao giờ hoạt động trên các đối tượng khác; nó chỉ có thể bị thao tác bởi các đối tượng khác.
  - *Agent*: Là đối tượng vừa có thể hoạt động trên các đối tượng khác, lại vừa có thể bị các đối tượng khác thao tác.

6. Mối quan hệ giữa các đối tượng

- **Mối quan hệ kết tập (aggregation)**
  - Mối quan hệ kết tập chỉ là một dạng đặc biệt của mối quan hệ liên hợp trong đó một đối tượng là sự tổng hợp của các đối tượng thành phần.
  - Ví dụ một chiếc xe ô tô có 4 bánh, một cần lái, một hộp số, một động cơ ...

Ví dụ về kết tập: Hợp đồng thuê xe ô tô

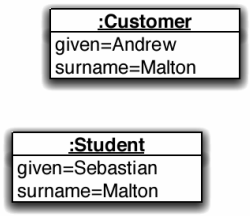


7. Khai báo, sử dụng lớp

- Khai báo lớp
- Khai báo đối tượng thuộc về lớp
  - Khai báo các biến tĩnh
  - Khai báo các biến động sử dụng con trỏ
- Các toán tử new, delete
- Đặc điểm lưu ý khi sử dụng các toán tử này

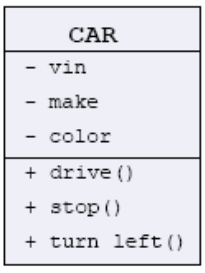
Khai báo lớp

- Trước khi tạo ra một lớp, cần phải:
  - Xác định các thuộc tính
  - Xác định các phương thức
- Thuộc tính thường là các biến có kiểu dữ liệu cơ sở, cũng có thể là các đối tượng
- Những ứng cử viên là thuộc tính:
  - định danh đối tượng: tên, ID
  - đặc tính vật lý: hình dạng, màu sắc
  - đặc tính thời gian: ngày, giờ
  - đặc tính số
  - miêu tả chung



Khai báo các phương thức

- Khi đã có mọi thành viên dữ liệu, ta chuyển sang các hành vi
- Hành vi của đối tượng được cài đặt trong C++ và Java bằng các hàm gọi là phương thức. Một phương thức (method) hay hàm thành phần là một hàm được định nghĩa bên trong một lớp
- Các đối tượng sinh ra từ một lớp có thể thực thi mọi phương thức mà lớp đó định nghĩa.



C++: Lớp Point

```
class point {
        double x,y;              // private data members
    public:
        point (int x0, int y0);          //  public methods
        point () { x = 0; y = 0;};       //  a constructor
         void move (int dx, int dy);
         void rotate (double alpha);
         int distance (point p);
};
```


Lớp hình chữ nhật

```
class T_hcn {
                int x,y;
                single w,h;
public :
   void set_value(int,int,single,single);
   single area(void)
   {
      return (w*h);
   };
} ;
void T_hcn::set_value(int a,in b,single
   c,single d)
{x=a; y=b; w=c; h=d;};
```

```
int main() {
T_hcn hcn;
hcn.set_value(0,0,3
,4);
 cout << '' dien tich
= '' << hcn. area();
 }
```

Cài đặt phương thức (C++)

- Giao diện của phương thức luôn đặt trong định nghĩa lớp, cũng như các khai báo thành viên dữ liệu.
- Phần cài đặt (định nghĩa phương thức) có thể đặt trong định nghĩa lớp hoặc đặt ở ngoài.
- Trong:
  - Khai báo inline: phương thức được định nghĩa bên trong khai báo lớp
- Ngoài:
  - Khai báo thông thường: phương thức được định nghĩa ngoài thông báo lớp, và dùng toán tử phạm vi ::

Phương thức C++

```
class Car {
...
void drive(int speed,
int distance)
};
...
void Car::drive (int speed,
int distance)
{...}
```

```
class Car {
...
void drive(int speed,
int distance) {
  // định nghĩa tại đây
}
};
```

C++: Lớp Account

```
class Account {
public:
  Account();
  float account_balance() const;       // Return the balance
  float withdraw( const float );       // Withdraw from account
  void deposit( const float );         // Deposit into account
  void set_min_balance( const float ); // Set minimum balance
private:
  float the_balance;                   // The outstanding balance
  float the_min_balance;               // The minimum balance
};
```


C++: Lớp Account

```
class Account {
public:
  Account();
  float account_balance() const;       // Return the balance
  float withdraw( const float );       // Withdraw from account
  void deposit( const float );         // Deposit into account
  void set_min_balance( const float ); // Set minimum balance
private:
  float the_balance;                   // The outstanding balance
  float the_min_balance;               // The minimum balance
};
```

C++: Lớp Account

```
Account::Account()
{
  the_balance = the_min_balance = 0.00;
}

float Account::account_balance() const
{
  return the_balance;
}

float Account::withdraw( const float money )
{
  if ( the_balance-money >=
     the_min_balance )
  {
    the_balance = the_balance - money;
    return money;
  } else {    return 0.00;  }
}
```

```
void Account::deposit( const float
money )
{
  the_balance = the_balance +
money;
}

void Account::set_min_balance(
const float money )
{
  the_min_balance = money;
}
```


C++, tách đặc tả lớp và cài đặt phương thức

- Nói chung, ta nên tách phần khai báo phương thức ra khỏi phần cài đặt (định nghĩa),
- Việc phân tách này cho hai ích lợi quan trọng trong đóng gói:
  - Do tách định nghĩa khỏi phần khai báo lớp, người dùng không cần quan tâm đến chi tiết (một khai báo lớp sẽ dài và khó đọc như thế nào nếu nó kèm theo khoảng 10 đến 20 phương thức, mỗi phương thức dài hàng trăm dòng lệnh?)
  - Tách giao diện phương thức ra khỏi cài đặt cho phép ta thay đổi chi tiết cài đặt mà không ảnh hưởng đến người dùng.

Lớp Rectangle (C++)

```
#include <iostream>
using namespace std;
class Rectangle
{
   private:
      float width, length;
   public:
      void setWidth(float);
      void setLength(float);
      float getWidth(),
         getLength(),
         getArea();
};
```

```
void Rectangle::setWidth(float w)
{width = w;}
void Rectangle::setLength(float
   len)
{length = len;}
float Rectangle::getWidth()
{return width;}
float Rectangle::getLength()
{return length;}
float Rectangle::getArea()
{return width * length;}
```

Đặc tả Lớp, với các nguyên mẫu hàm thành phần + dữ liệu

Định nghĩa hàm thành phần bên ngoài lớp

Đặt khai báo lớp riêng rẽ (C++)

- Để đảm bảo tính đóng gói, ta thường đặt khai báo của lớp trong file header
  - tên file thường trùng với tên lớp. Ví dụ khai báo lớp Car đặt trong file "car.h"
- Phần cài đặt (định nghĩa) đặt trong một file nguồn tương ứng
  - "car.cpp" hoặc "car.cc"
- Quy ước đặt khai báo/định nghĩa của lớp trong file trùng tên lớp được chấp nhận rộng rãi trong C++
  - là quy tắc bắt buộc đối với các lớp của Java

File header Car.h

```
// car.h
#ifndef CAR_H
#define CAR_H
class Car {
    public:
        //...
        void drive(int speed, int distance);
        //...
        void stop();
        //...
        void turnLeft();
    private:
        int vin; //...
        string make; //...
        string model; //...
        string color; //...
};
#endif
```


Khai báo lớp trong Java

- Tất cả các phương thức đều được định nghĩa và cài đặt bên trong lớp

- public class Account
  {
    protected double balance;

    // Constructor to initialize balance
    public Account(double amount)
  {
        balance = amount;
  }
 ……

tiếp lớp Account (Java)

```
// Overloaded constructor for empty balance
    public Account()
    {
        balance = 0.0;
    }

    public void deposit( double amount )
    {
        balance += amount;
    }
```


Tiếp lớp Account (Java)

```
public double withdraw( double amount )
    {
            // See if amount can be withdrawn
            if (balance >= amount)
            {
                    balance -= amount;
                return amount;
            }
            else
            // Withdrawal not allowed
                    return 0.0;
    }

      public double getbalance()
    {
            return balance;
    }
}
```

Sử dụng các đối tượng của lớp Account (Java)

```
class AccountDemo
{
      public static void main(String args[])
    {
            // Create an empty account
            Account my_account = new Account();

            // Deposit money
             my_account.deposit(250.00);

            // Print current balance
             System.out.println ("Current balance " +
                        my_account.getbalance());

            // Withdraw money
             my_account.withdraw(80.00);

            // Print remaining balance
             System.out.println ("Remaining balance " +
                        my_account.getbalance());
    }
}
```


Đối tượng trong C++ và Java

- C++: đối tượng của một lớp được tạo ra tại dòng lệnh khai báo:
  - Point p1;
- Java: Câu lệnh khai báo một đối tượng thực chất chỉ tạo ra một tham chiếu, sẽ trỏ đến đối tượng thực sự khi gặp toán tử new
  - Box x;
  - x = new Box();
  - Các đối tượng được cấp phát động trong bộ nhớ heap

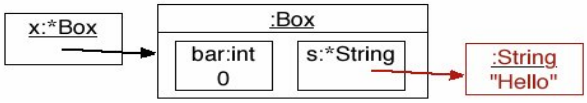
Đối tượng trong Java

```
class Box
{
    int bar;
    String s;
}
```

```
Box x;
```


```
x = new Box ();
```


```
x.s = "Hello";
```

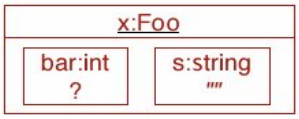


Đối tượng trong C++

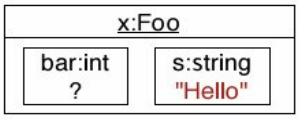
```
class Foo
{
        int bar;
        string s;
};
```

gotta have that semi

```
Foo x;
```




```
x.s = "Hello";
```



Phạm vi và phương thức

- Các thành viên dữ liệu của một lớp có phạm vi lớp
  - Có thể coi phạm vi này ở trên phạm vi phương thức "một bậc"
  - Tất cả các thể hiện của một lớp đều nằm trong phạm vi lớp của lớp đó
- Do vậy, phương thức của một lớp có thể truy nhập thành viên bất kỳ của lớp đó

```
class Foo {
   public:
      void bar();

   private:
      int x;
};
```

```
void Foo::bar()
{
   x = 5;
   ...
}
```

Foo::x

Ví dụ, câu hỏi, bài tập

Các câu hỏi:

1. Các khái niệm: biến, hàm, toán tử trong các NNLT
2. Các đặc điểm của hàm và khai báo hàm trong C/C++
3. Khái niệm cấp phát bộ nhớ động
4. Khái niệm trừu tượng hóa dữ liệu
5. Khai báo lớp trong C++
6. Phân biệt sự giống/khác nhau giữa cấu trúc và lớp
7. Các thành phần của đối tượng
8. Các cách sử dụng các thuộc tính của đối tượng

Bài tập tuần 2:

Xây dựng các hàm thực hiện các giải thuật sắp xếp/tìm kiếm trên mảng theo các giải thuật:

- Sắp xếp đơn giản: Sắp xếp lựa chọn (Selection Sort), Sắp xếp thêm dần (Insertion Sort), Sắp xếp kiểu đổi chỗ (Exchange Sort), Sắp xếp phân đoạn (Partition Sort), Sắp xếp nhanh (Quick Sort)
- Tìm kiếm tuần tự, Tìm kiếm nhị phân

(Tham khảo các giải thuật trong Sách Đỗ Xuân Lôi - Cấu trúc dữ liệu và giải thuật - NXB KH &KT. Trang 239-255; Trang 269-273)