VIỆN ĐIỆN TỬ - VIỄN THÔNG

School of Electronics and telecommunications

KỸ THUẬT LẬP TRÌNH C/C++

LỚP VÀ ĐỐI TƯỢNG

Giảng viên: TS. Nguyễn Thị Kim Thoa

Học kỳ: 20211

Năm học: 2021-2022

Nội dung

1. Một số khái niệm
2. Định nghĩa lớp
3. Biến thành viên
4. Hàm thành viên
5. Kiểm soát truy nhập
6. Hàm tạo và hàm hủy
7. Hàm tạo bản sao
8. Hàm toán tử gán
9. Thành viên tĩnh
10. Nạp chồng toán tử
11. Khai báo friend

1. Một số khái niệm

- **Đối tượng là gì?**
  - Mô hình đại diện của một đối tượng vật lý:
    - ✓ Person, student, employee, employer
    - ✓ Car, bus, vehicle,…
  - Đối tượng logic:
    - ✓ Trend, report, button, window,…
- **Một đối tượng có:**
  - Các thuộc tính
  - Trạng thái
  - Hành vi
  - Căn cước
  - Ngữ nghĩa

1. Một số khái niệm[...]

- **Lớp là gì?**
  - Là sự thực thi của các đối tượng có chung các thuộc tính, hành vi, quan hệ, ngữ nghĩa.
  - Lớp là một kiểu dữ liệu mới có cấu trúc, trong đó việc truy nhập các biến thành viên được kiểm soát thông qua các hàm thành viên.
  - Các dữ liệu của lớp $\Rightarrow$ biến thành viên
  - Các hàm của lớp $\Rightarrow$ hàm thành viên
  - Một biến của một lớp $\Rightarrow$ một đối tượng

2. Định nghĩa lớp

Xét kiểu dữ liệu có cấu trúc

```
sruct Date{
        int day, month, year;
};
void set_date(Date& date,int d, int m,
int y){
        date.day = d;
        date.month = m;
        date.year = y;
}
void add_day(Date& date, int n){
        date.day += n;
}
void add_month(Date& date, int n) ){
        date.month += n;
}
void add_year(Date& date, int n) ){
        date.year += n;
}
```

```
//sử dụng
void main(){
        Date d;
        set_date(d,32,13,2010);
        add_day(d,5);
        add_year(d,1);
        d.month = 13;
}
```

Truy nhập biến thành viên từ bên ngoài

2 Định nghĩa lớp [...]

Xét kiểu dữ liệu có cấu trúc [...]

```
sruct Date{
        int day, month, year;
        void set_date(int d, int m,
int y){
                day = d;
                month = m;
                year = y;
        }
        void add_day(int n){
                day += n;
        }
        void add_month(int n) ){
                month += n;
        }
        void add_year(int n) ){
                year += n;
        }
};
```

```
//sử dụng
void main(){
        Date d;
        d.set_date(32,13,2010);
        d.add_day(5);
        d.add_year(1);
        d.month = 13;
}
```

Truy nhập biến thành viên từ bên ngoài

Các vấn đề với struct

- Truy nhập trực tiếp vào các biến thành viên của cấu trúc → không an toàn
- Khi có sự thay đổi tên của các biến thành viên → người sử dụng phải thay đổi lại mã chương trình ứng dụng
- Ví dụ: thay đổi lại cấu trúc Date

```
struct Date{
  int d, m, y;
};
```

- Thì đoạn mã sau sẽ có lỗi khi biên dịch

```
Date d;
d.month = 10;
```

@ Lớp hóa

Định nghĩa lớp Date

```
class Date{
        int day;
        int month;
        int year;
public:
        void set_date(int d, int m, int
        y){
                day = d;
                month = m;
                year = y;
        }
        int get_day()   {   return day;
        }
        int get_month() {   return month;
        }
        int get_year()  {   return year;
        }
        void add_year(int n) ){
                year += n;
        }
};
```

Tên lớp

Biến thành viên

Kiểm soát quyền truy nhập

Hàm thành viên

@ Lớp hóa [...]

Sử dụng lớp Date

```
void main(){
        Date d;
        d.set_date(1,1,2010);
        d.add_year(10);
        d.day = 10; //??
        d.month = 10; //??
        d.year = 2009;//??
        int i = d.day;//??
        int day = d.get_day();
        int month = d.get_month();
        int year = d.get_year();
}
```

Đối tượng

Có lỗi. *day, month, year là các biến thành viên của Date thuộc kiểu không được phép truy nhập từ bên ngoài (kiểu private)*

Thay đổi tên biến thành viên của lớp

```
class Date{
        int d;
        int m;
        int y;
public:
        void set_date(int _d, int _m,
        int _y){
                d = _d;
                m = _m;
                y = _y;
        }
        int get_day()   {   return d;
        }
        int get_month() {   return m;
        }
        int get_year()  {   return y;
        }
        void add_year(int n) ){
                y += n;
        }
};
```

```
//Sử dụng
void main{
        Date d;
        d.set_date(1,1,2010);
        d.add_year(10);
        int day = d.get_day();
        int month =
d.get_month();
        int year =
d.get_year();
}
```

Khai báo các thành viên của lớp

Việc khai báo các thành phần của lớp có dạng như sau:

```
class <Tên lớp>{
private:
        <Khai báo các thành phần riêng>
protected:
        <Khai báo các thành phần được bảo vệ>
public:
        <Khai báo các thành phần công cộng>
};
```

3. Biến thành viên

- Khai báo biến thành viên của lớp

```
class Date{
   int day,month, year;   //khai báo tương tự như cấu trúc
   ...
};
```

- Mặc định các biến thành viên không truy nhập được từ bên ngoài

```
Date d;
d.day = 10;;      //Lỗi, vì biến thành viên day của Date thuộc kiểu private
```

- Có thể cho phép biến thành viên truy nhập từ bên ngoài bằng cách chuyển thành biến public. Tuy nhiên, ít khi sử dụng như vậy vì không còn che giấu dữ liệu

```
class Date{
public:
    int day,month, year;   //truy nhập được từ bên ngoài
    ...
};
```

3. Biến thành viên

- **Truy nhập các biến thành viên thông qua các hàm thành viên**

```
class Date{
  int day, month,year;
public:
  int get_day()   {   return day;  }
  void set_day(int d){ day = d;    }
};
```

Hàm thành viên để truy nhập biến thành viên

- **Khởi tạo biến thành viên thông qua hàm tạo**

```
class Date{
  int day, month,year;
public:
  Date(int d, int m, int y){ //hàm tạo
      day = d; month = m; year = y;
   }
   ...
};
```

4. Hàm thành viên

Khai báo và định nghĩa hàm thành viên

```
class Date{
  int day, month,year;
public:
  int get_day()   {   return day;  }
  void set_day(int d);
  . . .
};
void Date::set_day(int d){
   day = d;
}
```

Khai báo và định nghĩa

Chỉ khai báo, không định nghĩa

Định nghĩa bên ngoài phần khai báo lớp

Để che giấu cách thực hiện, hàm thành viên thường được khai báo trong tập tin đầu (*.h), phần định nghĩa được thực hiện trong tệp tin nguồn (*.cpp). Khi đóng gói thành thư viện, người sử dụng chỉ cần tệp tin thư viện (*.lib) và tệp tin đầu (*.h), không cần tệp tin nguồn (*.cpp)

4. Hàm thành viên

- **Khai báo hàm thành viên trong tệp tin đầu (*.h)**

```
//tệp tin Date.h
class Date{
  int day, month,year;
public:
  int get_day();
  void set_day(int d);
  . . .
};
```

- **Định hàm thành viên trong tệp tin nguồn (*.cpp)**

```
//tệp tin Date.cpp
void Date::set_day(int d){
    day = d;
}
void Date::get_day(){ return day;}
```

Con trỏ đối tượng

Sử dụng con trỏ đối tượng

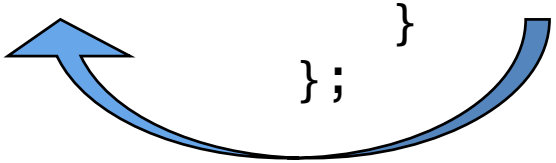
```
void main{
      Date d;
      d.set_date(1,1,2010);
      Date *pd = &d;                    //con trỏ pd trỏ vào đối tượng d
      pd->set_date(1,1,2010);
      Date *pd1 = new Date;  //cấp phát bộ nhớ, gọi hàm tạo
      pd1->set_day(10);
      . . .
      delete pd1;             //hủy bộ nhớ
      Date *pd2 = new Date[5];
      for(int i = 0 ;i < 5; i++)
            pd2[i].set_date(1,1,2010);
      . . .
      delete [] pd2;
}
```

5. Kiểm soát truy nhập

- private: các thành viên chỉ có thể truy nhập từ các thành viên của lớp và từ các bạn bè của lớp
- public: các thành viên công cộng, truy nhập được ở mọi nơi
- protected: các thành viên không truy nhập được từ bên ngoài, nhưng truy nhập được từ bạn hoặc từ lớp dẫn xuất

```
class A{
protected:
        int a;
        void f();
};
```

```
class B: public A{
public:
        void g(){
        f();
    }
};
```



Lớp dẫn xuất

Lớp cơ sở

@ Bài tập

Bài 1: Xây dựng một lớp tên Complex để đại diện cho một số phức với các yêu cầu:

- Thành phần thực và ảo đều là số thực float
- Hàm nhập thành phần thực, thành phần ảo từ bàn phím
- Hàm hiển thị thành phần thực và ảo ra màn hình
- Hàm lấy thành phần thực, hàm gán thành phần thực
- Hàm lấy thành phần ảo, hàm gán thành phần ảo
- Hàm đồng thời lấy thành phần thực, thành phần ảo
- Hàm đồng thời gán thành phần thực, thành phần ảo
- Hàm tính tổng của hai số phức m
- Viết chương trình chính minh họa cách sử dụng

3.6 Bài tập

Bài 2. Xây dựng một lớp tên person để đại diện cho một người với các yêu cầu:

- Tên có độ dài tối đa 50 ký tự
- Ngày, tháng, năm sinh có kiểu int (hoặc thuộc kiểu Date như đã gợi ý trong bài giảng).
- Quê quán có độ dài tối đa 100 ký tự
- Hàm nhập tên, ngày sinh, quê từ bán phím
- Hàm hiển thị thông tin ra màn hình
- Hàm lấy tên, hàm gán tên
- Hàm lấy quê quán, hàm gán quê quán
- Hàm lấy ngày sinh, hàm gán ngày sinh
- Viết chương trình chính minh họa cách sử dụng

6. Hàm tạo và hàm hủy

Vấn đề 1: Nghiên cứu đoạn mã sau

```
class Date{
    int day, month,year;
public:
    int get_day()   {   return day;  }
    void set_day(int d){ day = d;    }
};
void main(){
    Date d;
    int i = d.get_day();
}
```

Câu hỏi:

$d.day = ?$

$d.month = ?$

$d.year = ?$

- Làm thế nào để sau khi được tạo ra, đối tượng có trạng thái ban đầu theo ý muốn của người sử dụng?
- Giải pháp: sử dụng hàm tạo

6. Hàm tạo và hàm hủy [...]

Vấn đề 2: đối tượng sử dụng bộ nhớ động

```
class Array{
    int n;        //số phần tử của array
    int *data;    //mảng chứa giá trị các phần tử
public:
    ...
};
```

- Câu hỏi: làm thế nào để cấp phát bộ nhớ và hủy bộ nhớ cho biến thành viên data một cách an toàn
- Giải pháp: sử dụng hàm tạo và hàm hủy.

6. Hàm tạo và hàm hủy [...]

- **Hàm tạo:** luôn được gọi khi đối tượng được tạo ra
- **Hàm hủy:** luôn được gọi khi đối tượng bị hủy
- **Cú pháp:**

```
class A{
    int a, b;
public:
    A(){a = 0; b = 0;}          (1)
    A(int _a){ a = _a;}         (2)
    A(int _a, int _b){ a = _a; b = _b;}   (3)
    ~A();
};
```

- **Một lớp có thể có nhiều hàm tạo**
  - Hàm tạo 1: hàm tạo không đối
  - Hàm tạo 2: hàm tạo một đối
  - Hàm tạo 3: hàm tạo hai đối
- **Một lớp chỉ có duy nhất một hàm hủy**

6. Hàm tạo và hàm hủy [...]

Sử dụng

```
void main{
      A a();       //gọi hàm tạo (?)
      A a1(10);    //gọi hàm tạo (?)
      A a2(1,2);   //gọi hàm tạo (?)
}                  //gọi hàm hủy cho ?

void f(A a){
      A b(0,0);
      if(...){
       A c;
       ...
      }

}

void main{
      A *a = new A(10);      //gọi hàm tạo (?)
      . . . //sử dụng
      delete a;              //gọi hàm hủy
}
```

gọi hàm tạo (?)

gọi hàm tạo (?)

gọi hàm hủy cho ?

gọi hàm hủy cho ?

6. Hàm tạo và hàm hủy [...]

- Làm thế nào để không phải định nghĩa nhiều hàm tạo như ví dụ trên?
- **Giải pháp**: sử dụng hàm tạo có tham biến mặc định → một lớp chỉ cần một hàm tạo duy nhất

```
class A{
        int a, b;
public:
        A(int _a = 0, int _b = 0) { a = _a; b = _b;}
    . . .
};

void main{
      A a1;                    //a1.a = ?; a1.b = ?
      A a2(1);                 //a2.a = ?; a2.b = ?
      A a3(1,2);               //a3.a = ?; a3.b = ?
};
```

Tóm tắt về hàm tạo và hàm hủy

- Hàm tạo được sử dụng để:
  - Cấp phát bộ nhớ động
  - Khởi tạo các trạng thái ban đầu cho đối tượng
- Một lớp có thể có nhiều hàm tạo. Chúng khác nhau ở số lượng các tham số hoặc kiểu của các tham số.
- Nếu không định nghĩa hàm tạo thì compiler sẽ tự động sinh ra một hàm tạo với mã thực thi là rỗng, dẫn đến:
  - ✓ Trạng thái ban đầu của các biến thành viên là bất định
  - ✓ Không cấp phát bộ nhớ động cho các biến thành viên dạng mảng động
- Hàm hủy là duy nhất
- Hàm hủy không bao giờ có đối
- Nếu không định nghĩa hàm hủy thì compiler cũng tự động sinh ra nhưng mã thực thi của hàm hủy này là rỗng.
- Khi sử dụng đối tượng động (có sử dụng toán tử new) thì luôn phải nhớ hủy bộ nhớ đã cấp phát cho bộ nhớ động khi không cần dùng đến chúng nữa (sử dụng toán tử delete)
- Hàm tạo và hàm hủy có thể được định nghĩa bên ngoài phần khai báo lớp.

Ví dụ về lớp Array

```
//khai báo lớp Array
class Array{
  int n;
  int *data;
public:
 Array(int _n = 0, int _d = 0);
  ~ Array();
    . . .
};

// định nghĩa hàm tạo và hàm hủy
Array :: Array(int _n, int _d){
    n = _n;
    data = new int[n];
  for(int i = 0; i < n; i++)
      data[i] = _d;
}
Array ::~ Array(){
  delete [] data;
}
```

```
//sử dụng
void main(){
    Array a(5);
    Array *pa = new Array(5,1);
     ...
     delete pa;
}
```

Câu hỏi:

1. *Các giá trị của mảng data của a, pa bằng bao nhiêu?*
2. *Không sử dụng delete pa có được không?*
3. *Biến a được hủy khi nào?*

7. Hàm tạo bản sao

- Hàm tạo bản sao được gọi khi sao chép đối tượng (xem các ví dụ sau)
- Cú pháp chuẩn:

```
class A{
        int a, b;
public:
        A(const A& a1);
    . . .
};
```

Sao chép tham số từ a1, a1 không bị thay đổi do vô tình

```
//định nghĩa hàm sao chép
A::A(const A& a1){
        a = a1.a;
        b = a1.b;
}
//sử dụng
void main(){
        A a;
        A a1(a);
}
```

Gọi hàm tạo bản sao

7. Hàm tạo bản sao [...]

Hàm tạo bản sao được gọi khi sao chép đối tượng:

- Khi khai báo các biến x2-x4 như sau:

```
X x1;
X x2(x1);
X x3 = x1;
X x4 = X(x1);
```

- Khi truyền tham số qua giá trị cho một hàm

```
void f(X x) { ... }
void main(){
    X a;
    f(a);
}
```

ở đây có sự gọi hàm tạo bản sao để sao chép nội dung của a để truyền vào cho tham biến hình thức x của hàm f

- Khi một hàm trả về một đối tượng

```
X f() {
    X x1;
    . . . //thực hiện thuật toán
    return x1;
}
void main(){
    X x = f();
    . . .
}
```

ở đây có sự gọi hàm tạo bản sao để sao chép nội dung của biến tạm x1 sang cho biến x sau khi thực hiện xong lệnh return trong hàm f

7. Hàm tạo bản sao [...]

- Nếu không định nghĩa hàm tạo bản sao thì compiler sẽ tự sinh ra và sao chép từng bít
- Lớp không có tham biến được cấp phát động thì không cần định nghĩa hàm tạo bản sao
- Khi có tham biến được cấp phát động thì bắt buộc định nghĩa lại hàm tạo bản sao.

```
//khai báo lớp Array không có hàm tạo bản sao
class Array{
      int n;
      int *data;
public:
     Array(int _n = 0; int _d = 0) {...}
      ~ Array(){...}
      void set_data(int i, int d){
           if((i>=0) && (i<n))
                 data[i] = d;
};
```

7. Hàm tạo bản sao [...]

```
//sử dụng
void main{
      Array a(5,0);
       Array b(a);
       ...
      b.set_data(0,10);
};
```

Gọi hàm tạo bản sao để sao chép a sang b

Hỏi: a.data[0] = ?

- Do không định nghĩa hàm tạo bản sao, nên ở đây gọi hàm tạo bản sao mặc định do compiler sinh ra, hàm tạo này có dạng:

```
Array :: Array(const Array & a){
        n = a.n;
        data = a.data;
}
```

- Khi sử dụng Array b(a); thì mảng data của a và b là một, nên khi thay đổi b thì a sẽ thay đổi theo.
- **Giải pháp:** định nghĩa lại hàm tạo bản sao

7. Hàm tạo bản sao [...]

- Định nghĩa hàm tạo bản sao cho lớp Array như sau

```
Array :: Array(const Array & a) {
      n = a.n;
      data = new int[n];
      for (int i=0; i < n; ++i)
        data[i] = a.data[i];
    }
```

- Khi một lớp phải định nghĩa hàm hủy thì cũng cần thiết định nghĩa lại hàm tạo bản sao
- Trong trường hợp muốn cấm sao chép thì ta khai báo hàm tạo bản sao trong phần private.

Con trỏ *this*

- Từ khóa this được dùng trong khi định nghĩa các hàm thành viên dùng để trỏ đến đối tượng hiện tại
- Nói chung, con trỏ this ít khi được sử dụng tường minh, vì nó đã được ngầm sử dụng khi truy nhập vào các thành phần dữ liệu. Nó thường được sử dụng khi chúng ta muốn lấy địa chỉ của đối tượng hiện tại (như để trỏ vào chính đối tượng đó)

8. Hàm toán tử gán

Nghiên cứu ví dụ 1:

```
class A{
        int a, b;
public:
        A(int _a, int _b):a(_a), b(_b){} //hàm tạo
    . . .
};

void main(){
        A a(1,2);
        A b;
        b = a;
}
```

Gọi hàm toán tử gán (=).
ở đây sẽ có:
b.a = a.a
b.b = a.b

OK

Không định nghĩa hàm toán tử gán, compiler sẽ tự động sinh ra và gán từng bít (giống với hàm tạo bản sao)

8. Hàm toán tử gán [...]

Nghiên cứu ví dụ 2:

```
class Array{
        int n;
        int *data;
public:
        Array(int _n = 0; int _d = 0) {...}
        ~ Array(){...}
        ...
};

Void main() {
        Array a(5,1);
        Array a1;
        a1 = a;
}
```

Gọi hàm toán tử gán (=).
ở đây sẽ có:
a1.n = a.n
a1.data = a.data

- a1.data và a.data cùng trỏ vào một vùng nhớ → kết quả tương tự với trong trường hợp hàm tạo bản sao ở trên.
- Trong trường hợp này cần định nghĩa hàm toán tử gán

8. Hàm toán tử gán [...]

Cú pháp chuẩn hàm toán tử gán:

```
class A{
        ...
public:
        A& operator=(const A&);    //khái báo hàm toán tử gán
   . . .
};

//định nghĩa hàm toán tử gán:
A& A::operator=(const A& a1){
        ...//mã gán các biến thành viên
}
```

8. Hàm toán tử gán [...]

Ví dụ định nghĩa hàm toán tử gán cho lớp Array

```
class Array{
      int n;
      int *data;
public:
      Array(int _n = 0; int _d = 0) {...}
      ~Array(){...}
      Array& operator=(const Array& a);
};

Array& Array::operator=(const Array& a) {
   if (n != a.n) {
        delete [] data;
        n = a.n;
        data = new int[n];
   }
   for (int i=0; i < n; ++i)
        data[i] = a.data[i];
   return *this;
}
```

Bài tập

Định nghĩa lớp Array có các yêu cầu sau

- Hàm tạo, hàm hủy, hàm tạo bản sao, hàm toán tử gán
- Các hàm cho phép nhập dữ liệu vào từ bàn phím và hiển thị ra màn hình cho Array
- Các hàm cho phép gán/lấy (ghi/đọc) giá trị của một phần tử nào đó trong Array
- Viết chương trình chính minh họa cách sử dụng

9. Thành viên tĩnh

Biến thành viên tĩnh

- Vấn đề: Yêu cầu ghi lại số lượng các đối tượng được tạo ra từ lớp Date

```
void main(){
    Date d1(1,1,2010); // count++
    Date d2 = d1;      // count++
    …
}
```

Khai báo biến tĩnh

Định nghĩa biến tĩnh: bắt buộc và nằm bên ngoài khai báo lớp và ngoài các hàm

- Giải pháp: đưa biến count là một biến static của lớp Date

```
class Date{
    int day, month,year;
    static int count;
public:
    Date(int d, int m, int y);
    Date(const Date& d);
    ~Date();
    . . .
};
int Date:: count = 0;
```

```
Date::Date(int d, int m, int y){
    day = d; month = m; year = y;
    count++;
}
Date::Date(const Date& d){
    day = d.day; month = d.month;
    year = y.year; count++;
}
Date::~Date(){
    count--;
}
```

9. Thành viên tĩnh [...]

Hàm thành viên tĩnh

```
class A{
    int n;
     static int count;
public:
     A():n(0){}
    void f();
     static void g();
     . . .
};
int A:: count = 0;
void A::f(){
     n++;
}
void A::g(){
    n = 2;  //???
     f(x);   //???
     count = 2;  //OK
}
```

Hàm thành viên tĩnh là hàm chung cho cả lớp không phải riêng cho một đối tượng nào

Trong hàm thành viên tĩnh chỉ sử dụng được các biến thành viên tĩnh và chỉ gọi được các hàm thành viên tĩnh khác. Nếu muốn truy nhập vào các biến thành viên của lớp thì phải khai báo một đối tượng trung gian, vì trong hàm static không có đối tượng ngầm định *this.

9. Thành viên tĩnh [...]

Kết luận về thành viên tĩnh

- Được cấp phát một vùng nhớ cố định, tồn tại ngay cả khi lớp chưa có một đối tượng nào
- Chung cho cả lớp, không phải của riêng mỗi đối tượng
- Để biểu thị thành phần tĩnh ta dùng "tên lớp :: tên thành viên tĩnh" hoặc "tên đối tượng . Tên thành viên tĩnh"
- Được cấp phát bộ nhớ và khởi gán giá trị ban đầu bên ngoài khai báo lớp và ngoài các hàm (kể cả hàm main)

9. Thành viên tĩnh [...]

Bài tập: Xây dựng lớp HD (hóa đơn) gồm 2 dữ liệu là mshd (mã số hóa đơn) và tienban với các hàm thực hiện chức năng sau:

- Hàm tạo hóa đơn
- Hàm hủy hóa đơn
- Hàm sửa nội dung hóa đơn (sửa tiền bán)
- Hàm in ra tổng số hóa đơn và tổng số tiền bán sau các thao tác tạo, hủy, sửa hóa đơn.
- Viết hàm main để ứng dụng

@ 10. Friend

Nghiên cứu vấn đề:

```
class A{
    int n;
public:
    A():n(0){}
    . . .
};
class B{
    int m;
public:
    B():m(0){}
    void f(A a){ a.n = 5;}//???
    . . .
};
void g(A a){
    a.n = 10; //???
}
```

Làm thế nào để hàm phi thành viên, hàm thành viên của một lớp khác có thể truy nhập trực tiếp vào biến thành viên của một đối tượng?

10. Friend [...]

- **Giải pháp: khai báo bạn bè – friend**
- **Cái gì có thể là friend?**
  - Hàm phi thành viên định nghĩa ở bên ngoài
  - Hàm thành viên của một lớp khác
  - Cả lớp khác

```
class A{
    int n;
public:
    A():n(0){}
    friend void g(A a);
    friend void B::f(A a);
    friend class C;
    . . .
};
void g(A a){
    a.n = 10; //OK
}
```

```
class B{
    int m;
public:
    B():m(0){}
    void f(A a){ a.n = 5;}//OK
    . . .
};
```

11. Nạp chồng toán tử

- Ví dụ 1:

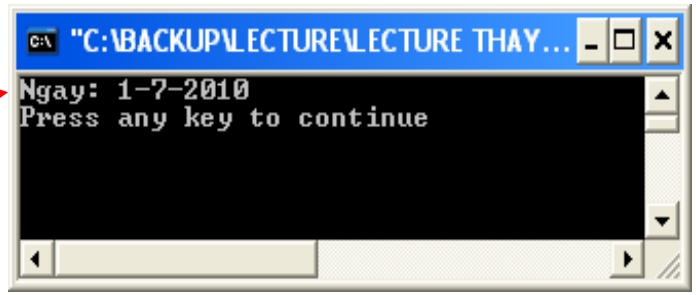
```
#include <iostream.h>
class Date{
    int day, month,year;
public:
    Date(int d, int m, int y){
        day = d; month = m; year = y;
    }
};
void main(){
    int n = 5;
    cout<< n;    //OK
    Date d(1,1,2010);
    cout<<d;     //Lỗi
}
```

Tại sao lỗi? Làm thế nào để có thể sử dụng được như thế?

- Biến n thuộc kiểu cơ sở, thư viện xuất/nhập đã hỗ trợ hàm toán tử xuất ra màn hình cho các biến cơ sở đó. Biến d thuộc kiểu Date do người sử dụng định nghĩa.
- **Giải pháp**: định nghĩa lại toán tử xuất cho lớp Date

Nạp chồng toán tử xuất cho lớp Date

```
#include <iostream.h>
class Date{
    int day, month,year;
public:
    Date(int d=1, int m=1, int y=2010){
        day = d; month = m; year = y;
    }
    friend ostream& operator<<(ostream& os, const Date& d);
};
ostream& operator<<(ostream& os, const Date& d){
   os<<“Hay hien thi ngay thang nam: ”;
os<<“Ngay: ”<<d.day<<“-”<<d.month<<“-”<<d.year<<“\n”;
    return os;
}
void main(){
    Date d(1,7,2010);
    cout<<d;
}
```



11. Nạp chồng toán tử [...]

Ví dụ 2:

```
#include <iostream.h>
class Complex{
    int real,imag;
public:
    Complex(int r=0, int i=0){
        real = r; imag = i;
    }
};
void main(){
    Complex a(1,2), b(5,6);
    Complex c;
    c = a + b; //Lỗi
}
```

Lý do tương tự như lớp Date ở trên

Nạp chồng toán tử cho lớp số phức complex

```
#include <iostream.h>
class Complex{
    int real,imag;
public:
    Complex(int r = 0, int i =0): real(r),imag(i) {}
    Complex operator+(const Complex& b) const {
           Complex z(real + b.real, imag + b.imag);
           return z;
    }
    Complex operator-(const Complex& b) const {
           return Complex(real - b.real, imag - b.imag);
    }
    Complex operator*(const Complex&) const;
    Complex operator/(const Complex&) const;
    Complex& operator +=(const Complex&);
    Complex& operator -=(const Complex&);
  ...
};
```

Yêu cầu: sinh viên hãy thực hiện nốt các hàm còn lại

11. Nạp chồng toán tử [...]

Nạp chồng các toán tử cho lớp complex sử dụng hàm bạn

```
    #include <iostream.h>
    class Complex{
        int real,imag;
    public:
        Complex(int r = 0, int i =0): real(r),imag(i) {}
    friend Complex operator+(const Complex&, const Complex&) const;
    friend Complex operator-(const Complex&, const Complex&) const;
    friend Complex operator*(const Complex&, const Complex&) const;
    friend Complex operator/(const Complex&, const Complex&) const;
  ...
};
Complex operator+(const Complex& a,const Complex& b) const{
        Complex z(a.real + b.real, a.imag + b.image);
        return z;
}
...
```

11. Nạp chồng toán tử [...]

- **Các phép toán có thể nạp chồng**
  - Các toán tử số học: ++ -- + - * / % += -= ...
  - Các toán tử logic, logic bit: && || ! & &= | |= ...
  - Các toán tử so sánh: == != > < >= <=
  - Các toán tử thao tác bit: << >> >>= <<=
  - Các toán tử khác: [] (), ...
- **Các toán tử sau không nạp chồng được:**
  - Toán tử truy nhập phạm vi (dấu hai chấm đúp): ::
  - Toán tử truy nhập thành viên cấu trúc (dấu chấm): .
  - Toán tử gọi hàm thành viên qua con trỏ: *->
  - Toán tử điều kiện : ? :

Bài tập

1. Hãy nạp chồng các toán tử +,-,*, +=, -=, *=, ==, !=, >, < sử dụng hàm thành viên, toán tử nhập/xuất cho lớp số complex sử dụng hàm bạn. Viết chương trình chính sử dụng lớp này
2. Bổ sung các yêu cầu sau vào lớp Array
   - Hàm nạp chồng toán tử [] (lấy giá trị của một phần tử),
   - Định nghĩa toán tử nhập, xuất một array
   - Định nghĩa toán tử gọi hàm () để lấy ra giá trị max của mảng.
   - Nạp chồng toán tử +, -, * hai array, hoặc array với một số
   - Viết chương trình chính sử dụng lớp này