

KỸ THUẬT LẬP TRÌNH C/C++

Lớp và đối tượng

Thi-Lan Le

Thi-Lan.Le@mica.edu.vn; lan.lethi1@hust.edu.vn

[Webpage: http://www.mica.edu.vn/perso/Le-Thi-Lan](http://www.mica.edu.vn/perso/Le-Thi-Lan)

Các nội dung chính

- ◆ Lớp và các thao tác đối với lớp
- ◆ Sử dụng các đối tượng
- ◆ Hàm thành viên
- ◆ Con trỏ *this*
- ◆ Hàm bạn (friend function)
- ◆ Định nghĩa lại các toán tử trong lớp
- ◆ Cấp phát động bộ nhớ
- ◆ Các thành phần kiểu *static*

Định nghĩa một lớp mới

- ◆ Định nghĩa một lớp mới cho phép tạo ra một lớp mới, bao gồm các thành phần dữ liệu và các hàm thành viên cần thiết.
- ◆ Cú pháp:

```
class <Tên lớp>
{
    //Đn các thành phần dữ liệu
    <E:> <type> d1; ...
    //Đn các hàm thành viên
    <E:> <type> f1();...
};
```

E: từ khóa xác định mức độ che dấu (hay thuộc tính truy xuất): *private*, *public* hoặc *protected*

Type: kiểu dữ liệu hoặc kiểu hàm và có thể là tên lớp

Vị trí đặt đ/n lớp: có thể trước hoặc sau hàm *main()*.

Không được đ/n một lớp trong một lớp khác

Ví dụ về đ/n lớp: Program 9.1

```
class Circle {  
    private:  
        static const float PI=3.1415; //Hằng số tĩnh, hằng số của lớp  
        float r;           //Bán kính, thành phần dữ liệu của từng đối tượng  
  
    public:  
        void setRadius(float re)  
        {  
            r=re;  
        }  
        float getRadius()  
        {  
            return r;  
        }  
        float area()  
        {  
            return PI*r*r;  
        }  
};
```

Khai báo lớp

```
class Circle; //Khai báo lớp
```

```
int main()
```

```
{
```

```
    Circle c, c2; //Khai báo đối tượng thuộc lớp
```

```
    c.setRadius(10);
```

```
    c2.setRadius(10);
```

```
    ...
```

```
}
```

```
class Circle {... //Định nghĩa lớp đưa ra sau hàm main
```

```
};
```

Sử dụng các đối tượng

- ◆ Các thao tác cơ bản cho đối tượng:
 - **Khai báo:** là thao tác đầu tiên để sử dụng được một đối tượng
 - **Truy nhập vào các thành phần:** có 2 cách:
 - Sử dụng toán tử “.” cho đối tượng thông thường,
 - Sử dụng toán tử “->” cho đối tượng kiểu con trỏ.

Mở rộng Program 9.1

//Đ/n lớp Circle

```
int main()
```

```
{
```

```
    Circle c; //Khai báo và sử dụng đối tượng thông thường
```

```
    c.setRadius(10);
```

```
    cout<<"Area of circle with r = "<<c.getRadius()<<
```

```
        " is "<<c.area()<<endl;
```

```
    Circle *pc = &c; //Khai báo và sử dụng đối tượng kiểu con trỏ
```

```
    pc->setRadius(20);
```

```
    cout<<"Area of circle with r = "<<pc->getRadius()<<" is "<<
```

```
        pc->area()<<endl;
```

```
    return EXIT_SUCCESS;
```

```
}
```

Hàm thành viên (member functions)

- ◆ Phân biệt giữa hàm thành viên và hàm tự do
- ◆ Các thao tác cơ bản cho hàm thành viên
- ◆ Hàm tự thiết lập và hàm tự hủy

Hàm thành viên và hàm tự do

- ◆ **Hàm thành viên:** là hàm thuộc một lớp, và cũng sẽ thuộc về các đối tượng của lớp đó
- ◆ **Hàm tự do:** là các hàm được định nghĩa bên ngoài các lớp, chính là hàm con trong C.

Các thao tác cơ bản cho hàm thành viên

- ◆ Tương tự như hàm tự do, cũng có 2 thao tác cơ bản cho hàm thành viên:
 - **Khai báo:** chỉ khai báo phần đầu của hàm. Có sự khác biệt cơ bản trong việc khai báo giữa hàm thành viên và hàm tự do. Mục đích của việc khai báo hàm tự do là để chuẩn bị sử dụng (gọi) hàm đó. Còn việc khai báo hàm thành viên chỉ để chuẩn bị cho việc định nghĩa hàm này.
 - **Định nghĩa:** trong C++, định nghĩa hàm có thể được đặt bên trong lớp hoặc đưa ra ngoài.

Program 9.2: Xây dựng và sử dụng lớp Point

```
class Point { //Đ/n lớp Point
    float _x, _y;

    public:
        void setXY(float x, float y); //Khai báo hàm tv
        float getX(){ return _x; } //Đ/n hàm tv
        float getY(){ return _y; } //Đ/n hàm tv
        float distanceTo(Point p); //Khai báo hàm tv
};
```

Program 9.2 (tiếp)

//Đ/n các hàm t/v bên ngoài lớp

```
void Point::setXY(float x, float y){
```

```
    _x = x;
```

```
    _y = y;
```

```
}
```

```
float Point::distanceTo(Point p){
```

```
    float d = (p._x-_x)*(p._x-_x) + (p._y-_y)*(p._y-_y);
```

```
    return sqrt(d);
```

```
}
```

Program 9.2 (tiếp và hết)

```
int main()
{
    Point p1, p2;
    //p1._x = 10;
    p1.setXY(10,10);
    p2.setXY(20,20);
    cout<<"D="<<p1.distanceTo(p2)<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Hàm tự thiết lập (constructor)

- ◆ **Khái niệm:** hay còn được gọi ngắn gọn là “hàm tạo” của một lớp, là hàm sẽ tự động được gọi sau khi ta khai báo một đối tượng phù hợp của lớp đó.
- ◆ **Vai trò:** dùng để khởi tạo các giá trị ban đầu cần thiết cho các đối tượng
- ◆ **Một số tính chất:**
 - ❑ Có tên trùng với tên lớp
 - ❑ Không có kiểu hàm (**không phải** hàm kiểu void)
 - ❑ Mức độ che dấu là **public**
 - ❑ Có thể định nghĩa chồng hàm tạo cho một lớp

Hàm tự hủy (destructor)

- ◆ **Khái niệm:** còn được gọi là “hàm hủy” của một lớp, là hàm sẽ tự động được gọi ngay trước khi giải phóng một đối tượng thuộc lớp đó.
- ◆ **Vai trò:** dùng để giải phóng các tài nguyên mà đối tượng đã dùng, nhất là các vùng nhớ cấp phát động.
- ◆ **Một số tính chất:**
 - Tên có dạng: ~<tên lớp>()
 - Không có kiểu hàm (**không phải** hàm kiểu void)
 - Hàm không có tham số
 - Mức độ che dấu là **public**
 - Chỉ có thể định nghĩa 1 hàm hủy cho một lớp

Mở rộng Program 9.2 với các hàm tạo và hàm hủy

```
class Point {  
    float _x, _y;  
    public:  
        Point(float x=0, float y=0){ //Hàm tạo với tham số mặc định  
            _x=x;  
            _y=y;  
            cout<<"Goi ham tao"<<endl;  
        }  
        ~Point(){ //Hàm hủy  
            cout<<"Goi ham huy"<<endl;  
        }  
        void setXY(float x, float y);  
        float getX(){ return _x; }  
        float getY(){ return _y; }  
        float distanceTo(Point p);  
};
```


Mở rộng Program 9.2 (tiếp và hết)

```
int main()
{
    {   Point p1;
        Point p2(10);
        Point p3(20,20);
        cout<<"D12="<<p1.distanceTo(p2)<<endl;
        cout<<"D23="<<p2.distanceTo(p3)<<endl;
    }
    return EXIT_SUCCESS;
}
```

Hàm tự thiết lập sao chép (copy constructor)

- ◆ Là hàm tạo có một tham số là đối tượng p thuộc chính lớp đó, nhằm tạo ra một đối tượng có nội dung giống hệt như p.
- ◆ Cú pháp hàm tự thiết lập sao chép cho một lớp A:
 - public: <kiểu hàm> A (A & p); hoặc
 - public: <kiểu hàm> A (const A & p);

Mở rộng Program 9.2

```
class Point {  
    float _x, _y;  
    public:  
        Point(float x=0, float y=0){ //Hàm thiết lập  
            _x=x;  
            _y=y;  
        }  
        Point(Point & p){ //Hàm thiết lập sao chép  
            _x=p._x;  
            _y=p._y;  
        }  
        float getX(){ return _x; }  
        float getY(){ return _y; }  
};
```

Mở rộng Program 9.2 (tiếp theo và hết)

```
int main()
{
    Point p1(10,20) ;
    Point p2(p1);

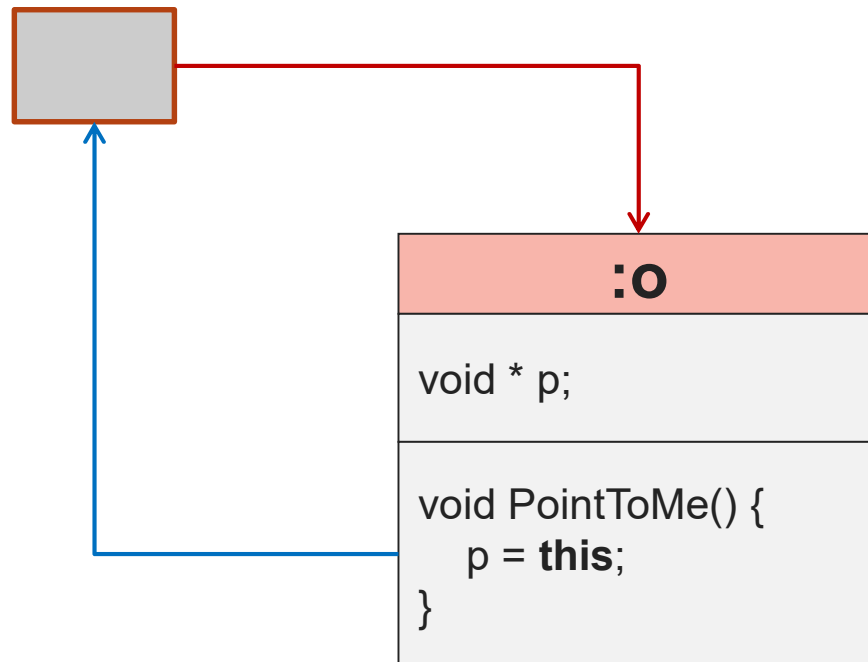
    cout<<"p1: X="<<p1.getX()<<" , Y="<<p1.getY()<<endl;
    cout<<"p2: X="<<p2.getX()<<" , Y="<<p2.getY()<<endl;

    return system("PAUSE"), EXIT_SUCCESS;
}
```

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)

Con trỏ *this*



Program 9.3: sử dụng *this*

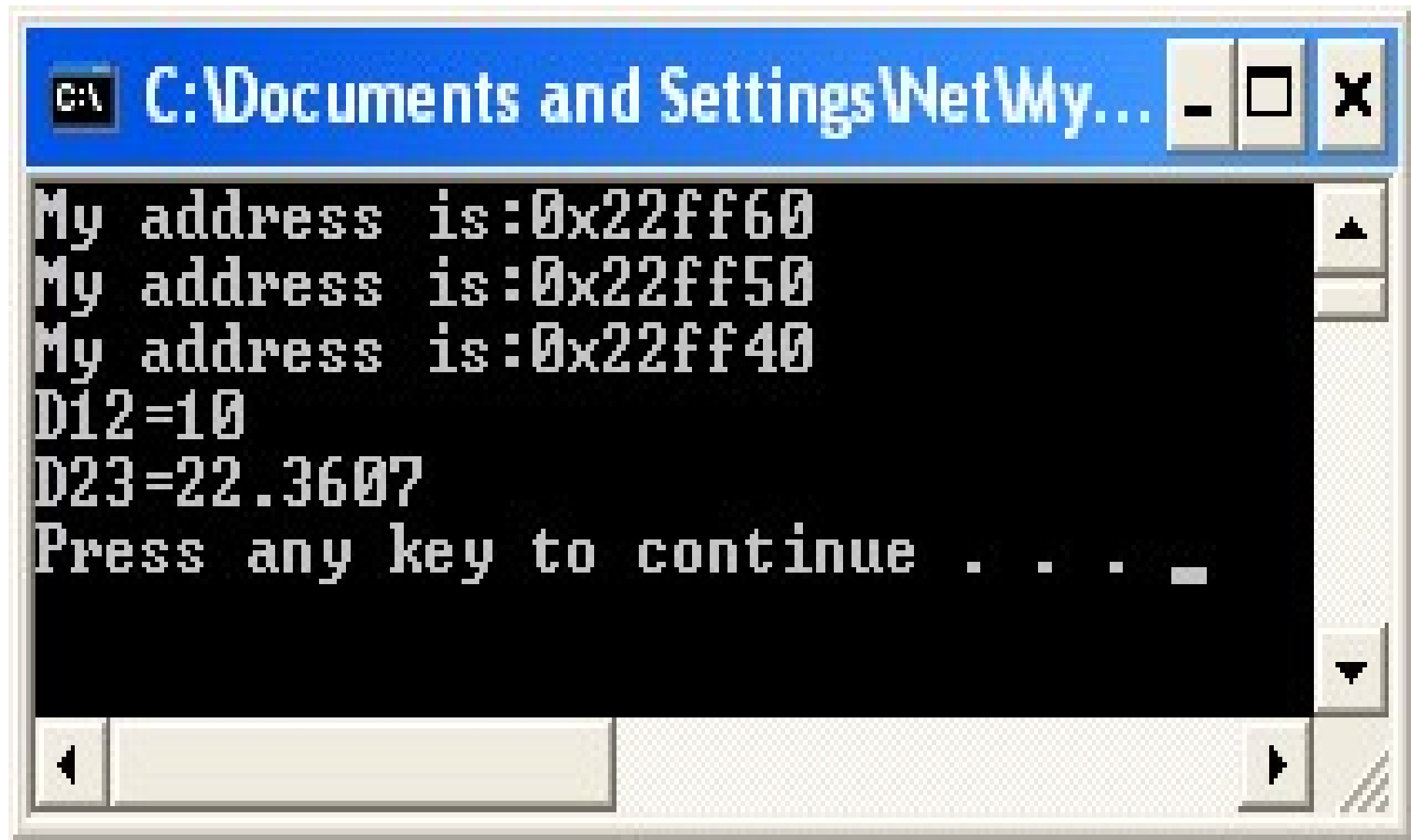
```
class Point {  
    float _x, _y;  
    void * self;  
  
    public:  
        Point(float _x=0, float _y=0){  
            this->_x=_x;  
            this->_y=_y;  
            self = this;  
            cout<<"My address is:"<<self<<endl;  
        }  
        float distanceTo(Point p);  
};
```

Program 9.3 (tiếp theo và hết)

```
int main()
{
    {    Point p1;
        Point p2(10);
        Point p3(20,20);
        cout<<"D12="<<p1.distanceTo(p2)<<endl;
        cout<<"D23="<<p2.distanceTo(p3)<<endl;
    }

    return system("PAUSE"), EXIT_SUCCESS;
}
```


Kết quả chạy

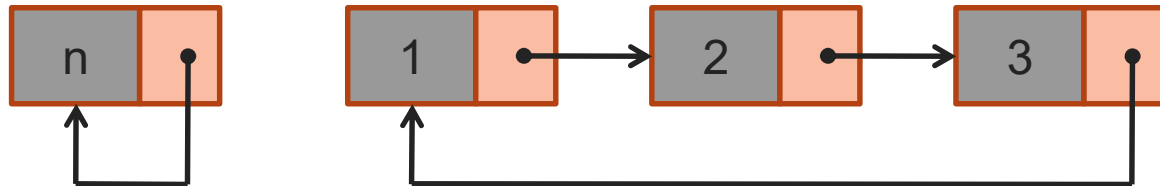


A screenshot of a Windows XP command prompt window. The title bar is blue and contains the text "C:\Documents and Settings\Net\My..." followed by standard window control buttons (minimize, maximize, close). The command prompt area has a black background with white text. It displays three lines of network addresses: "My address is:0x22ff60", "My address is:0x22ff50", and "My address is:0x22ff40". Below these are two lines of decimal values: "D12=10" and "D23=22.3607". The final line is "Press any key to continue . . .". The window has a scroll bar on the right and a status bar at the bottom with a left arrow, a text box, and a right arrow.

```
My address is:0x22ff60
My address is:0x22ff50
My address is:0x22ff40
D12=10
D23=22.3607
Press any key to continue . . .
```

Program 9.3: sử dụng **this** trong cấu trúc móc nối

```
class C {
    int n;
    C * next;
public:
    C (int m) {
        n = m;
        next = this;
    }
    void add(const C &a) {
        a.next = next;
        next = &a;
    }
    void show() {
        C * here = this;
        C * p = here;
        do {
            cout<<p->n<<" ";
            p = p->next;
        }
        while (p != here);
    }
};
```



```
int main() {
    C c1(1), c2(2), c3(3);
    c1.add(c2);
    c2.add(c3);
    c1.show();
    c2.show();
}
```

Kết quả:
1 2 3
2 3 1

Hàm bạn (friend function)

- ◆ **Khái niệm:** là loại hàm không phải là hàm thành viên của một lớp, nhưng có thể truy nhập vào các thành phần riêng tư của lớp đó
- ◆ Hàm bạn có thể là hàm tự do, hoặc là hàm thành viên của lớp khác
- ◆ Khi một lớp A muốn cho phép một hàm f là bạn của mình, thì trong A sẽ khai báo f là hàm bạn với từ khóa ***friend***.

Program 9.4

```
class Point {  
    float _x, _y;  
  
    public:  
        Point(float x=0, float y=0){  
            this->_x=x;  
            this->_y=y;  
        }  
  
        float getX(){ return _x; }  
        float getY(){ return _y; }  
        //Khai báo hàm tự do là hàm bạn  
        friend float distanceP2P(Point p, Point q);  
};
```

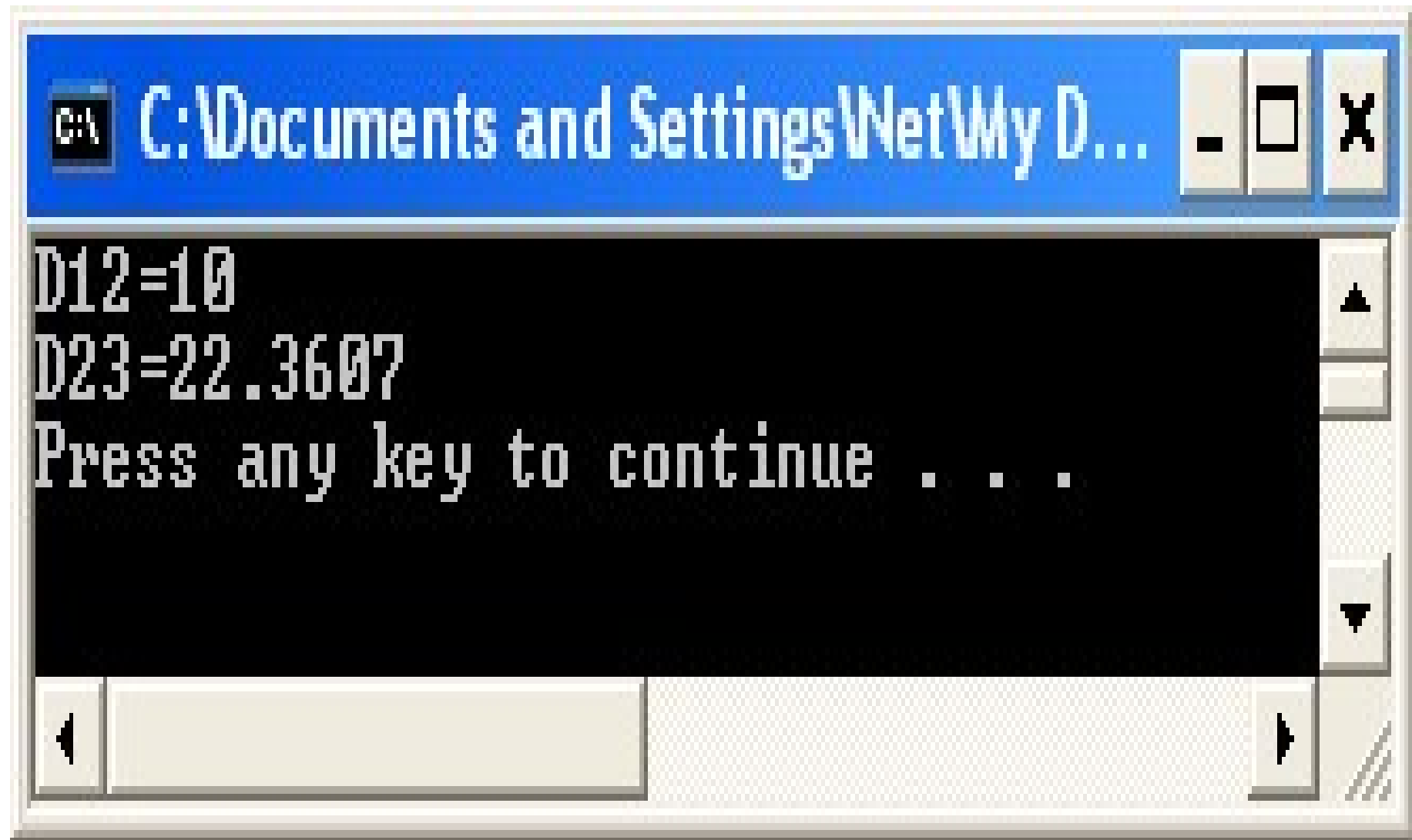
Program 9.4 (tiếp và hết)

//Đ/n hàm bạn cho phép truy nhập các thành phần riêng tư của lớp Point

```
float distanceP2P(Point p, Point q){  
    return sqrt((p._x-q._x)*(p._x-q._x) + (p._y-q._y)*(p._y-q._y));  
}
```

```
int main()  
{  
    Point p1, p2(10), p3(20,20);  
    cout<<"D12="<<distanceP2P(p1,p2)<<endl;  
    cout<<"D23="<<distanceP2P(p2,p3)<<endl;  
  
    return system("PAUSE"), EXIT_SUCCESS;  
}
```

Kết quả chạy chương trình



A screenshot of a Windows XP command prompt window. The title bar is blue and contains the text "C:\Documents and Settings\Net\My D..." followed by minimize, maximize, and close buttons. The command prompt itself has a black background with white text. It displays the output of a program: "D12=10", "D23=22.3607", and "Press any key to continue . . .". The window has a standard Windows XP interface with a taskbar at the bottom.

```
C:\Documents and Settings\Net\My D...  
D12=10  
D23=22.3607  
Press any key to continue . . .
```

Định nghĩa lại các toán tử trong lớp

- ◆ **Khái niệm:** là khả năng cho phép định nghĩa lại các toán tử đã có (như các phép toán số học, so sánh, nhập/xuất, v.v) trong một lớp mới, nhằm sử dụng các toán tử đó cho các đối tượng thuộc lớp này.
- ◆ Trong C++, sử dụng từ khóa ***operator*** và có 1 trong 2 cách để thực hiện
 - Sử dụng hàm thành viên của lớp đó
 - Sử dụng hàm tự do là hàm bạn của lớp đó

Program 9.5

```
class Phanso {  
    int tuso, mauso;  
    public:  
        Phanso(int ts=0,int ms=1){  
            tuso = ts;  
            mauso = ms;  
        }  
        Phanso operator*(Phanso p){ //Sử dụng hàm thành viên  
            Phanso q;  
            q.tuso = p.tuso* tuso;  
            q.mauso = p.mauso * mauso;  
            return q;  
        }  
        Phanso operator!() {  
            Phanso p;  
            p.tuso = mauso;  
            p.mauso = tuso;  
            return p;  
        };  
        //Sử dụng hàm bạn là hàm tự do  
        friend Phanso operator+(Phanso p, Phanso q);  
        friend ostream& operator<<(ostream & os, Phanso p);  
};
```


Program 9.5 (tiếp)

//Đ/n lại các toán tử, là các hàm tự do

```
Phanso operator+(Phanso p, Phanso q) {
```

```
    Phanso r;
```

```
    r.tuso = p.tuso*q.mauso + p.mauso*q.tuso;
```

```
    r.mauso = p.mauso*q.mauso;
```

```
    return r;
```

```
}
```

```
ostream& operator<<(const ostream & os, Phanso p) const {
```

```
    os<<p.tuso<<"/"<<p.mauso;
```

```
    return os;
```

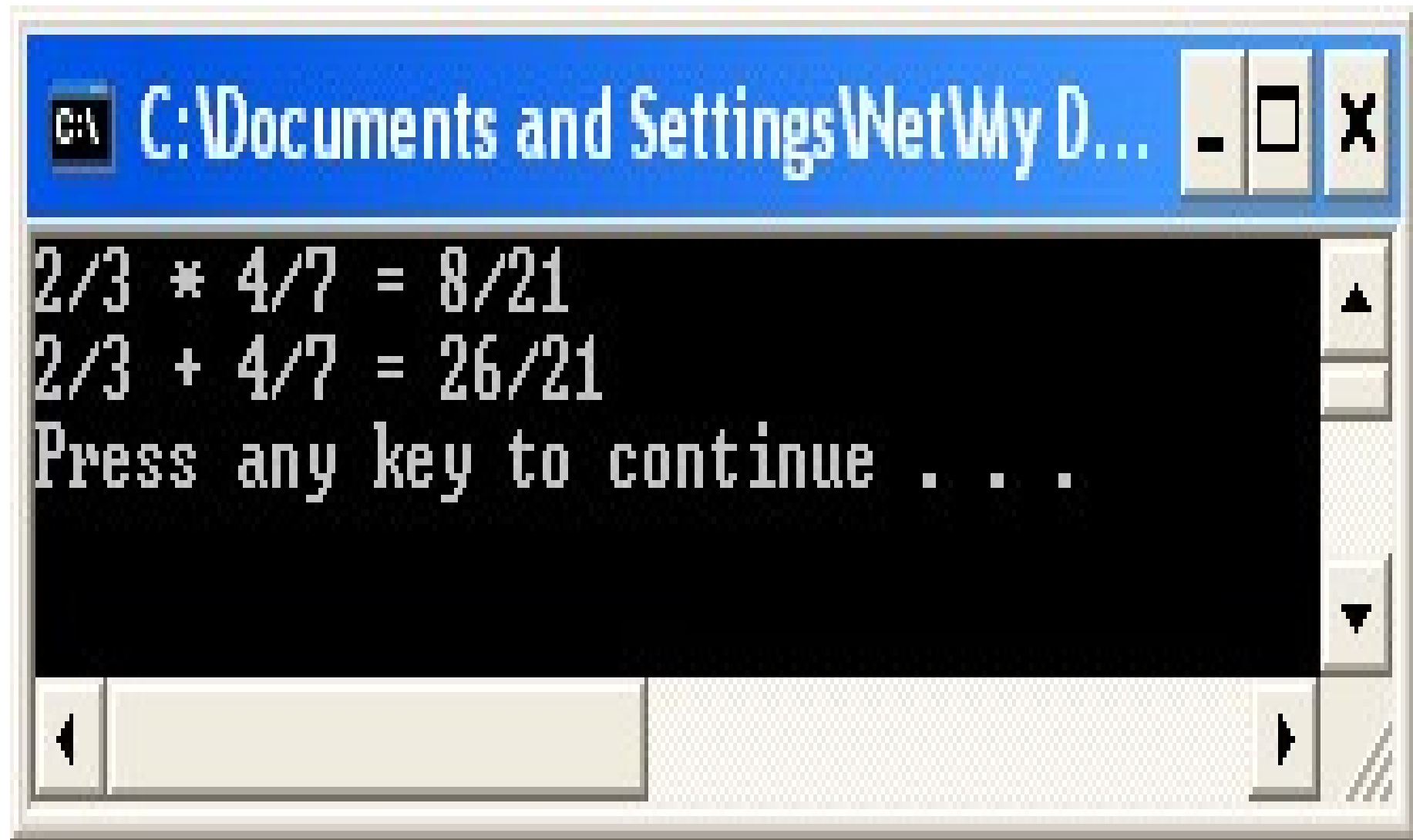
```
}
```

Program 9.5 (tiếp và hết)

```
int main()
{
    Phanso p(2,3),q(4,7);
    cout<<p<<" * "<<q<<" = "<<p*q<<endl;
    cout<<p<<" + "<<q<<" = "<<p+q<<endl;
    cout<<p*2 + q + 4;
    cout<<2*p; //Lỗi
    cout<<q+3;
    cout<<3+q;
    cout<<!p;

    return system("PAUSE"), EXIT_SUCCESS;
}
```

Kết quả chạy



A screenshot of a Windows XP command prompt window. The title bar is blue and contains the text "C:\Documents and Settings\Net\My D..." followed by standard window control buttons (minimize, maximize, close). The command prompt area has a black background with white text. It displays two fraction calculations: $2/3 * 4/7 = 8/21$ and $2/3 + 4/7 = 26/21$. Below these, it says "Press any key to continue . . .". The window has a scroll bar on the right and a status bar at the bottom.

```
C:\Documents and Settings\Net\My D...  
2/3 * 4/7 = 8/21  
2/3 + 4/7 = 26/21  
Press any key to continue . . .
```

Program 9.6: xây dựng mảng động – số phần tử của nó có thể thay đổi

```
class DynamicArray {  
    int * p;  
    unsigned int size;  
    public:  
        DynamicArray(unsigned int asize=0){  
            p = new int[asize];  
            size = asize;  
        }  
        ~DynamicArray(){  
            if (size>0) delete [] p;  
        }  
        //tiếp trang sau  
};
```

Program 9.6 (tiếp)

```
class DynamicArray {  
    void Init(){ //Hàm khởi tạo các giá trị ban đầu ngẫu nhiên cho mảng  
        if (size>0){  
            for (int i=0;i<size;i++)  
                p[i]=rand();  
        }  
    }  
    void ChangeSize(unsigned newsize){ //Hàm thay đổi kích thước mảng  
        if (size>0) delete [] p;  
        p = new int[newsize];  
        size = newsize;  
    }  
    //tiếp trang sau  
};
```

Program 9.6 (tiếp)

```
class DynamicArray {  
    void Show() { //Hàm in nội dung của mảng  
        if (size>0){  
            cout<<"Array with the size:"<<size<<endl;  
            for (int i=0;i<size;i++) cout<<p[i]<<" ";  
            cout<<endl;  
        }  
    }  
};
```

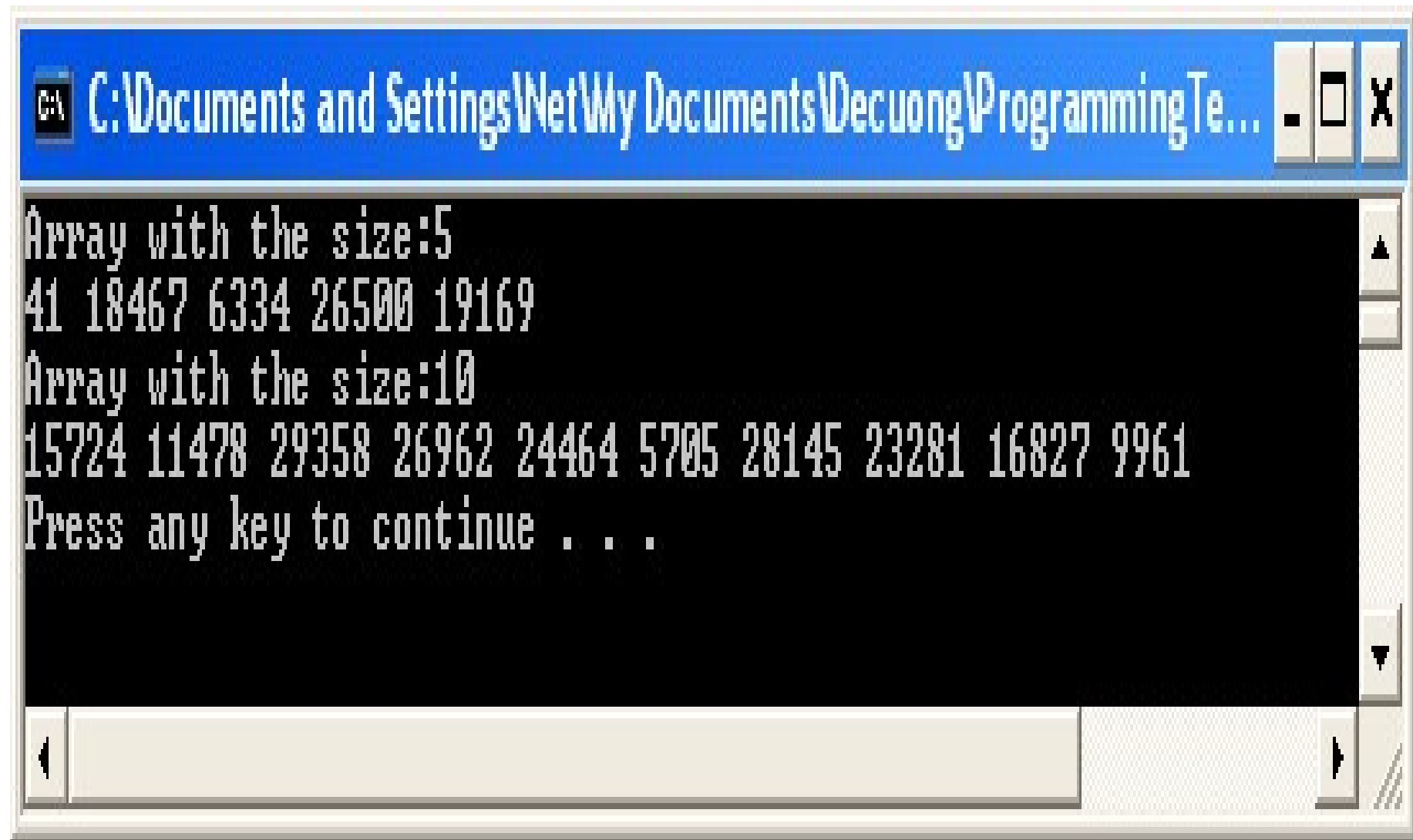
Program 9.6 (tiếp và hết)

```
int main()
{
    DynamicArray a(5),b;
    a.Init();
    a.Show();

    a.ChangeSize(10);
    a.Init();
    a.Show();

    return system("PAUSE"), EXIT_SUCCESS;
}
```

Kết quả chạy



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\Documents and Settings\Net\My Documents\Decuong\ProgrammingTe...". The command prompt area is black with white text. It displays the following output:

```
Array with the size:5  
41 18467 6334 26500 19169  
Array with the size:10  
15724 11478 29358 26962 24464 5705 28145 23281 16827 9961  
Press any key to continue . . .
```


Các thành phần kiểu *static*

- ◆ Ý nghĩa: các thành phần dữ liệu khai báo thông thường trong một lớp, thực ra là các thành phần dữ liệu riêng cho từng đối tượng của lớp đó. Nhưng đôi khi, ta lại muốn có các thành phần dữ liệu của chính lớp đó, tức là thành phần dữ liệu chung cho tất cả các đối tượng thuộc lớp đó. C++ cung cấp kiểu dữ liệu kiểu *static* cho phép chúng ta làm việc này.

Program 9.7: minh họa kiểu *static*: xây dựng một lớp sản sinh ra các số nguyên ngẫu nhiên

```
#include <cstdlib>
#include <iostream>
#include <time.h>

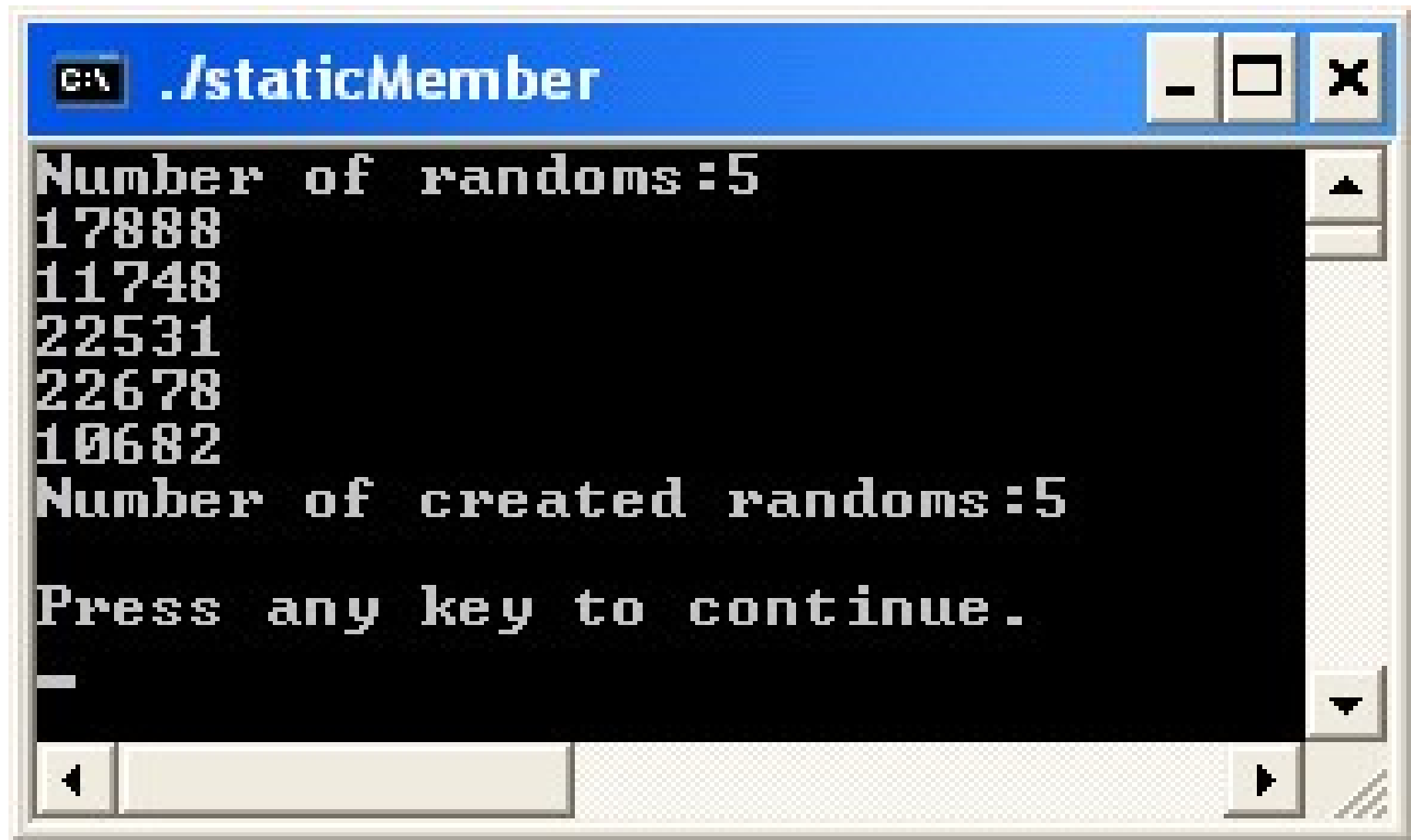
using namespace std;
```

```
class IntFactory{
    static unsigned count;
public:
    static void Init(){//Hàm khởi tạo
        srand((unsigned)time(0));
        count=0;
    }
    static int CreateRandom(){//Khởi tạo 1 số ngẫu
nhiên
        count++;
        return rand();
    }
    //Hàm trả về số lượng số nguyên đã được tạo ra
    static unsigned getCount(){
        return count;
    }
};
```

Program 9.7(next)

```
unsigned IntFactory::count;
int main()
{
    IntFactory fac;
    IntFactory::Init();
    int n;
    cout<<"Number of randoms:";
    cin>>n;
    for (int i=0;i<n;i++)
        cout<<fac.CreateRandom()<<" "<<endl;
    cout<<"Number of created
    randoms:"<<IntFactory::getCount()<<endl;
    return EXIT_SUCCESS;
}
```

Kết quả chạy



```
C:\ ./staticMember

Number of randoms:5
17888
11748
22531
22678
10682
Number of created randoms:5
Press any key to continue.
_
```

Kết luận

- ◆ Cách định nghĩa lớp và sử dụng các đối tượng của lớp đó
- ◆ Cách định nghĩa và sử dụng hàm thành viên
- ◆ Ý nghĩa và cách sử dụng hàm bạn
- ◆ Ý nghĩa con trỏ *this* và cách sử dụng
- ◆ Cách định nghĩa lại các toán tử trong lớp
- ◆ Ý nghĩa và cách sử dụng thành phần *static*

Bài tập

- ◆ Bài 1: Mở rộng lớp **Phân Số** bằng cách định nghĩa thêm 1 hàm cho phép tối giản phân số và định nghĩa lại các phép toán sau trên lớp đó:
 - $/$: thực hiện phép chia hai phân số
 - $!$: phép toán 1 ngôi cho phép đảo ngược 1 phân số
- ◆ Bài 2: Xây dựng lớp **Số Phức** với các yêu cầu sau:
 - Có một hàm tạo với các tham số nhận giá trị mặc định để tạo ra một số phức mặc định là $1+j$
 - Định nghĩa lại các phép toán $+$, $-$, $*$, $/$ để thực hiện các phép toán trên kiểu số phức
 - Định nghĩa lại phép toán $<<$ cho phép in ra một số phức có dạng “ $a + jb$ ” (với a là phần thực và b là phần ảo)

Bài tập (tiếp theo)

- ◆ Bài 3: Xây dựng một lớp mảng động một chiều, với các yêu cầu sau:
 - Có một hàm tạo cho phép tạo ra một mảng có kích thước có thể cho trước (giá trị mặc định là 0)
 - Định nghĩa lại phép toán ++ và --, với ý nghĩa là tăng lên và giảm đi kích thước mảng 1 phần tử
 - Định nghĩa lại phép toán += n và -= n với ý nghĩa tăng lên và giảm đi kích thước mảng n phần tử

Bài tập (tiếp theo và hết)

- ◆ Bài 4: Xây dựng một lớp String chứa một chuỗi các kí tự, có kích thước động theo các yêu cầu sau:
 - Có một hàm tạo String() cho phép tạo một chuỗi rỗng
 - Có một hàm tạo String(char ch) cho phép tạo một chuỗi có độ dài 1 kí tự bằng ch.
 - Có một hàm tạo String(char * str) cho phép tạo một chuỗi có kích thước và nội dung bằng với chuỗi str
 - Định nghĩa lại các phép toán + (để ghép 2 chuỗi) và gán = (để gán chuỗi cho chuỗi), và ! (để trả về chuỗi đảo ngược)
 - Có một hàm getLen() trả về kích thước chuỗi