Môn PPLTHĐT

Hướng dẫn thực hành tuần 5

Mục đích

Tìm hiểu về thành phần tĩnh của lớp đối tượng, giới thiệu thư viện STL với các lớp string và vector, làm bài tập áp dụng.

Nội dung

- Thành phần tĩnh.
- Thư viện STL - string.
- Thư viện STL – vector.
- Bài tập.

Yêu cầu

Nắm vững những nội dung được trình bày trong các bài hướng dẫn thực hành từ tuần 1 đến tuần 3.

1. Thành phần tĩnh

Các thành phần tĩnh *(static members)* của một lớp đối tượng bao gồm các thuộc tính tĩnh *(static attributes)* và các phương thức tĩnh *(static methods)*. Thành phần tĩnh được khai báo bắt đầu bằng từ khóa "static".

Thuộc tính tĩnh

Các thuộc tính tĩnh được xem như là các thuộc tính của lớp *(class variables)* bởi vì chúng chỉ có một giá trị duy nhất cho tất cả các đối tượng thuộc lớp đó. Giá trị của thuộc tính tĩnh là như nhau ở tất cả các đối tượng.

Ví dụ chúng ta có thể dùng thuộc tính tĩnh để đếm số lượng đối tượng được tạo ra của một lớp.

```
#include <iostream>

using namespace std;

class PhanSo
{
private:
	int	m_iTuSo;
	int	m_iMauSo;

public:
	static int m_iNumberOfInstances;	// Thuoc tinh ti~nh luu so luong doi tuong.

	PhanSo()
	{
		m_iNumberOfInstances++;
	}

	virtual ~PhanSo()
	{
		m_iNumberOfInstances--;
	}
};

int PhanSo::m_iNumberOfInstances = 0;		// Khoi tao gia tri ban dau.

void main()
{
	PhanSo		a;				// m_iNumberOfInstances = 1.
	PhanSo		b[5];				// m_iNumberOfInstances = 6.
	PhanSo		*c;
```

```
        c = new PhanSo;                                 // m_iNumberOfInstances = 7.

        cout << a.m_iNumberOfInstances << endl;
        delete c;                                       // m_iNumberOfInstances = 6.
        cout << PhanSo::m_iNumberOfInstances << endl;
}
```

Trong đoạn chương trình trên, m_iNumberOfInstances là thuộc tính tĩnh của lớp PhanSo. Giá trị của nó là như nhau ở tất cả các đối tượng của lớp PhanSo. Còn giá trị của m_iTuSo và m_iMauSo là khác nhau giữa các đối tượng PhanSo. Trong hàm dựng của lớp PhanSo, chúng ta tăng biến đếm m_iNumberOfInstances lên 1. Trong hàm hủy, chúng ta giảm biến đếm đi 1.

Chúng ta có 2 cách để truy xuất vào thành phần tĩnh của một lớp đối tượng:

- Sử dụng tên lớp: PhanSo::m_iNumberOfInstances.
- Sử dụng tên đối tượng: a.m_iNumberOfInstances.

Các thuộc tính tĩnh có tính chất như các biến toàn cục ngoại trừ việc nó được khai báo trong một lớp đối tượng nào đó.

Phương thức tĩnh

Các phương thức tĩnh được xem như các phương thức của lớp *(class methods)*. Chúng ta dùng phương thức tĩnh để viết lại ví dụ đếm số lượng đối tượng được tạo ra của một lớp.

```
#include <iostream>

using namespace std;

class PhanSo
{
private:
        int     m_iTuSo;
        int     m_iMauSo;

        static int m_iNumberOfInstances;    // Thuoc tinh ti~nh luu so luong doi tuong.

public:
        // Phuong thuc tinh tra ve so luong doi tuong.
        static int GetNumberOfInstances()
        {
                return m_iNumberOfInstances;
        }

        PhanSo()
        {
```

```
            m_iNumberOfInstances++;
        }

        virtual ~PhanSo()
        {
            m_iNumberOfInstances--;
        }
};

int PhanSo::m_iNumberOfInstances = 0;          // Khoi tao gia tri ban dau.

void main()
{
        PhanSo      a;                          // m_iNumberOfInstances = 1.
        PhanSo      b[5];                       // m_iNumberOfInstances = 6.
        PhanSo      *c;

        c = new PhanSo;                         // m_iNumberOfInstances = 7.

        cout << a.GetNumberOfInstances() << endl;
        delete c;                               // m_iNumberOfInstances = 6.
        cout << PhanSo::GetNumberOfInstances() << endl;
}
```

Trong phương thức tĩnh, chúng ta chỉ có thể truy xuất vào các thuộc tính tĩnh của lớp chứ không được phép truy xuất vào các thuộc tính của đối tượng hoặc sử dụng con trỏ this.

2. Thư viện STL - string

Giới thiệu

Thư viện chuẩn **STL** (**S**tandard **T**emplate **L**ibrary) của C++ có hỗ trợ kiểu **string** cùng với các phép toán và phương thức khá tiện lợi cho người lập trình. Hiện tại thì Visual C++ và hầu hết các trình biên dịch C++ trên Linux/Unix đều có sẵn thư viện STL nên ta có thể dùng kiểu **string**. Riêng các phiên bản hiện nay của Borland C++ thì dùng thư viện template riêng mà không bao gồm thư viện STL.

Chương trình sau đây cho thấy việc sử dụng kiểu string nhờ vào dùng thư viện STL khá đơn giản và tiện lợi.

```
#include <iostream.h>
#include <string>

using namespace std;

void Sort(int n, string str[]);

void main()
{
	string	str1("012");
	string	str2("345");
	string	s = str1 + str2;

	cout << "Result :" << s << endl;

	string	country[] = {"Viet nam", "Lao", "Campuchia", "Thai lan", "Trung quoc"};
	int	n = 5;

	Sort(n, country);
	cout << "After sorting" << endl;
	for (int i = 0;i < n; i++)
		cout << "\t"<< country[i] << endl;
}

void Sort(int n, string str[])
{
	for (int i = 0; i < n - 1; i++)
		for (int j = i + 1;j < n;j++)
			if (str[i] > str[j])
			{
				string	s = str[i];
				str[i] = str[j];
				str[j] = s;
			}
```

}

Trong chương trình trên, chúng ta cần chú ý các điểm sau:

- Các chỉ thị **include** và **using namespace** cần thiết để khai báo sử dụng kiểu **string**.
- Có thể khởi động và sao chép giá trị nhờ phép gán =, có thể dùng phép + để ghép chuỗi.
- Trong hàm **strSort()** ta có thể dùng phép so sánh > để so sánh 2 biến chuỗi.
- Có thể dùng toán tử << với **cout** để xuất một chuỗi ra màn hình hoặc dùng toán tử >> với **cin** để nhập một chuỗi ký tự đến khi gặp một khoảng trống thì dừng.

Các phương thức/phép toán tiện ích của kiểu string

Các phép toán và phương thức cơ bản

- Phép cộng + dùng để ghép hai chuỗi và cũng để ghép một ký tự vào chuỗi.
- Các phép so sánh theo thứ tự từ điển : == , !=,>,>=,<.<=.
- Phương thức **length()** và phép lấy chỉ số, toán tử **[ ]** để duyệt từng ký tự của chuỗi: nếu s là biến kiểu **string** thì s[i] là ký tự thứ i của s với 0 <= i < s.length().
- Phép gán = dùng để gán biến kiểu string bằng một hằng chuỗi.

```
{
        char   st[]="ABCDEF";
        string  s = "XYZ";

        cout << s << endl;
        s = st;
        cout << s.length() << ":" << s << endl;
}
```

Các phương thức chèn, xoá, lấy chuỗi con

- Lấy chuỗi con: **substr(int pos, int nchar)**
  - Ví dụ: **str.substr(2,4)** trả về chuỗi con gồm 4 ký tự của str kể từ vị trí thứ 2.
- Chèn thêm ký tự hay chuỗi vào một vị trí nào đó của chuỗi str. Có nhiều cách dùng:
  - **str.insert(int pos, char* s)**: chèn s (kết thúc bằng '\0') và vị trí pos của str.
  - **str.insert(int pos, string s)**: chèn chuỗii s vào vị trí pos của chuỗi str.
  - **str.insert(int pos, int n, int ch)**: chèn n lần ký tự ch vào vị trí pos của str.
- Xoá **erase(int pos, int n)**: xoá n ký tự của chuỗi str kể từ vị trí pos, nếu không quy định giá trị n thì tất cả các ký tự của str từ vị trí pos trở đi đều bọ xoá.

Các phương thức tìm kiếm và thay thế:

- Phương thức **find()** tìm kiếm xem một ký tự hay một chuỗi nào đó có xuất hiện trong một chuỗi str cho trước hay không. Có nhiều cách dùng phương thức này:
  - **str.find(int ch, int pos = 0):** tìm ký tự ch kể từ vị trí pos đến cuối chuỗi str.
  - **str.find(char* s, int pos = 0)**: tìm chuỗi con s kể từ vị trí pos đến cuối chuỗi str.
  - **str.find(string s,int pos=0)**: tìm chuỗi s kể từ vị trí pos đến cuối chuỗi str.
  - Nếu không qui định giá trị pos thì hiểu mặc nhiên là 0, nếu tìm có kết quả thì phương thức trả về vị trí xuất hiện đầu tiên, ngược lại trả về giá trị -1.
- Phương thức **replace()** thay thế một đoạn con trong chuỗi str cho trước (đoạn con kể từ một vị trí pos và đếm tới nchar ký tự về phía cuối chuỗi) bởi một chuỗi s nào đó hoặc bởi n ký tự ch nào đó. Có nhiều cách dùng, thứ tự tham số như sau:
  - **str.replace(int pos, int nchar, char *s)**
  - **str.replace(int pos, int nchar, string s)**
  - **str.replace(int pos, int nchar, int n, int ch)**

Một số phương thức khác

Còn nhiều phương thức tiện ích khác như: **append(), rfind(), find_first_not_of(), find_last_not_of(), swap(), c_str()**. Cách dùng các hàm này đều được trình bày trong hệ thống hướng dẫn (help) của các môi trường có hỗ trợ STL (trong VC++ là MSDN). Ngoài ra các phương thức như **find_first_of()** tương tự như **find()**, **find_last_of()** tương tự như **rfind()**.

3. Thư viện STL – vector

Lớp mảng động **vector<T>** có sẵn trong thư viện chuẩn STL của C++ cho phép định nghĩa một mảng động các phần tử kiểu T, có thể xác định kích thước mảng vào lúc đang chạy chương trình, có thể thay đổi kích thước mảng khi cần thiết.

vector<T>::size()	Trả về kích thước hiện hành của mảng.
vector<T>::resize(int iNewSize)	Thay đổi kích thước mảng.
T & vector<T>::operator [](int iIndex)	Tham chiếu đến phần tử ở vị trí iIndex trong mảng (0 <= iIndex <= this.size()).

Chẳng hạn đoạn chương trình sau khai báo và nhập vào n = 10 số thực:

```
#include <iostream>
#include <vector>

using namespace std;

void main()
{
	int		n = 10;
	vector<float>	a(n);

	for (int i = 0; i < a.size(); i++)
	{
		cout << "a[" << i << "] = ";
		cin >> a[i];
	}

	cout << "After entering data…" << endl;

	for (i = 0; i < a.size(); i++)
		cout << a[i] << endl;
}
```

Trong đoạn chương trình trên, kích thước của mảng a được xác định bởi biến n.

Trường hợp xác định kích thước mảng khi chương trình đang chạy, chúng ta dùng hàm dựng mặc định *(default constructor)* để khai báo mảng chưa xác định kích thước, sau đó dùng phương thức **resize()** để xác định kích thước của mảng khi cần. Chương trình sau đây nhập vào n từ (word) mỗi từ là một chuỗi kiểu string:

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;
```

```
void main()
{
	int 		iWordNum;
	vector<string> arrWords;

	cout << "Enter number of words = ";
	cin >> iWordNum;

	if (iWordNum <= 0)
	{
		cout << "Fail!";
		return;
	}

	arrWords.resize(iWordNum);

	for (int i = 0; i < arrWords.size(); i++)
	{
		cout << "Enter word " << i << " = ";
		cin >> arrWords[i];
	}

	cout << "After entering data..." << endl;

	for (i = 0; i < arrWords.size(); i++)
		cout << arrWords[i] << endl;
}
```