Môn PPLTHĐT

Hướng dẫn thực hành tuần 9

Mục tiêu

Tìm hiểu về function template, class template. Bài tập áp dụng đa hình trong lập trình hướng đối tượng và giới thiệu một số vấn đề mở rộng.

Nội dung

- Function template
- Class template
- Bài tập áp dụng đa hình
- Kỹ thuật tạo mẫu đối tượng
- Bài toán mạch điện

Yêu cầu

Nắm vững lý thuyết về kế thừa, đa hình.

1. Function Template

Khái niệm

Xét ví dụ hàm tìm Min giữa 2 số cho nhiều kiểu dữ liệu khác nhau:

```
// min for ints.
int min(int a, int b)
{
   return (a < b) ? a : b;
}

// min for floats.
float min(float a, float b)
{
   return (a < b) ? a : b;
}

// min for chars.
char min(char a, char b)
{
   return (a < b) ? a : b;
}
```

Với mỗi kiểu dữ liệu khác nhau (int, float, char…) chúng ta phải viết hàm tìm Min tương ứng với kiểu dữ liệu đó (overload). Điều này gây ra sự dư thừa không đáng có trong chương trình. Và hơn nữa các hàm trên vẫn không đủ dùng trong mọi trường hợp.

=> Dùng Function Template.

Function Template là hàm tổng quát cho phép dùng nhiều kiểu dữ liệu khác nhau cho tham số và kết quả trả về của nó. Chúng ta không phải viết nhiều hàm cho từng kiểu dữ liệu cụ thể.

template < template-argument-list > declaration

Tất cả các function template được định nghĩa bắt đầu với từ khóa **template** theo sau là một danh sách các tham số hình thức vây quanh trong cặp dấu (< và >). Mỗi tham số hình thức được đặt trước bởi từ khóa **class** và được phân cách bởi dấu phẩy:

template <**class** *T*>

hoặc

template <**class** *T1*, **class** *T2*,…>

Lúc này các hàm tìm Min có thể được thay thế bởi một hàm Function Template duy nhất:

```
template <class T>
T min(T a, T b)
```

```
{
  return (a < b) ? a : b;
}
```

Nhận xét

- Dùng function template chúng ta chỉ cần viết một hàm duy nhất cho nhiều kiểu dữ liệu khác nhau thay vì phải viết nhiều hàm cho từng kiểu dữ liệu cụ thể.
- Dùng function template giúp giảm được kích thước và tăng tính linh động của chương trình.

2. Class Template

Khái niệm

Xét ví dụ xây dựng lớp mảng cho nhiều kiểu dữ liệu khác nhau.

```
// Mang so nguyen.
class IntegerArray
{
private:
	int	*m_pElement;
	// Viet cai dat cho lop.
};

// Mang so thuc.
class FloatArray
{
private:
	float	*m_pElement;
	// Viet cai dat cho lop.
};

// Mang ky tu.
class CharArray
{
private:
	char	*m_pElement;
	// Viet cai dat cho lop.
};
```

Với mỗi kiểu dữ liệu khác nhau (int, float, char…) chúng ta phải xây dựng lớp mảng tương ứng cho kiểu dữ liệu đó. Điều này gây ra sự dư thừa không đáng có trong chương trình. Và hơn nữa các lớp trên vẫn không đủ dùng trong mọi trường hợp.

=> Dùng Class Template.

Class Template là lớp đối tượng tổng quát cho phép dùng nhiều kiểu dữ liệu khác nhau cho các thuộc tính và phương thức của lớp. Tương tự như Function Template, Class Template được khai báo bắt đầu bằng từ khóa “template”.

```
template <class T>
class SampleClass
{
	// Viet cai dat.
};
```

Ví dụ

Để hiểu rõ hơn về Class Template, chúng ta xét ví dụ xây dựng lớp mảng cho nhiều kiểu dữ liệu khác nhau Array.

Bước 1: vào VS, tạo project dạng Console Application (Visual C++).

Bước 2: thêm vào project file Array.h, viết khai báo và cài đặt cho lớp Array như sau:

```
template <class T>
class Array
{
private:
	T	*m_pElement;
	int	m_iLength;

public:
	Array(int iLength)
	{
		if (iLength < 0)
		{
			cout << "Loi: chieu dai mang la so am.";
			return;
		}

		m_iLength = iLength;
		m_pElement = new T[m_iLength];
	}

	Array(const Array &obj)
	{
		m_iLength = obj.m_iLength;
		m_pElement = new T[m_iLength];

		// Sao chep vung nho da cap phat cho m_pElement cua obj.
		for (int i = 0; i < m_iLength; i++)
			m_pElement[i] = obj.m_pElement[i];
	}

	int GetLength()
	{
		return m_iLength;
	}

	T & ElementAt(int iIndex)
	{
		if (iIndex < 0 || iIndex >= m_iLength)
			cout << "Loi: truy xuat phan tu ngoai pham vi mang.";
```

```
		return m_pElement[iIndex];
	}

	T & operator [](int iIndex)
	{
		return ElementAt(iIndex);
	}

	Array & operator =(const Array &obj)
	{
		m_iLength = obj.m_iLength;
		m_pElement = new T[m_iLength];

		// Sao chep vung nho da cap phat cho m_pElement cua obj.
		for (int i = 0; i < m_iLength; i++)
			m_pElement[i] = obj.m_pElement[i];

		return *this;
	}

	virtual ~Array()
	{
		if (m_pElement != NULL)
			delete []m_pElement;
	}
};
```

Bước 4: thêm vào project file main.cpp và viết đoạn chương trình sử dụng lớp Array vừa tao như sau:

```
#include "Array.h"

void main()
{
	Array<int>	a(3);

	a[0] = 0;
	a[1] = 1;
	a[2] = 2;

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

Bước 5: biên dịch và chạy thử chương trình.

3. Bài toán dẫn nhập

Công ty ABC cần xây dựng một ứng dụng quản lý nhân sự và tính lương cho nhân viên trong công ty như sau:

1. Quản lý thông tin về nhân viên: (Họ tên, ngày sinh, địa chỉ)
2. Tính lương cho nhân viên

Hiện công ty có 3 loại nhân viên và cách tính lương tương ứng cho từng loại nhân viên như sau:

a. Nhân viên sản xuất: số sản phẩm x 20000 đ

b. Nhân viên công nhật: số ngày công x 50.000 đ

c. Nhân viên quản lý: lương cơ bản x hệ số lương

Hãy viết chương trình quản lý và tính tổng lương các nhân viên của công ty ABC. Trước tiên ta phác thảo sơ bộ các lớp cần xây dựng như sau:

NVSanXuat
HoTen NgaySinh DiaChi **SoSanPham**
Nhap **Xuat** **TinhLuong**

NVCongNhat
HoTen NgaySinh DiaChi **SoNgayCong**
Nhap **Xuat** **TinhLuong**

NVQuanLy
HoTen NgaySinh DiaChi **LuongCoBan** **HeSoLuong**
Nhap **Xuat** **TinhLuong**

Ba lớp đối tượng trên có những thuộc tính và phương thức giống nhau. Xây dựng lớp cha có tên là **NhanVien** chứa những phần giống nhau của 3 lớp trên. Ba lớp trên sẽ kế thừa từ lớp NhanVien.

Tuy nhiên do các công việc Nhập, Xuất, và Tính Lương là khác nhau cho từng loại nhân viên. Đến đây có 2 hướng giải quyết:

3.1 Không sử dụng cơ chế đa hình

Ở mỗi lớp con, định nghĩa lại (nạp chồng hàm) 3 phương thức trên.

```
class NhanVien
{
private:
	string m_strHoTen;
	string m_strNgaySinh;
	string m_strDiaChi;
public:
	void Nhap();
	void Xuat();
	float TinhLuong();
};

class NVSanXuat : public NhanVien
{
private:
	int m_iSoSanPham;
public:
	void Nhap();
	void Xuat();
```

```
        float TinhLuong();
};

class NVCongNhat : public NhanVien
{
private:
        int m_iSoNgayCong;
public:
        void Nhap();
        void Xuat();
        float TinhLuong();
};

class NVQuanLy : public NhanVien
{
private:
        float m_fHeSoLuong;
        float m_fLuongCoBan;
public:
        void Nhap();
        void Xuat();
        float TinhLuong();
};
```

Cài đặt theo cách này, chúng ta cần quản lý 3 danh sách

```
vector <NVSanXuat> arrNVSX;
vector <NVCongNhat> arrNVCN;
vector  <NVQuanLy> arrNVQL;
```

Trong thao tác nhập liệu cho toàn bộ nhân viên của công ty, tùy vào loại nhân viên sẽ gọi phương thức *Nhập* tương ứng:

```
cout << "Nhap loai nhan vien (1:NVSX, 2:NVCN, 3:NVQL)";
cin >> iLoai;
```

```
NVSanXuat nvSX;
NVCongNhat nvCN;
NVQuanLy nvQL;

switch (iLoai)
{
case 1: nvSX.Nhap();
      arrNVSX.push_back(nvSX);
      break;
case 2:
      nvCN.Nhap();
      arrNVCN.push_back(nvCN);
      break;
case 3: nvQL.Nhap();
      arrNVQL.push_back(nvQL);
      break;
}
```

Muốn tính tổng lương của cả danh sách, phải lần lượt tính tổng lương của mỗi danh sách.

```
float fTongLuong = 0;
for (int i = 0; i < arrNVSX.size(); i++)
      fTongLuong += arrNVSX[i].TinhLuong();

for (int j = 0; j < arrNVCN.size(); j++)
      fTongLuong += arrNVSX[j].TinhLuong();

for (int k = 0; k < arrNVQL.size(); k++)
      fTongLuong += arrNVQL[k].TinhLuong();
```

3.2 Áp dụng cơ chế đa hình

```
class NhanVien
{
```

```
private:
	string m_strHoTen;
	string m_strNgaySinh;
	string m_strDiaChi;
public:
	virtual void Nhap() = 0;
	virtual void Xuat() = 0;
	virtual float TinhLuong() = 0;
};

class NVSanXuat : public NhanVien
{
private:
	int SoSanPham;
public:
	void Nhap();
	void Xuat();
	float TinhLuong();
};

class NVCongNhat : public NhanVien
{
private:
	int SoNgayCong;
public:
	void Nhap();
	void Xuat();
	float TinhLuong();
};

class NVQuanLy : public NhanVien
{
private:
	float HeSoLuong;
	float LuongCoBan;
```

```
public:
      void Nhap();
      void Xuat();
      long TinhLuong();
};
```

Lúc này chỉ cần một danh sách nhân viên duy nhất để quản lý toàn bộ nhân viên của công ty:

```
vector<CNhanVien*> arrNhanVien;
// Nhập 1 nhân viên mới
cout << ”Nhap loai nhan vien (1:NVSX, 2:NVCN, 3:NVQL)”;
cin >> iLoai;

switch(iLoai)
{
      case 1: arrNhanVien.push_back(new NVSanXuat);
              break;
      case 2: arrNhanVien.push_back(new NVCongNhat);
              break;
      case 3: arrNhanVien.push_back(new NVQuanLy);
              break;
}

arrNhanVien[arrNhanVien.size() - 1].Nhap();
//Tính tổng lương nhân viên
float fTongLuong = 0;
for (int i = 0; i < arrNhanVien.size(); i++)
      fTongLuong += arrNhanVien[i].TinhLuong();
```

Cơ chế đa xạ cho phép gọi đúng những phương thức của đối tượng mà không cần biết trước kiểu dữ liệu của đối tượng đó nhờ vào một con trỏ ảo (virtual pointer) bên trong đối tượng. Con trỏ này giữ địa chỉ của một bảng ảo (virtual table) trong đó chứa thông tin về kiểu dữ liệu và địa chỉ các phương thức tương ứng của đối tượng. Đây được xem là điểm then chốt của cơ chế đa xạ.

Có thể khai báo lớp DanhSachNhanVien để xử lý các nghiệp vụ công ty như sau.

```
class DanhSachNhanVien
{
private:
	vector<NhanVien*> arrNhanVien;
public:
	void NhapDanhSach();
	void XuatDanhSach();
	void XuatBangLuong();
	CNhanVien* TimNhanVienLuongCaoNhat();
	CNhanVien* TimNhanVienLuongThapNhat();
	float TongLuong();
	……..
};
```

4. Áp dụng kỹ thuật tạo đối tượng mẫu

Trong cả 2 giải pháp trên, trong thao tác nhập danh sách nhân viên đều dùng đến bảng chọn (*1:NVSX, 2:NVCN, 3:NVQL*) và các câu lệnh switch. Bảng chọn trên là khá cứng nhắc, Giả sử có nhu cầu thay đổi thứ tự trên hoặc thêm một loại nhân viên mới, đều phải thay đổi bảng chọn này.

Để giải quyết vấn đề trên, có thể áp dụng kỹ thuật tạo mẫu đối tượng.

```
class NhanVien
{
……
public:
		virtual NhanVien*  TaoDoiTuong() = 0;
		……..
};

class NVSanXuat()
{
……
```

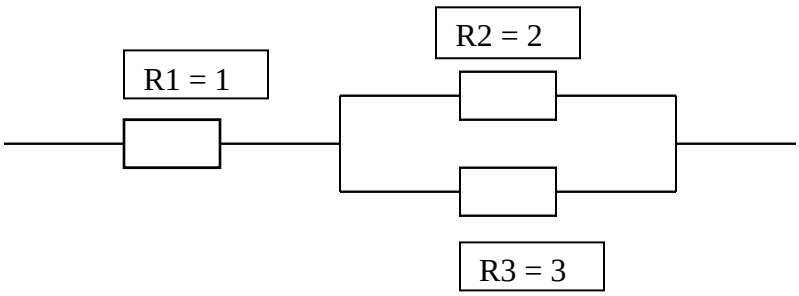
```
public:
      NhanVien* TaoDoiTuong()
      {
            NhanVien* nv = new NVSanXuat();
            return nv;
      }

      string TenLop()
      {
            return "NVSanXuat";
      }
      ……..
};

vector <NhanVien*> arrDoiTuongMau;
arrDoiTuongMau.push_back(new NVSanXuat);
arrDoiTuongMau.push_back(new NVCongNhat);
arrDoiTuongMau.push_back(new NVQuanLy);

// Nhập 1 nhân viên mới
for (int i = 0; i < arrDoiTuongMau.size(); i++)
      cout << i + 1 << ":"
            << arrDoiTuongMau[i]->TenLop() << endl;
int iLoai;
cin >> iLoai;
arrNhanVien.push_back(arrDoiTuongMau[id-1]->TaoDoiTuong());
```

5. Bài toán mạch điện



Tính tổng trở của một mạch điện phức hợp gồm nhiều mạch song song, nối tiếp với nhau.

Gợi ý thiết kế lớp như sau

```
class Circuit
{
	public:
		virtual float Resistance() = 0;
};

class SimpleCircuit : public Circuit
{
private:
	float m_fResistance;
public:
	float Resistance()
	{
			return m_fResistance;
	}
};

class CompositeCircuit : public Circuit
{
 protected:
	vector <Circuit> arrCircuit;
  public:
	double Resistance()
	{
		return 0;
	}
};
```

```
class ParallelCircuit : public CompositeCircuit
{
public:
	double Resistance()
	{
		// tính điện trở cho mạch song song
	}
}

class SerialCircuit : public CompositeCircuit
{
 public:
	double Resistance()
	{
		// tính điện trở cho mạch nối tiếp
	}
};
```