# Nội dung tuần 10

- Xử lý lỗi ngoại lệ (Exception Hadling).
- Ôn tập

# Bài tập

Bài 4 được tính thành 10 bài theo các dòng lệnh ở hàm main có đánh số

❖ Yêu cầu

A: bài 1, bài 2, bài 3

H: Làm hết 4 bài (tính thành 13 bài).

## Bài 1

Cài đặt hoàn thiện lại ví dụ 1, 2, 3.

## Bài 2

Cài đặt lại ví dụ 4, 5.

## Bài 3

Cài đặt lại ví dụ 6.

## Bài 4

Mô tả **FILE SYSTEM** như sau

Mỗi **Drive** chứa lượng lớn các **file** và **folder**.

- **File** có **name**, **size**, thuộc tính **read-only** và thuộc tính **hidden**
- **Folder** chứa nhiều file và folder con. Folder cũng có **name**, thuộc tính **read-only** và thuộc tính **hidden. Size** của folder được tính bằng tổng tất cả các file và folder mà nó chứa.
- Thông tin khi **Print** là ***name***, ***size*** và các thuộc tính ***read-only***, ***hidden*** đối với cả file và folder.

Hãy xây dựng các class phù hợp để cho hàm **main** sau chạy đúng.

```
int main() {
	CFolder C("C");			// (1)
	CFolder System("System");
	CFolder Windows("Windows");
	CFile a_txt("a.txt", 123);	// (2)
	CFile b_doc("b.doc", 456);
	System.Add(&a_txt);		// (3)
	Windows.Add(&b_doc);
	C.Add(&System);			// (4)
```

```
	C.Add(&Windows);
	cout << "Content of folder C ->" << endl;
	bool isPrintHiddenItems = false;
	C.Print(isPrintHiddenItems); // print hidden items or not? 1/0 (5)
	CItem* p = C.RemoveByName("System");	// (6)
	cout << "Content of folder C afer removing folder System ->" << endl;
	C.Print(false);
	p = C.FindByName("b.doc");			// (7)
	if (p != NULL) {
		cout << "b.doc is found in folder C" << endl;
	}
	else {
		cout << "b.doc is not found" << endl;
	}
	p = C.FindByName("a.txt");
	if (p != NULL) {
		cout << "a.txt is found" << endl;
	}
	else {
		cout << "a.txt is not found" << endl;
	}
	p = C.FindByName("Windows");			// (8)
	bool isHidden;
	bool isAlsoApplyToChildren;
	if (p != NULL) {
		cout << "Folder Windows is folder. Content of folder Windows ->" <<
endl;
		isHidden = true; isAlsoApplyToChildren = false;
		// set HIDDEN to folder p and do not change hidden attributes of its
sub-items
		p->SetHidden(isHidden, isAlsoApplyToChildren);  // (9)
		p->Print(false);
		// set HIDDEN to folder p and all its items
		isHidden = true; isAlsoApplyToChildren = true;
		p->SetHidden(isHidden, isAlsoApplyToChildren);  // (10)
		p->Print(false);
	}
	else {
		cout << "Folder Windows is not found" << endl;
	}
	return 0;
}
```

# Hướng dẫn

## Ví dụ 1: Chương trình bị crash vì không được xử lý lỗi

```
class CA
{
private:
	double _value;
public:
	CA(const double& v) { _value = v; }
	/// hàm minh họa gây lỗi
```

```
	double GetTest()
	{
		return this->_value;
	}
};

class Level1
{
public:
	/// hàm minh họa chạy không kiểm soát lỗi có thể xảy ra
	static double RunTest1(CA* pCA)
	{
		double rs = pCA->GetTest();
		return sqrt(rs);
	}
};

int main()
{
	cout << Level1::RunTest1(nullptr) << endl;
	system("pause");
	return 0;
}
```

## Ví dụ 2: Xử lý lỗi thông qua thông báo lỗi bằng giá trị trả về

```
class CA
/// không đổi

class Level1
{
public:
	/// hàm minh họa thông báo lỗi theo cách sử dụng giá trị trả về
	static double RunTest2(CA* pCA)
	{
		if (pCA == nullptr)
		{
			return -1;
		}
		double rs = pCA->GetTest();
		return sqrt(rs);
	}
};

int main()
{
	cout << Level1::RunTest2(nullptr) << endl;
	system("pause");
	return 0;
}
```

- ✓ Chuong trình không còn bị crash
- ✓ Hạn chế dễ thấy là không gian giá trị trả về nếu phủ đầy thì khó triển khai.
- ✓ Việc chuyển lỗi lên tầng trên nếu có thêm nhiều tầng xử lý cũng phức tạp hơn như ví dụ 3 dưới đây

## Ví dụ 3: Chuyển lỗi lên cao hơn theo cách làm thông báo lỗi qua giá trị trả về

```
class CA
{
private:
	double _value;
public:
	CA(const double& v) { _value = v; }
	/// hàm minh họa gây lỗi
	double GetTest()
	{
		return this->_value;
	}
};
/// không đổi

class Level1
/// không đổi

class Level2
{
public:
	/// hàm minh họa xử lý lỗi dựa vào giá trị trả về
	static double RunTest1()
	{
		CA* pCA = nullptr;
		double dRS = Level1::RunTest2(pCA);
		if (dRS == -1)
		{
			return -1;
		}
		return sqrt(dRS);
	}
};

int main()
{
	cout << Level2::RunTest1() << endl;
	system("pause");
	return 0;
}
```

✓ Nếu ở Level 2 ko xử lý cũng như không chuyển lỗi lên thì như vậy lỗi đã không được xử lý triệt để.

## Ví dụ 4: Sử dụng exception

```
class CA
/// không đổi

class Level1
{
public:
	/// hàm minh họa thông báo lỗi theo cách quăng exception
	static double RunTest3(CA* pCA)
```

```
	{
		if (pCA == nullptr)
		{
			throw exception("Argument must not be null");
		}

		double rs = pCA->GetTest();
		return sqrt(rs);
	}
};
int main()
{
	cout << Level1::RunTest3(nullptr) << endl;
	system("pause");
	return 0;
}
```

- ✓ Chương trình vẫn bị crash mặc dù dường như đã có kiểm tra lỗi !?!
- ✓ Throw exception bản chất là quăng lỗi ra báo cho hệ thống có lỗi và từ đâu.

## Ví dụ 5: Exception handling

```
class CA
/// không đổi

class Level1
/// không đổi

class Level2
{
public:
	/// hàm minh họa bỏ qua xử lý lỗi, để tầng cao hơn xử lý
	static double RunTest2()
	{
		CA* pCA = nullptr;
		double dRS = Level1::RunTest3(pCA);
		return sqrt(dRS);
	}

};


int main()
{
	try
	{
		cout << Level2::RunTest2() << endl;
	}
	catch (const exception& ex)
	{
		cout << "error: " << ex.what() << endl;
	}

	system("pause");
	return 0;

}
```

- ✓ Khối lệnh đặt trong try {} nếu có quăng ra **exception** thì sẽ được bắt lại để xử lý trong khối lệnh catch {}. Nếu trong catch không còn quăng ra exception thì chương trình vẫn tiếp tục với các lệnh sau khối catch. Như vậy việc xử lý lỗi đã được thực hiện hoàn toàn.
- ✓ Lưu ý với cơ chế sẵn sàng đón bắt exception thì việc các khối lệnh trong try thực sự sẽ có giảm hiệu năng nên chỉ thực sự những lệnh nào có khả năng quăng ra exception mới đặt trong try.

## Ví dụ 6: Custom exception để thêm thông tin rõ ràng hơn cho error cũng như phân tách xử lý hơn

```
class CA
/// không đổi

/// tạo custom exception bằng cách dẫn xuất
class MyException : public exception
{
private:
	string _errorDescription;
	int _errorCode;
public:
	/// hàm khởi tạo có tạo message cho base
	MyException(const int& errCode, const string& errDes) :
exception(BuildErrorMessage(errCode, errDes))
	{
		_errorCode = errCode;
		_errorDescription = errDes;
	}
	const string& GetErrorDescription()
	{
		return _errorDescription;
	}
	const int& GetErrorCode()
	{
		return _errorCode;
	}
private:
	static const char* BuildErrorMessage(const int& errCode, const string&
errDes)
	{
		ostringstream oss;
		oss << "Error code: " << errCode;
		oss << " - Description: " << errDes;
		int len = strlen(oss.str().c_str());
		char* temp = new char[len + 1];
		strcpy_s(temp, len + 1, oss.str().c_str());
		return temp;
	}
};

class Level1
{
public:
	/// hàm minh họa thông báo lỗi theo cách quăng custom exception
	static double RunTest4(CA* pCA)
```

```
	{
		if (pCA == nullptr)
		{
			throw MyException(123, "Argument must not be null");
		}

		double rs = pCA->GetTest();
		return sqrt(rs);
	}
};

class Level2
{
public:
	/// hàm minh họa gọi hàm tạo ra custom exception
	static double RunTest3()
	{
		CA* pCA = nullptr;
		double dRS = Level1::RunTest4(pCA);
		return sqrt(dRS);
	}
	/// hàm minh họa quăng exception
	static void RunErr()
	{
		throw exception("Common error");
	}
};

int main()
{
	for (int i = 0; i < 2; i++)
	{
		cout << "****************************************" << endl;
		try
		{
			if (i == 0)
			{
				cout << Level2::RunTest3() << endl;
			}
			else
			{
				Level2::RunErr();
			}
		}
		catch (const MyException& me)
		{
			cout << "custom error: " << me.what() << endl;
		}
		catch (const exception& ex)
		{
			cout << "common error: " << ex.what() << endl;
		}
	}
	system("pause");
	return 0;
}
```