Nội dung tuần 04

Luyện tập cách xây dựng các hàm toán tử cho lớp đối tượng.

Bài tập

❖ Yêu cầu

A: 3 bài 1, 2, 3

H: Làm hết 5 bài

Lưu ý: mục tiêu quan trọng là các thành phần dữ liệu hợp lệ của các lớp đối tượng, gợi ý kết quả chỉ là một trong số các cách khởi tạo giá trị với tham số.

Khi khởi tạo với nhiều thành phần dữ liệu có mối liên quan với nhau thì nên thực hiện mối liên quan đó (ví dụ số ngày dư thì đổi sang tháng, tháng dư thì đổi sang năm,...)

Bài 1

Cài đặt lại ví dụ 1.

Bài 2

Cài đặt lại ví dụ 2.

Bài 3

Cài đặt lại ví dụ 3.

Bài 4

Khai báo và cài đặt lớp đối tượng **LinkedList** để chạy đúng với hàm main sau:

```
int main()
{
	srand((unsigned int)time(NULL));
	const int maxV = 1000, iTest = 2;
	int n = 5 + rand() % 10;
	LinkedList<int> ll;
	for (int i = 0; i < n; i++)
	{
		if (rand() % 2 == 0)
		{
			ll.AddHead(rand() % maxV);
		}
		else
		{
			ll.AddTail(rand() % maxV + maxV);
		}
	}
	cout << ll << endl;
	ll[iTest] = -123;
	ll[iTest + 2] = 9876;
```

```
        ll.RemoveHead();
        ll.RemoveTail();
        cout << "Sau thay doi:" << endl;
        cout << ll << endl;

        LinkedList<CString> llCS;
        llCS.AddTail(CString("cstring-1"));
        llCS.AddTail(CString("cstring-2"));
        llCS.AddTail(CString("cstring-3"));
        cout << "List CString: " << llCS << endl;
        llCS[iTest] = CString("new CString");
        cout << "List CString changed: " << endl << llCS << endl;
        system("pause");
        return 0;
}
```

```
408, 1770, 1672, 1686, 1029, 1876,
Sau thay doi:
-123, 1686, 9876, 1876,
List CString: cstring-1, cstring-2, cstring-3,
List CString changed:
cstring-1, cstring-2, new CString,
Press any key to continue . . .
```

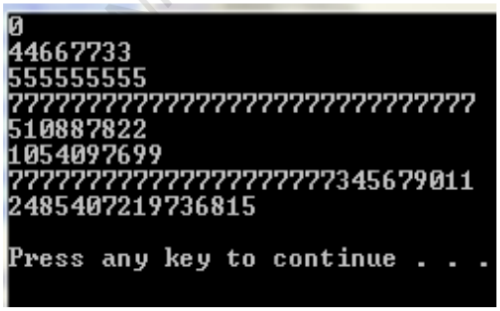
Bài 5

Khai báo và cài đặt lớp đối tượng **SoNguyenLon** với số lượng chữ số không giới hạn (sử dụng **LinkedList**) sao cho hàm main sau chạy đúng

```
int main()
{
        SoNguyenLon snl1;
        SoNguyenLon snl2(44667733);
        SoNguyenLon snl3(5, 9);
        SoNguyenLon snl4(7, 30);
        SoNguyenLon snl5 = snl3 - snl2;
        SoNguyenLon snl6 = 1098765432 - snl2;
        SoNguyenLon snl7 = snl4 - snl3 + 123456789;
        SoNguyenLon snl8 = snl2 * snl3;

        cout << snl1 << endl << snl2 << endl << snl3 << endl;
        cout << snl4 << endl << snl5 << endl << snl6 << endl;
        cout << snl7 << endl << snl8 << endl << endl;
        system("pause");
        return 0;
}
```

Với kết quả như sau:



Biết rằng **SoNguyenLon** không có giá trị âm (số bé hơn trừ số lớn hơn sẽ có kết quả là 0).

Hướng dẫn

Ví dụ 1 (sử dụng tham chiếu khi trả về giá trị trong operator)

```
class CString
{
private:
	char* _mang;
public:
	CString();
	CString(const char*);
	~CString();

	char operator[](const int&);
	friend ostream& operator<<(ostream&, const CString&);
};

int main()
{
	const int iN = 2;
	CString cs1("cs1_content");
	cout << cs1 << endl;
	cout << "cs1[" << iN << "] = " << cs1[iN] << endl;
	cs1[iN] = 'N';		// error
	system("pause");
	return 0;
}
```

- ✓ Chương trình sẽ build lỗi khi cố gắng gán giá trị mà sử dụng operator[] trả về tham trị.
- ✓ Sửa lại toán tử như sau và cài đặt thêm operator++ (chú ý về postfix và prefix)

```
class CString
{
private:
	char* _mang;
public:
	CString();
	CString(const char*);
	~CString();
	char& operator[](const int&);
	friend ostream& operator<<(ostream&, const CString&);
};
```

```
CString::~CString()
{
	if (_mang != nullptr)
	{
		delete[] _mang;
	}
}

CString::CString()
{
	_mang = nullptr;
}

CString::CString(const char *str)
{
	int len = strlen(str);
	_mang = new char[len + 1];
	strcpy_s(_mang, len + 1, str);
}

char& CString::operator[](const int& i)
{
	int len = strlen(_mang);
	if (i < 0 || i >= len)
	{
		throw exception("out of range");
	}
	return _mang[i];
}

ostream& operator<<(ostream &os, const CString &cs)
{
	os << cs._mang;
	return os;
}

int main()
{
	const int iN = 2;
	CString cs1("cs1_content");
	cout << cs1 << endl;
	cout << "cs1[" << iN << "] = " << cs1[iN] << endl;
	cs1[iN] = 'N';
	cout << cs1 << endl;
	system("pause");
	return 0;
}
```

Ví dụ 2 (phân biệt Copy Constructor và Operator=)

```
class CString
{
private:
	char* _mang;
public:
	CString();
	CString(const char*);
	CString(const CString&);
```

```
        ~CString();

        char& operator[](const int&);
        CString& operator=(const CString&);
        friend ostream& operator<<(ostream&, const CString&);
};

CString::~CString()
{
        if (_mang != nullptr)
        {
                delete[] _mang;
        }
}

CString::CString()
{
        _mang = nullptr;
}

CString::CString(const char *str)
{
        int len = strlen(str);
        _mang = new char[len + 1];
        strcpy_s(_mang, len + 1, str);
}
// sử dụng ủy quyền khởi tạo từ constructor đã có
CString::CString(const CString &cs) : CString(cs._mang)
{
        cout << "The copy constructor was called." << endl;
}

CString& CString::operator=(const CString &cs)
{
        cout << "The operator= was called." << endl;
        if (_mang != nullptr)
        {
                delete[] _mang;
        }
        int len = strlen(cs._mang) + 1;
        _mang = new char[len];
        strcpy_s(_mang, len, cs._mang);
        return *this;
}

char& CString::operator[](const int& i)
{
        int len = strlen(_mang);
        if (i < 0 || i >= len)
        {
                throw exception("out of range");
        }
        return _mang[i];
}

ostream& operator<<(ostream &os, const CString &cs)
{
        os << cs._mang;
```

```
	return os;
}

int main()
{
	const int iN = 2;
	CString cs1("cs1_content");
	cout << "cs2(cs1)" << endl;
	CString cs2(cs1);
	cs2[iN] = '2';
	cout << "cs3 = cs1" << endl;
	CString cs3 = cs1;
	cs3[iN] = '3';
	cout << cs1 << endl << cs2 << endl << cs3 << endl;
	cout << "cs3 = cs2" << endl;
	cs3 = cs2;
	cout << cs3 << endl;
	system("pause");
	return 0;
}
```

```
cs2(cs1)
The copy constructor was called.
cs3 = cs1
The copy constructor was called.
cs1_content
cs2_content
cs3_content
cs3 = cs2
The operator= was called.
cs2_content
Press any key to continue . . .
```

✓ Trong hàm main có 2 dòng lệnh sử dụng operator= của CString nhưng operator= chỉ được gọi 1 lần, lần gán đầu tiên thì Copy Constructor được gọi. Như vậy hàm toán tử gán chỉ được gọi khi đối tượng đã được khởi tạo trước đó (cs3 đã được khởi tạo trước ở lần gán thứ hai). ***Do vậy nguyên tắc khi có sử dụng cấp phát động thì phải thu hồi trong hàm toán tử gán***.

Ví dụ 3 (cài đặt operator++)

```
class CString
{
private:
	char* _mang;
public:
	CString();
	CString(const char*);
	CString(const CString&);
	~CString();

	CString& operator++();		// prefix
	CString operator++(int);	// postfix
	char& operator[](const int&);
```

```
        CString& operator=(const CString&);
        friend ostream& operator<<(ostream&, const CString&);
};

/// ...

CString& CString::operator++()
{
        if (_mang != nullptr)
        {
                int len = strlen(_mang);
                _mang[len - 1]++;
        }
        return *this;
}

CString CString::operator++(int)
{
        // tạo bản copy
        CString temp = *this;
        if (_mang != nullptr)
        {
                int len = strlen(_mang);
                _mang[len - 1]++;
        }
        return temp;
}

int main()
{
        const int iN = 2;
        CString cs1("cs1_content");
        cout << cs1 << endl;
        cout << "cs1++: " << cs1++ << endl << cs1 << endl;
        cout << "++ ++cs1: " << ++ ++cs1 << endl << cs1 << endl;
        system("pause");
        return 0;
}
```

```
cs1_content
cs1++: cs1_content
cs1_contenu
++ ++cs1: cs1_contenw
cs1_contenw
Press any key to continue . . .
```

- ✓ Khi cài đặt operator++ (hoặc --) thì cần chú ý có 2 phiên bản là prefix và postfix. Về khai báo thì phân biệt bởi tham số dummy int. Về cài đặt thì postfix cần trả về 1 bản copy giá trị trước khi thay đổi (việc thay đổi giá trị là trực tiếp cho đối tượng ở cả postfix và prefix).

Ví dụ 4 (sử dụng random và template)

- ✓ Trước khi sử dụng được hàm phát sinh số ngẫu nhiên ***rand()*** thì trước hết phải thực hiện gieo hạt giống cho nó thông qua hàm ***srand(unsigned int)***, và chỉ cần gieo 1 lần cho suốt chương trình là được.

- ✓ Vấn đề cần có giá trị trong khoảng [a,b] mong muốn thì sử dụng phép đồng dư (%).
- ✓ Khi cài đặt nhiều class hoàn toàn giống nhau về phần xử lý chỉ khác duy nhất về kiểu dữ liệu thì cần sử dụng template để không phải lặp code nhiều lần. Lưu ý với C++ để đảm bảo không lỗi biên dịch thì khi cài đặt template phải để tất cả code cài đặt class trong 1 file duy nhất là file .h.

File "LinkedList.h"

```
template<class T>
struct Node
{
	T info;
	Node* pNext;
};

template<class T>
class LinkedList
{
private:
	Node<T>* _pHead, * _pTail;
	int _n;
	static Node<T>* CreateNode(const T&);

public:
	LinkedList();
	~LinkedList();

	Node<T>* AddHead(const T&);
	Node<T>* AddTail(const T&);
	Node<T>* RemoveHead();
	Node<T>* RemoveTail();

	T& operator[](const int&);

	template<class T>
	friend ostream& operator<<(ostream&, const LinkedList<T>&);
};

template<class T>
LinkedList<T>::LinkedList()
{
	_pHead = _pTail = nullptr;
	_n = 0;
}

template<class T>
LinkedList<T>::~LinkedList()
{
	_n = 0;
	Node<T>* node = _pHead;
	while (_pHead != nullptr)
	{
		_pHead = _pHead->pNext;
		delete node;
		node = _pHead;
```

```
	}
}

template<class T>
Node<T>* LinkedList<T>::CreateNode(const T& value)
{
	Node<T>* node = new Node<T>{ value, nullptr };
	return node;
}

template<class T>
Node<T>* LinkedList<T>::AddTail(const T& value)
{
	Node<T>* node = CreateNode(value);
	if (node == nullptr)
	{
		return nullptr;
	}
	if (_pHead == nullptr)
	{
		_pHead = _pTail = node;
		_n++;
		return node;
	}
	_pTail->pNext = node;
	_pTail = node;
	_n++;
	return node;
}

template<class T>
Node<T>* LinkedList<T>::AddHead(const T& value)
{
	// tự code
}

template<class T>
Node<T>* LinkedList<T>::RemoveHead()
{
	if (_pHead == nullptr)
	{
		return nullptr;
	}
	Node<T>* node = _pHead;
	_pHead = _pHead->pNext;
	_n--;
	if (_pHead == nullptr)
	{
		_pTail = nullptr;
	}
	return node;
}

template<class T>
Node<T>* LinkedList<T>::RemoveTail()
{
	// tự code
}
```

```
template<class T>
T& LinkedList<T>::operator[](const int& i)
{
	if (i < 0 || i >= _n)
	{
		throw exception("out of range");
	}
	// tự code
}

template<class T>
ostream& operator<<(ostream &os, const LinkedList<T> &ll)
{
	Node<T>* node = ll._pHead;
	while (node != nullptr)
	{
		os << node->info << ", ";
		node = node->pNext;
	}
	return os;
}
```