## Nội dung tuần 05

Ứng dụng các lớp đối tượng trong thư viện chuẩn **STL**.

## Bài tập

- Yêu cầu

  A: bài 1

  H: Làm hết 4 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 hoàn thiện ví dụ 2.

### Bài 3

Cài đặt hoàn thiện ví dụ 3.

### Bài 4

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

```
int main()
{
	MyString ms1("abcdsf");
	MyString ms2 = "____" + ms1;
	MyString ms3 = ms1 + ms2;
	cout << "ms1= " << ms1 << endl;
	cout << "ms2= " << ms2 << endl;
	cout << "ms3= " << ms3 << endl << endl;
	MyString ms = "a,b,c;d.r;.,h;e,w__u,t.f;j_..";
	vector<char> arrChar;
	arrChar.push_back(',');
	arrChar.push_back('.');
	arrChar.push_back(';');
	vector<MyString> vMs = ms.Split(arrChar, false);
	cout << "Split:" << endl;
	for (vector<MyString>::iterator itMS = vMs.begin(); itMS != vMs.end();
itMS++)
	{
		cout << *itMS << " ";
	}
```

```
	cout << endl << "size= " << vMs.size() << endl << endl;
	vMs = ms.Split(arrChar, true);
	cout << "Split co loai bo empty:" << endl;
	for (vector<MyString>::iterator itMS = vMs.begin(); itMS != vMs.end();
itMS++)
	{
		cout << *itMS << " ";
	}
	cout << endl << "size= " << vMs.size() << endl << endl;

	system("pause");
	return 0;
}
```

```
ms1= abcdsf
ms2= ____abcdsf
ms3= ____abcdsfabcdsf

Split:
a b c d r   h e w__u t f j_
size= 15

Split co loai bo empty:
a b c d r h e w__u t f j_
size= 11

Press any key to continue . . .
```

## Hướng dẫn

### Ví dụ 1 (kiểu dữ liệu tự định nghĩa trong lớp đối tượng)

```
#define NOE 10
class A
{
private:
	int info[NOE];
public:
	A();

	typedef int* indexOf;
	indexOf GetInfo(const int&);

	void Xuat(ostream&);
};

A::A()
{
	for (int i = 0; i < NOE; ++i)
	{
		info[i] = i + NOE;
	}
}

A::indexOf A::GetInfo(const int &i)
{
	indexOf rt = info + i;
	return rt;
}
```

```
void A::Xuat(ostream &os)
{
	for (int i = 0; i < NOE; ++i)
	{
		os << info[i] << ", ";
	}
	os << endl;
}

int main()
{
	const int i = 7;
	A a;
	A::indexOf io = a.GetInfo(i);
	cout << *io << endl;
	*io = 10;
	a.Xuat(cout);

	system("pause");
	return 0;
}
```

✓ Chú ý dòng tô vàng là định nghĩa kiểu dữ liệu indexOf trong lớp A. Mục đích sử dụng như là tham chiếu chỉ mục đến phần tử của A. Kết quả như hình dưới.

```
17
10, 11, 12, 13, 14, 15, 16, 10, 18, 19,
Press any key to continue . . .
```

## Ví dụ 2 (iterator trong STL)

✓ Thư viện chuẩn STL (***Standard Template Library***) cung cấp một tập hợp các lớp và hàm mẫu (templates) được thiết kế sẵn, cho phép sử dụng các cấu trúc dữ liệu và thuật toán phổ biến một cách hiệu quả và dễ dàng.

✓ Các **container** trong thư viện chuẩn **STL** đều cung cấp giải pháp sử dụng ***iterator*** để duyệt các phần tử, đặc biệt với các **container** không có toán tử chỉ mục như ***list*** thì bắt buộc chỉ có dùng ***iterator*** để duyệt phần tử. Về bản chất thì ***iterator*** cũng chỉ là kiểu dữ liệu mới được khai báo trong lớp **container**.

✓ Để sử dụng được các hàm mẫu trong STL thì phải xây dựng class có cung cấp các chức năng iterator tiêu chuẩn

```
template<class T>
class SLList
{
private:
	struct Node
	{
		T _info;
		Node* _pNext;
	};
```

```
	Node* _pHead, * _pTail;
	int _n;
	static Node* CreateNode(const T &value)
	{
		Node* node = new Node{ value, nullptr };
		return node;
	}

public:
	class Iterator;

	SLList();
	~SLList();

	void AddHead(const T&);
	void AddTail(const T&);
	void RemoveHead();
	void RemoveTail();
	void Clear();

	/// --- Iterator Access ---
	Iterator begin() {
		return Iterator(_pHead);
	}
	Iterator end() {
		return Iterator(nullptr); // Iterator end() trỏ tới nullptr
	}

	template<class T>
	friend ostream& operator<<(ostream& os, const SLList<T>& ll)
	{
		SLList<T>::Node* node = ll._pHead;
		while (node != nullptr)
		{
			os << node->_info << ", ";
			node = node->_pNext;
		}
		return os;
	}
};

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

template<class T>
SLList<T>::~SLList()
{
	// tự code
}

template<class T>
void SLList<T>::Clear()
{
```

```
	// tự code
}

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

template<class T>
void SLList<T>::AddHead(const T& value)
{
	// tự code
}

template<class T>
void SLList<T>::RemoveHead()
{
	// tự code
}

template<class T>
void SLList<T>::RemoveTail()
{
	// tự code
}

template<class T>
class SLList<T>::Iterator
{
public:
	// Các type alias cần thiết cho iterator traits (để tương thích STL)
	using iterator_category = forward_iterator_tag;     // Chỉ duyệt tiến
	using difference_type = ptrdiff_t;
	using value_type = T;
	using pointer = T*;
	using reference = T&;
private:
	Node* current_node;
	Iterator(Node* node) : current_node(node) {}
	friend class SLList<T>;
public:
	Iterator() : current_node(nullptr) {}
	reference operator*() const {
		if (!current_node) {
```

```
                throw out_of_range("Dereferencing end() or null iterator");
            }
            return current_node->_info;
        }
        pointer operator->() const {
            if (!current_node) {
                throw out_of_range("Accessing member via end() or null
iterator");
            }
            return &(current_node->_info);
        }
        Iterator& operator++() {
            if (current_node) {
                current_node = current_node->_pNext;
            }
            else {
                throw out_of_range("Incrementing end() or null iterator");
            }
            return *this;
        }
        Iterator operator++(int) {
            if (!current_node) {
                throw out_of_range("Incrementing end() or null iterator");
            }
            Iterator temp = *this;
            current_node = current_node->next;
            return temp;
        }
        bool operator==(const Iterator& other) const {
            return current_node == other.current_node;
        }
        bool operator!=(const Iterator& other) const {
            return !(*this == other);
        }
};

#include<algorithm>

void fnAction(int val)
{
        cout << val << ", ";
}

void fnAction2(int &val)
{
        val *= 2;
}

bool fnPredict(int val)
{
        const int threshold = 3;
        return val > threshold;
}

int main()
{
        SLList<int> l;
        l.AddTail(1);
```

```
	l.AddTail(1);
	l.AddTail(0);
	l.AddTail(4);
	l.AddTail(2);
	l.AddTail(5);
	/// duyệt sử dụng iterator (chuẩn STL)
	for (SLList<int>::Iterator it = l.begin(); it != l.end(); ++it)
	{
		cout << *it << ", ";
	}
	cout << endl;
	/// sử dụng hàm for_each trong algorithm
	for_each(l.begin(), l.end(), fnAction2);
	for_each(l.begin(), l.end(), fnAction);
	cout << endl;
	/// sử dụng hàm replace_if trong algorithm
	replace_if(l.begin(), l.end(), fnPredict, 100);
	cout << l << endl;
	/// sử dụng hàm fill trong algorithm
	fill(l.begin(), l.end(), 123);
	cout << l << endl;
	l.Clear();

	system("pause");
	return 0;
}
```

```
1, 1, 0, 4, 2, 5,
2, 2, 0, 8, 4, 10,
2, 2, 0, 100, 100, 100,
123, 123, 123, 123, 123, 123,
Press any key to continue . . .
```

### Ví dụ 3 (ứng dụng STL)

```
class SoNguyenLon
{
private:
	vector<unsigned char> lCS;
	void Pow10(const int& n);

public:
	SoNguyenLon(void);
	SoNguyenLon(const int& cs, const int& scs);
	SoNguyenLon(const unsigned long& n);
	SoNguyenLon(const SoNguyenLon& snl);
	SoNguyenLon(const char *s);
	~SoNguyenLon(void);

	int SoCS();
	SoNguyenLon operator+(SoNguyenLon snl);
	SoNguyenLon operator-(SoNguyenLon snl);
	SoNguyenLon operator*(SoNguyenLon snl);
	bool operator>(SoNguyenLon& snl);
	const SoNguyenLon& operator=(const SoNguyenLon& snl);
	SoNguyenLon& operator+=(SoNguyenLon snl);
```

```
	friend SoNguyenLon operator+(const unsigned int& n, const SoNguyenLon&
snl);
	friend SoNguyenLon operator-(const unsigned int& n, const SoNguyenLon&
snl);
	friend ostream& operator<<(ostream& os, const SoNguyenLon &snl);
};

void SoNguyenLon::Pow10(const int& n)
{
	for (int i = 0; i < n; ++i)
	{
		lCS.insert(lCS.begin(), 0);
	}
}

SoNguyenLon::SoNguyenLon(void)
{
	lCS.push_back(0);
}

SoNguyenLon::SoNguyenLon(const int& cs, const int& scs)
{
	int csR = cs;
	if (csR < 1)
	{
		csR = 1;
	}
	if (csR > 9)
	{
		csR = 9;
	}
	int soCS = abs(scs);
	if (soCS < 1)
	{
		soCS = 1;
	}
	for (int i = 0; i < soCS; ++i)
	{
		lCS.push_back(csR);
	}
}

SoNguyenLon::SoNguyenLon(const unsigned long& n)
{
	unsigned long temp = n;
	while (temp > 0)
	{
		lCS.push_back(temp % 10);
		temp /= 10;
	}
}

SoNguyenLon::SoNguyenLon(const SoNguyenLon& snl)
{
	lCS = snl.lCS;
}

SoNguyenLon::SoNguyenLon(const char *s)
```

```
{
	// tự code
}

SoNguyenLon::~SoNguyenLon(void)
{
}

int SoNguyenLon::SoCS()
{
	return lCS.size();
}

bool SoNguyenLon::operator>(SoNguyenLon& snl)
{
	if (lCS.size() > snl.lCS.size())
	{
		return true;
	}
	if (lCS.size() < snl.lCS.size())
	{
		return false;
	}
	for (int i = lCS.size() - 1; i >= 0; --i)
	{
		if (lCS[i] == snl.lCS[i])
		{
			continue;
		}
		if (lCS[i] > snl.lCS[i])
		{
			return true;
		}
		return false;
	}
	return false;
}

const SoNguyenLon& SoNguyenLon::operator=(const SoNguyenLon& snl)
{
	lCS = snl.lCS;
	return *this;
}

SoNguyenLon& SoNguyenLon::operator+=(SoNguyenLon snl)
{
	// tự code
}

SoNguyenLon SoNguyenLon::operator+(SoNguyenLon snl)
{
	SoNguyenLon snlKQ;
	snlKQ.lCS.clear();
	SoNguyenLon* snlSCSMax = (SoCS() > snl.SoCS()) ? this : &snl;
	SoNguyenLon* snlSCSMin = (SoCS() < snl.SoCS()) ? this : &snl;
	int nho = 0, temp;
	for (int i = 0; i < snlSCSMin->SoCS(); ++i)
	{
```

```
		temp = lCS[i] + snl.lCS[i] + nho;
		snlKQ.lCS.push_back(temp % 10);
		nho = temp / 10;
	}
	for (int i = snlSCSMin->SoCS(); i < snlSCSMax->SoCS(); ++i)
	{
		temp = snlSCSMax->lCS[i] + nho;
		snlKQ.lCS.push_back(temp % 10);
		nho = temp / 10;
	}
	if (nho > 0)
	{
		snlKQ.lCS.push_back(1);
	}
	return snlKQ;
}

SoNguyenLon SoNguyenLon::operator-(SoNguyenLon snl)
{
	// tự code
}

SoNguyenLon SoNguyenLon::operator*(SoNguyenLon snl)
{
	SoNguyenLon snlKQ, * psnlTemp;
	SoNguyenLon* snlSCSMax = (SoCS() > snl.SoCS()) ? this : &snl;
	SoNguyenLon* snlSCSMin = (SoCS() < snl.SoCS()) ? this : &snl;
	int nho = 0, temp;
	for (int i = 0; i < snlSCSMin->SoCS(); ++i)
	{
		psnlTemp = new SoNguyenLon;
		psnlTemp->lCS.clear();
		nho = 0;
		for (int j = 0; j < snlSCSMax->SoCS(); ++j)
		{
			temp = snlSCSMin->lCS[i] * snlSCSMax->lCS[j] + nho;
			psnlTemp->lCS.push_back(temp % 10);
			nho = temp / 10;
		}
		if (nho > 0)
		{
			psnlTemp->lCS.push_back(nho);
		}
		psnlTemp->Pow10(i);
		snlKQ += *psnlTemp;
		delete psnlTemp;
	}
	return snlKQ;
}

SoNguyenLon operator+(const unsigned int& n, const SoNguyenLon& snl)
{
	// tự code
}

SoNguyenLon operator-(const unsigned int& n, const SoNguyenLon& snl)
{
	// tự code
```

```
}

ostream& operator<<(ostream& os, const SoNguyenLon &snl)
{
	for (int i = snl.lCS.size() - 1; i >= 0; --i)
		os << (int)snl.lCS[i];
	return os;
}

int main()
{
	const int iTest = 78912;
	SoNguyenLon snl1(123);
	SoNguyenLon snl2(40000);
	cout << snl1 << " + " << snl2 << " = ";
	cout << (snl1 + snl2) << endl;
	cout << snl1 << " * " << snl2 << " = ";
	cout << (snl1 * snl2) << endl;
	cout << snl1 << " - " << snl2 << " = ";
	cout << (snl1 - snl2) << endl;
	cout << snl2 << " - " << snl1 << " = ";
	cout << (snl2 - snl1) << endl;
	cout << iTest << " - " << snl2 << " = ";
	cout << (iTest - snl2) << endl;

	system("pause");
	return 0;
}
```

```
123 + 40000 = 40123
123 * 40000 = 4920000
123 - 40000 = 0
40000 - 123 = 39877
78912 - 40000 = 38912
Press any key to continue . . .
```