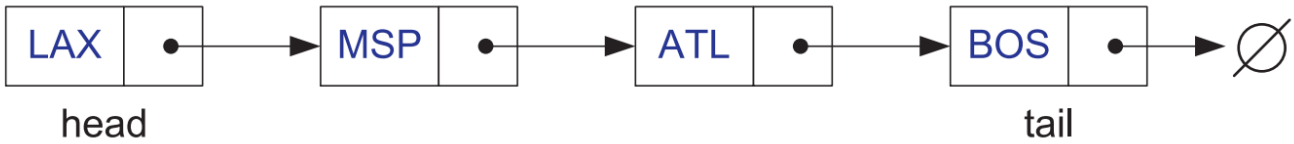
Danh sách liên kết đơn

Một **danh sách liên kết**, ở dạng đơn giản nhất, là một tập hợp các nút cùng nhau tạo thành một thứ tự tuyến tính. Mỗi nút có một con trỏ, gọi là `next`, trỏ đến nút tiếp theo trong danh sách. Ngoài ra, mỗi nút sẽ lưu trữ phần tử gắn với nó.



Ví dụ, hình trên đây mô tả một danh sách liên kết đơn để lưu trữ các mã sân bay. Danh sách này gồm 4 nút.

- Nút đầu tiên lưu trữ phần tử ***LAX***, con trỏ `next` của nút này trỏ đến nút có phần tử **MSP**.
- Nút thứ ba lưu trữ phần tử **ATL** và có `next` trỏ đến nút thứ tư, là nút lưu trữ phần tử **BOS**. Đây là nút cuối cùng trong danh sách và nút này có `next` bằng `NULL`.

Cấu trúc này gọi là **danh sách liên kết đơn** bởi vì mỗi nút chỉ lưu trữ một liên kết đơn.

1. Cài đặt danh sách liên kết đơn

Ta sẽ cài đặt một danh sách liên kết của các xâu (`string`). Mỗi nút của nó lưu trữ hai giá trị:

- `elem` là phần tử gắn với nút, ở đây nó là một `string`; và
- `next` lưu trữ con trỏ tới nút tiếp theo.

Chúng ta cho phép lớp danh sách liên kết `StringLinkedList` là **friend** của `StringNode` để nó có thể truy cập vào các thành viên `private` của lớp `StringNode`.

```
class StringNode
{
    private:
        string elem;
        StringNode *next;
        friend class StringLinkedList;
};
```

Đoạn mã dưới đây định nghĩa danh sách liên kết `StringLinkedList` . Dữ liệu `private` là con trỏ `head` tới nút đầu của danh sách.

```
class StringLinkedList
{
public:
    StringLinkedList();
    ~StringLinkedList();
    bool empty() const;                 // Kiểm tra danh sách có rỗng
    const string &front() const;        // dữ liệu đầu tiên trong danh sách
    void addFront (const string& e);    // thêm dữ liệu vào đầu danh sách
    void removeFront ();                // xóa dữ liệu đầu tiên trong danh sách
    void print ();                      // in toàn bộ danh sách ra màn hình
private:
    StringNode *head;                   // con trỏ đến nút đầu tiên trong danh sách
};
```

- Cấu tử tạo ra một danh sách rỗng bằng cách gán con trỏ `head` là `NULL`.

```
StringLinkedList::StringLinkedList():
    head(NULL) {  }
```

- Hủy tử sẽ gọi hàm `removeFront` để xóa hết các phần tử trong danh sách.

```
StringLinkedList::~StringLinkedList()
{ while (!empty()) removeFront(); }
```

- Hàm `empty()` trả về `true` nếu danh sách rỗng, trả về `false` nếu không. Hàm này chỉ đơn giản kiểm tra xem con trỏ `head` có bằng `NULL` hay không.

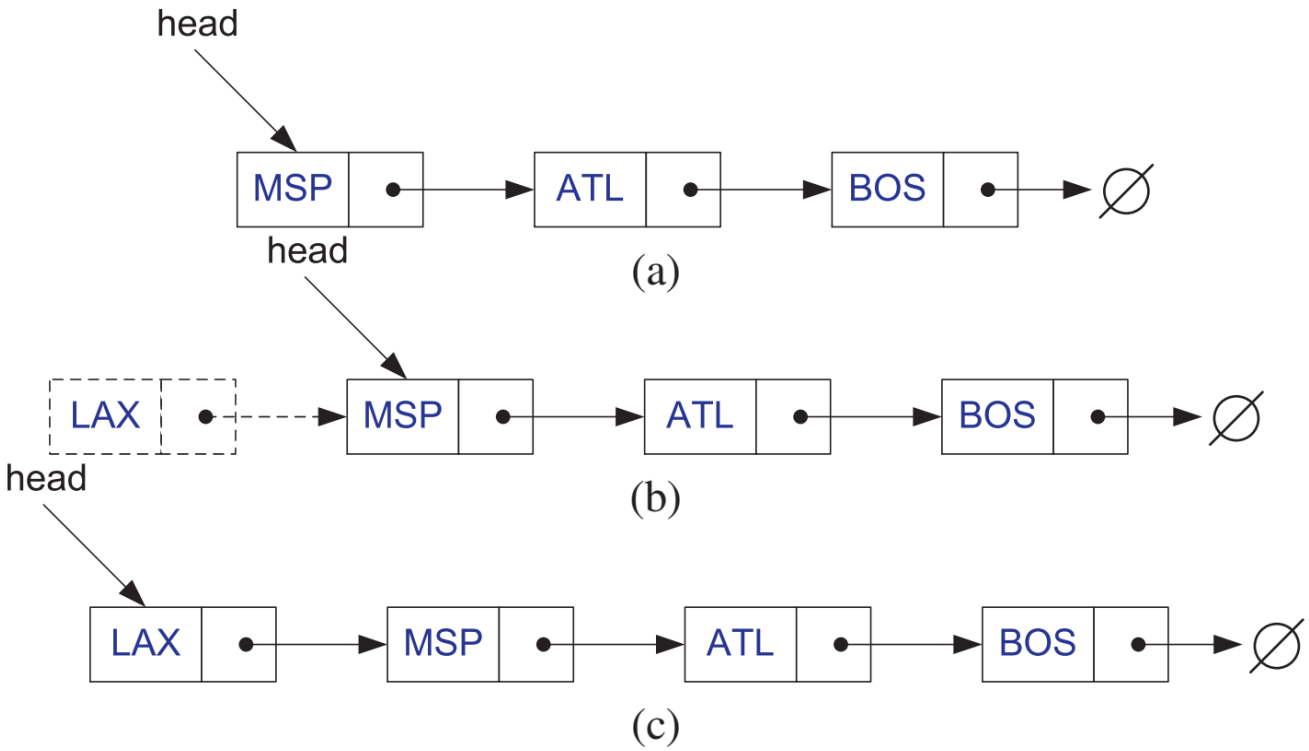
```
bool StringLinkedList::empty() const
{ return head == NULL; }
```

- Hàm `front()` trả về phần tử đầu tiên của danh sách.

```
const string& StringLinkedList::front() const
{ return head->elem; }
```

2. Thêm một phần tử vào đầu danh sách

Để thêm một nút vào đầu danh sách, ta chỉ đơn giản tạo ra một nút mới, cho con trỏ `next` của nút mới này trỏ đến đầu danh sách (tức là nơi trỏ bởi con trỏ `head`) và đưa con trỏ `head` trỏ vào nút mới này.

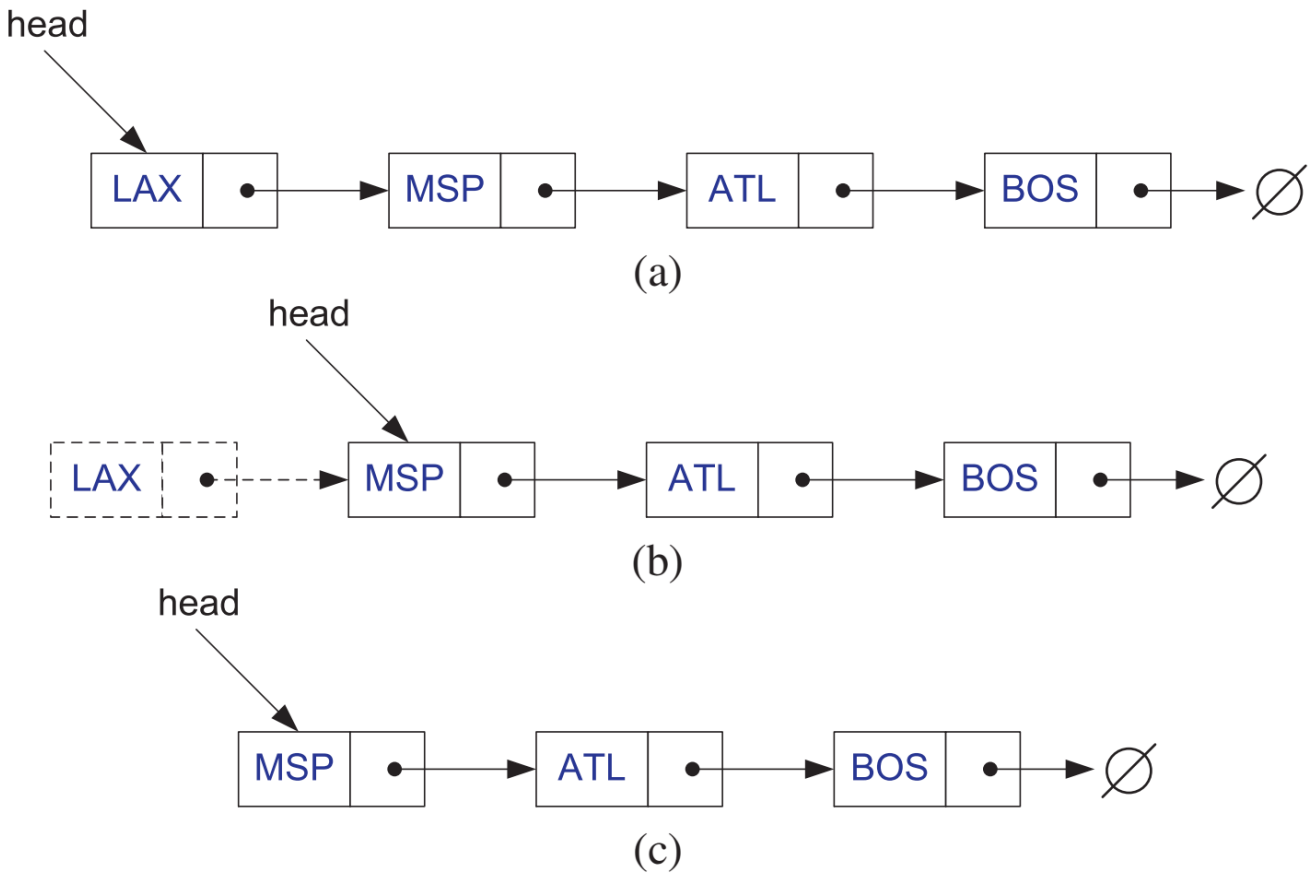


Các hình trên mô tả ba bước nói ở trên. Hình (a) mô tả danh sách liên kết trước khi thêm, gồm ba phần tử **MSP**, **ALT**, và **BOS**. Hình (b) mô tả việc tạo ra một nút mới với phần tử **LAX** và cho con trỏ `next` của nó trỏ đến nút có phần tử **MSP**. Và hình (c) mô tả việc đưa con trỏ `head` trỏ đến nút mới này. Thao tác này được cài đặt như sau.

```
void StringLinkedList::addFront(const string& e)
{
    StringNode* v = new StringNode;
    v->elem = e;
    v->next = head;
    head = v;
}
```

3. Xóa phần tử ở đầu danh sách

Để xóa phần tử ở đầu danh sách, ta phải lưu lại con trỏ tới nút `head` cũ và đặt con trỏ `head` trỏ tới nút tiếp theo trong danh sách, và xóa nút trỏ bởi con trỏ `head` cũ.



Ba bước trên được mô tả bởi các hình trên. Hình (a) mô tả danh sách ban đầu gồm 4 nút. Hình (b) mô tả việc di chuyển con trỏ head trỏ đến nút tiếp theo. Hình (c) mô tả việc xóa nút đầu khỏi danh sách. Thao tác này được cài đặt như sau.

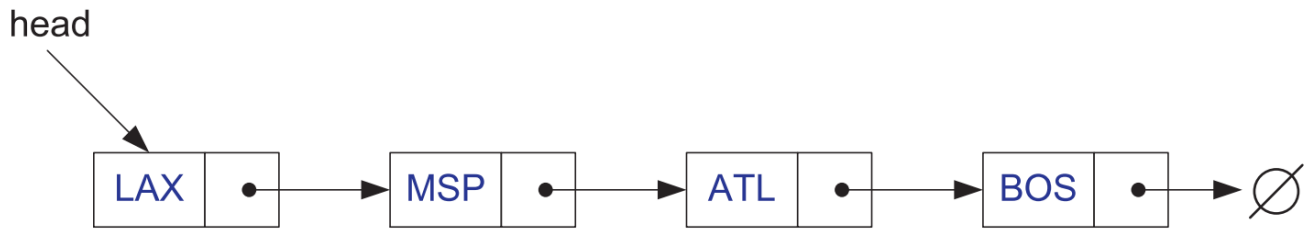
```
void StringLinkedList::removeFront(){
    if (!empty()){
        StringNode *v = head;
        head = head->next;
        delete v;
    }
}
```

4. In danh sách

Để in danh sách, ta bắt đầu tử con trỏ `head`, in nút hiện tại, và di chuyển tới nút tiếp theo. Dừng khi tới phần tử cuối dãy, tức con trỏ tới nút hiện tại là `NULL`.

```
void StringLinkedList::print(){
    StringNode *v = head;
    while (v!=NULL) {
        cout<<v->elem<<endl;
        v = v->next;
    }
}
```

Với các hàm trên, để có được danh sách như hình dưới đây



ta cài đặt hàm `main` với bốn lần gọi hàm `addFront()` như dưới đây.

```
int main(int argc, char const *argv[])
{
    StringLinkedList x;
    x.addFront ("BOS");
    x.addFront ("ATL");
    x.addFront ("MSP");
    x.addFront ("LAX");
    x.print ();
    return 0;
}
```

5. Danh sách liên kết với kiểu bất kỳ

Cài đặt danh sách liên kết đơn ở trên giả sử rằng kiểu dữ liệu chúng ta muốn lưu trữ là kiểu `string`. Ta có thể dùng `template` để tạo ra danh sách có kiểu bất kỳ. Danh sách này gọi là `SLinkedList`.

Ta bắt đầu với lớp nút, gọi là `SNode`, cài đặt như dưới đây. Kiểu phần tử gắn với mỗi nút được tham số hóa bởi biến kiểu `E`. Bởi vậy, khác với lớp `StringNode` ở trên, kiểu dữ liệu `string` được thay bởi `E`. Để tham chiếu đến lớp có kiểu bất kỳ, ta thêm hậu tố `<E>` vào lớp nút và lớp danh sách. Ví dụ, lớp `SLinkedList<E>` có lớp nút gắn với nó là `SNode<E>`.

```
template<typename E>
class SNode{
    private:
        E elem;
        SNode<E> *next;
        friend class SLinkedList<E>;
}
```

Lớp danh sách liên kết đơn với kiểu bất kỳ được trình bày như dưới đây.

```
template <typename E>
class SLinkedList {
public:
    SLinkedList();
    ~SLinkedList();
    bool empty() const;
    const E& front() const;
    void addFront(const E& e);
    void removeFront();
private:
    SNode<E>* head;
};
```

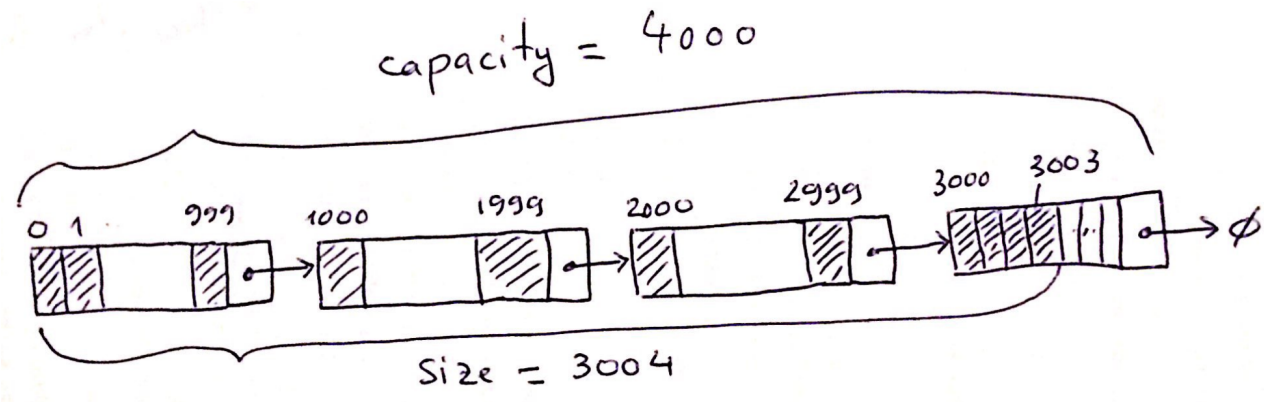
Bài tập: Hãy cài đặt đầy đủ lớp `SLinkedList` và viết hàm `main` để test lớp `SLinkedList<int>`.

6. Bài tập

1. Thêm hàm `size()` cho lớp danh sách liên kết đơn. Liệu bạn có thể thiết kế hàm này chạy trong thời gian $O(1)$?
2. Hãy cài đặt thuật toán trả về một con trỏ tới ***nút trước nút cuối cùng*** trong danh sách liên kết đơn. Nút cuối cùng trong danh sách liên kết đơn là nút có con trỏ `next` bằng `NULL`.
3. Hãy mô tả thuật toán ghép hai danh sách liên kết đơn L và M thành một danh sách đơn L'. Danh sách L' chứa các nút của L sau đó đến các nút của M.
4. **(Bạn phải nộp mã nguồn đầy đủ của bài tập này)**. Hãy sử dụng danh sách liên kết đơn để cài đặt lớp `Vector` đã học trong buổi trước.

   Cài đặt được mô tả như sau:

   - Mỗi nút của danh sách liên kết có phần tử là 1000 phần tử của `Vector` ; `next` là con trỏ tới nút tiếp theo.
   - `size` của `Vector` là kích số phần tử của `Vector`,
   - `capacity` chỗ chứa đã được cấp phát. `capacity` = `size` + *chỗ đã cấp phát nhưng chưa dùng đến*. Giá trị này là bội của 1000; và nên là giá trị nhỏ nhất lớn hơn `size` và là bội của 1000.



Vì vậy, để truy cập vào `v[3003]` thì ta phải truy cập vào phần tử của nút thứ tư, lấy chỉ số 3 của mảng này. Tức là phần tử `head->next->next->next->elem[3]`.

Cụ thể, lớp `Vector` được khai báo như sau:

```
#define MAX 1000
template<typename T> class Vector;
template<typename T>
class SNode {
    private:
        T elem[MAX];        // Mỗi nút chứa MAX phần tử của vector
        SNode<T> * next;
        friend class Vector<T>;
};
template<typename T>
class Vector {
public:
    Vector() {head = NULL;}
    ~Vector() ;
    T& operator[](int i);
    int size() const { return sz; }
    void push_back(const E &e);
private:
    SNode* head;   // con trỏ tới phần tử đầu của mảng
    int sz;        // số phần tử
    int capacity;  // chỗ chứa thực sự của Vector
};
```

7. Mã nguồn cho lớp `SLinkedList`

- Cài đặt đầy đủ lớp `SLinkedList` trong file **slinkedlist.h**.

```
#ifndef SLINKEDLIST_H
#define SLINKEDLIST_H
#include <iostream>
using namespace std;

template <typename E> class SLinkedList;
template<typename E>
class SNode {
    private:
        E elem;
        SNode<E> *next;
        friend class SLinkedList<E>;
};

template<typename E>
class  SLinkedList {
    private:
        SNode<E> *head;
    public:
        SLinkedList ();
        ~SLinkedList ();
        bool empty() const;
        const E &front() const;
        void addFront (const E& e);
        void removeFront ();
        void print () const;
};

template<typename E>
SLinkedList<E>::SLinkedList() {
    head = NULL;
}
template<typename E>
SLinkedList<E>::~SLinkedList() {
        while (!empty()) removeFront();
}
template<typename E>
bool SLinkedList<E>::empty() const {
    return head == NULL;
}
template<typename E>
const E &SLinkedList<E>::front() const{
    return head->elem;
}
template<typename E>
void SLinkedList<E>::addFront (const E& e){
    SNode<E> * v = new SNode<E>;
```

```
    v->elem = e;
    v->next = head;
    head = v;
}
template<typename E>
void SLinkedList<E>::removeFront(){
    if (empty()) return ;
    SNode<E> *v = head;
    head = head->next;
    delete v;
}
template<typename E>
void SLinkedList<E>::print () const{
    SNode<E> *v = head;
    while (v!=NULL){
        cout<<v->elem<<" -> ";
        v = v->next;
    }
}
#endif
```

- Hàm `main` được cài đặt trong file **main.cpp** như dưới đây.

```
#include "node.h"
int main(int argc, char const *argv[]) {
    SLinkedList<int> x;
    x.addFront(1);  x.addFront(3); x.addFront(5);
    x.addFront(7);  x.addFront(9);
    x.print();
    return 0;
}
```