Singly Linked Lists

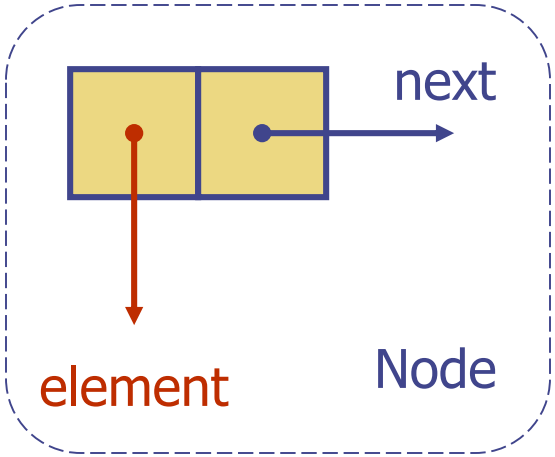
Jyh-Shing Roger Jang (張智星)
CSIE Dept, National Taiwan University

Arrays vs. Linked Lists

- Arrays
  - Pros:
    - Simple concept
    - Easy coding
  - Cons:
    - Difficult in resizing due to consecutive memory requirement
    - Slow in insertions and deletions
- Linked lists
  - Pros:
    - Flexible in resizing
    - Quick in insertions and deletions
  - Cons:
    - Advanced coding
    - More prone to memory leak

Singly Linked List

- A singly linked list (SLL) is a data structure consisting of a sequence of nodes which are not necessarily stored in consecutive memory
- Each node stores
  - Element of the node
  - Link to the next node



Comparison of Memory Usage

- Arrays and singly linked lists have different ways of using memory
  - Array: A=(int *)malloc(4*sizeof(int));
  - Linked list:

Class Definitions for SLL

```
class StringNode {                          // a node in a list of strings
private:
  string elem;                              // element value
  StringNode* next;                         // next item in the list

  friend class StringLinkedList;            // provide StringLinkedList access
};
```

Code Fragment 3.13: A node in a singly linked list of strings.

```
class StringLinkedList {                    // a linked list of strings
public:
  StringLinkedList();                       // empty list constructor
  ~StringLinkedList();                      // destructor
  bool empty() const;                       // is list empty?
  const string& front() const;              // get front element
  void addFront(const string& e);           // add to front of list
  void removeFront();                       // remove front item list
private:
  StringNode* head;                         // pointer to the head of list
};
```

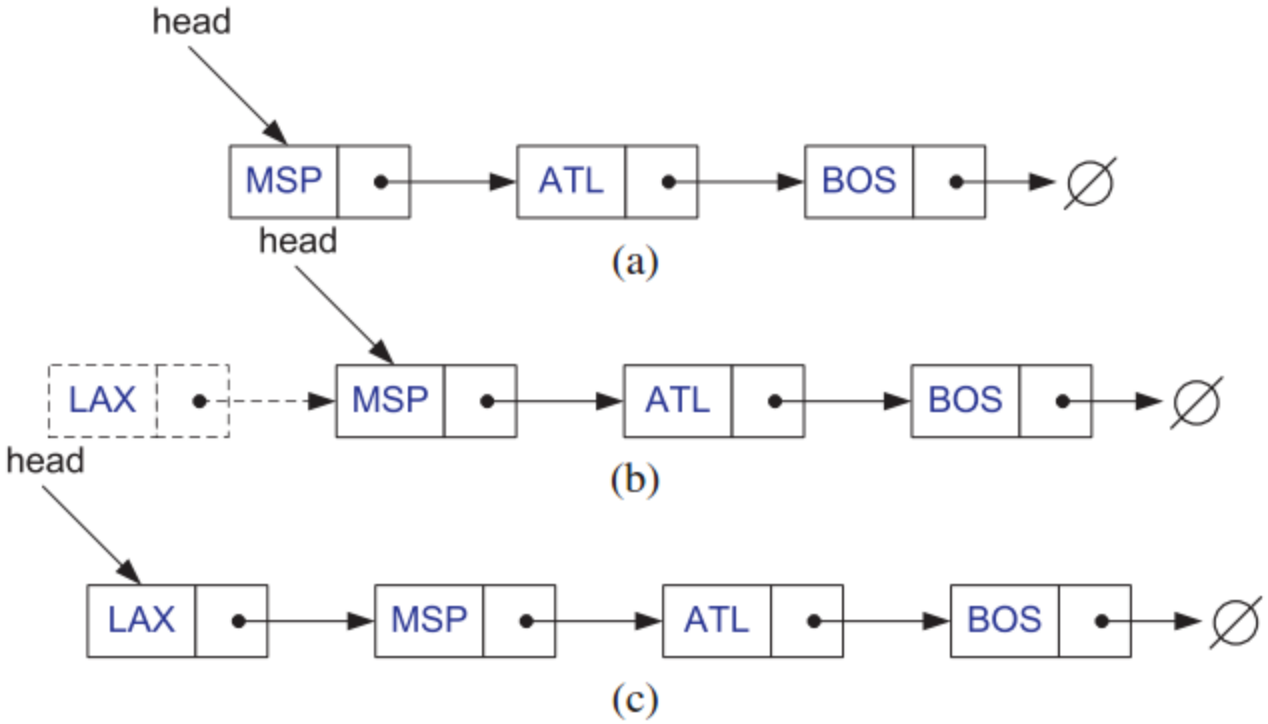
Code Fragment 3.14: A class definition for a singly linked list of strings.

Member Functions for SLL

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

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

bool StringLinkedList::empty() const                    // is list empty?
  { return head == NULL; }

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

Code Fragment 3.15: Some simple member functions of class StringLinkedList.

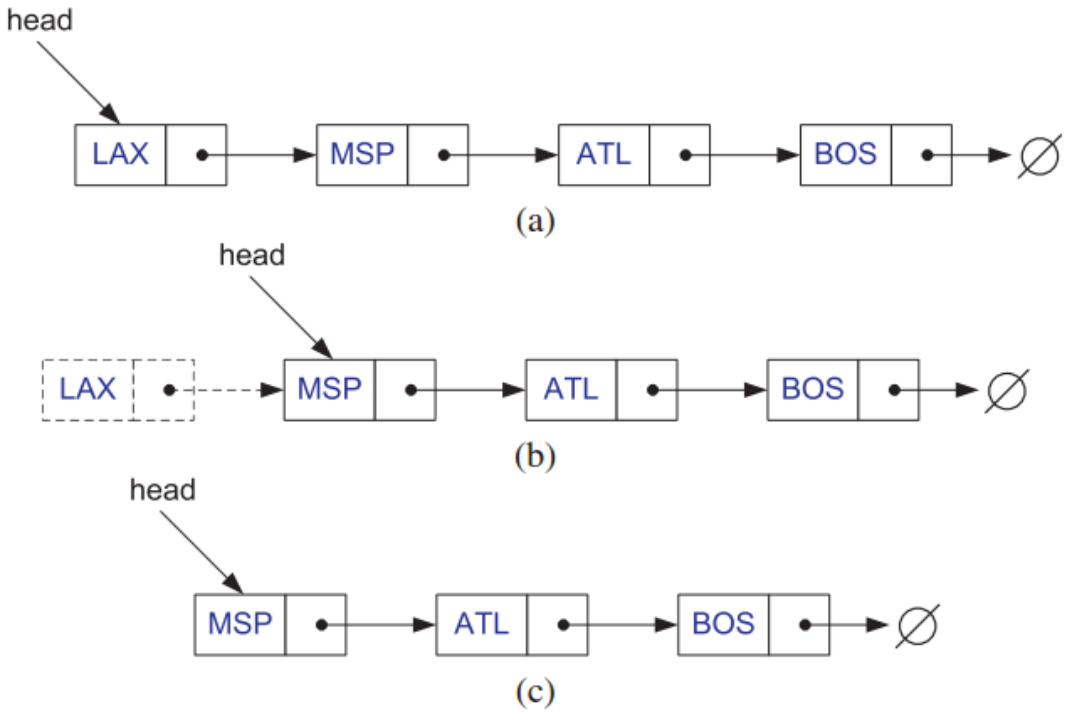
Insertion at the Head




```
void StringLinkedList::addFront(const string& e) {  // add to front of list
  StringNode* v = new StringNode;                    // create new node
  v->elem = e;                                       // store data
  v->next = head;                                    // head now follows v
  head = v;                                          // v is now the head
}
```

Code Fragment 3.16: Insertion to the front of a singly linked list.

Removal from the Head




```
void StringLinkedList::removeFront() {          // remove front item
  StringNode* old = head;                       // save current head
  head = old->next;                             // skip over old head
  delete old;                                   // delete the old head
}
```

Code Fragment 3.17: Removal from the front of a singly linked list.

Generic Singly Linked Lists

```
template <typename E>
class SNode {                                   // singly linked list node
private:
  E elem;                                       // linked list element value
  SNode<E>* next;                               // next item in the list
  friend class SLinkedList<E>;                  // provide SLinkedList access
};
```

Code Fragment 3.18: A node in a generic singly linked list.

```
template <typename E>
class SLinkedList {                             // a singly linked list
public:
  SLinkedList();                                // empty list constructor
  ~SLinkedList();                               // destructor
  bool empty() const;                           // is list empty?
  const E& front() const;                       // return front element
  void addFront(const E& e);                    // add to front of list
  void removeFront();                           // remove front item list
private:
  SNode<E>* head;                               // head of the list
};
```

Code Fragment 3.19: A class definition for a generic singly linked list.

Example of Generic SLL

- By using generic SLL, you can put any data of any types into the data part of a node.
  - Example