List - Array List and Singly Linked List

Data Structures and Algorithms

Luu Quang Huan, MsC
Faculty of Computer Science and Engineering
Ho Chi Minh University of Technology, VNU-HCM

Overview

Luu Quang Huan, MsC

Linear list concepts

Luu Quang Huan, MsC

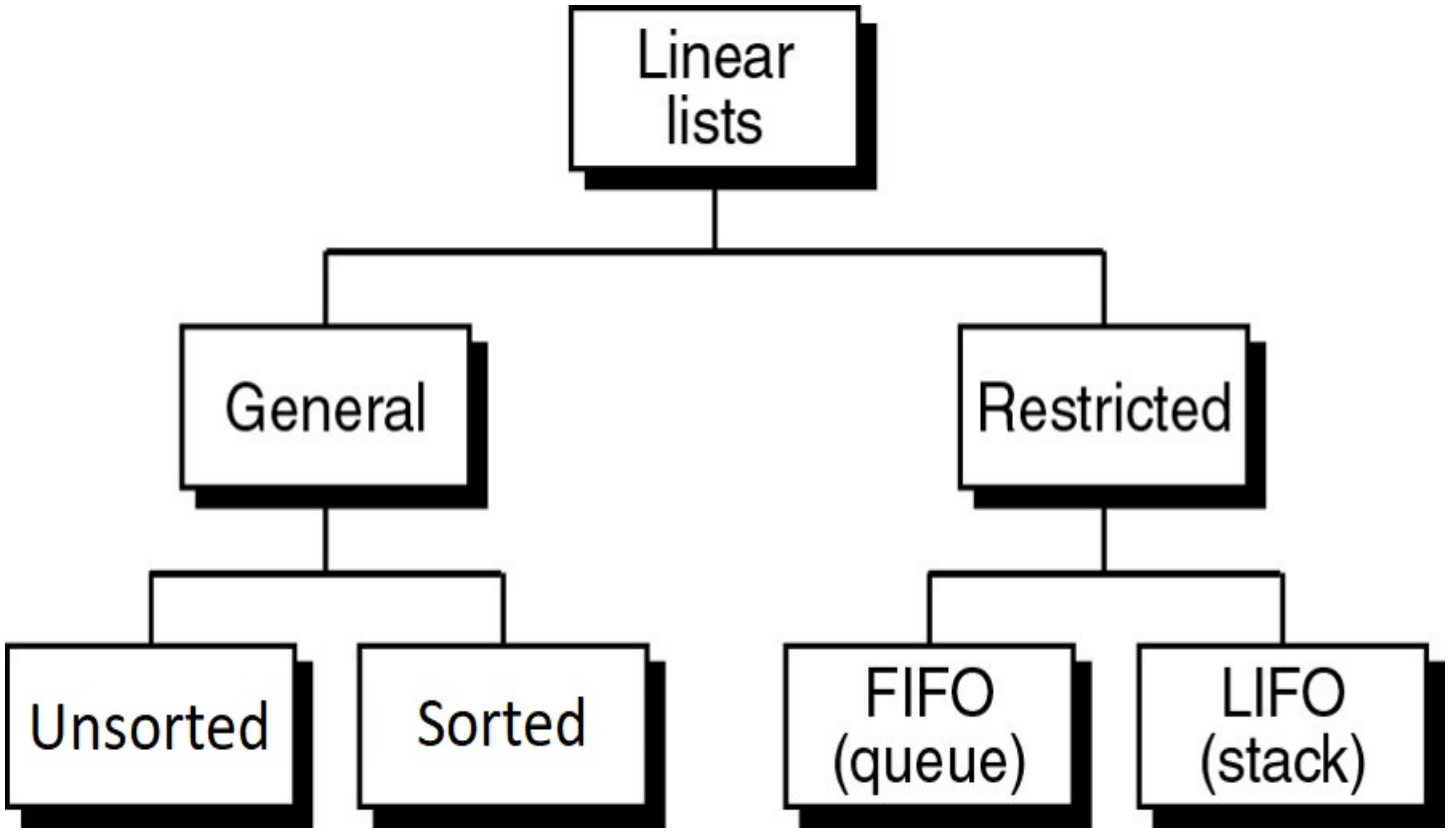
Linear list concepts

Definition

A linear list is a finite, ordered sequence of data items known as elements. "Ordered" in this definition means that each element has a position in the list.

Luu Quang Huan, MsC

Linear list concepts



Luu Quang Huan, MsC

Linear list concepts

General list:

- No restrictions on which operation can be used on the list.
- No restrictions on where data can be inserted/deleted.
- Unsorted list: data are not arranged in particular order.
- Sorted list: data are arranged according to a key.

Luu Quang Huan, MsC

Linear list concepts

Restricted list:

- Only some operations can be used on the list.
- Data can be inserted/deleted only at the ends of the list.
- Queue: FIFO (First-In-First-Out).
- Stack: LIFO (Last-In-First-Out).

Luu Quang Huan, MsC

List ADT

Definition

A list of elements of type T is a finite, ordered sequence of elements of T.

Basic concepts:

- A list is empty when it contains no elements.
- The number of elements currently stored is called the length (size) of the list.
- The beginning of the list is called the head, the end of the list is called the tail.

Luu Quang Huan, MsC

List ADT

Basic operations:

- Construct a list, leaving it empty.
- Insert an element.
- Remove an element.
- Search an element.
- Retrieve an element.
- Traverse the list, performing a given operation on each element.

Luu Quang Huan, MsC

List ADT

Extended operations:

- Determine whether the list is empty or not.
- Determine whether the list is full or not.
- Find the size of the list.
- Clear the list to make it empty.
- Replace an element with another element.
- Merge two ordered list.
- Append an unordered list to another.

Luu Quang Huan, MsC

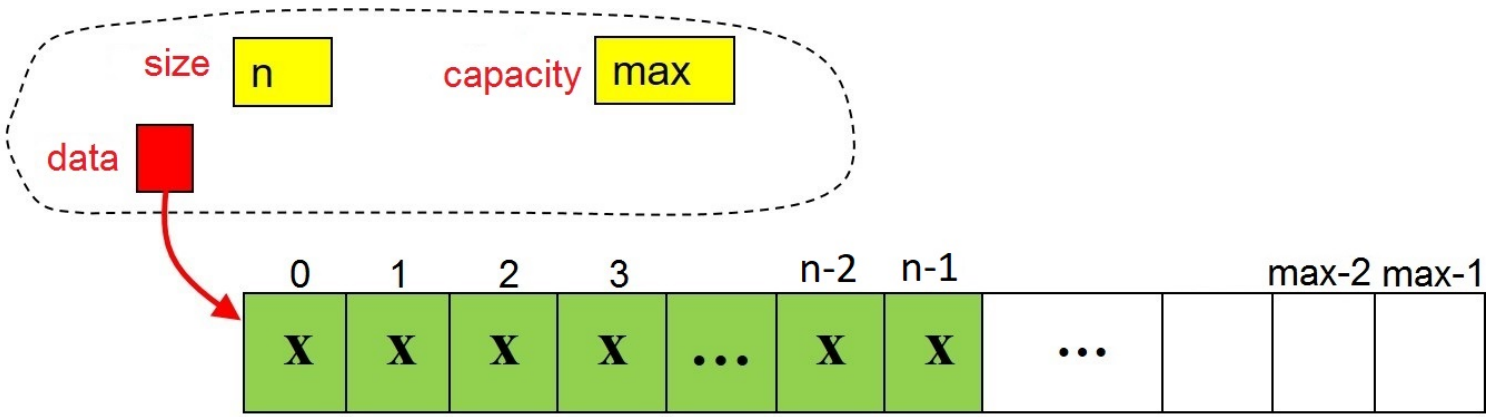
List ADT: Implementation in C++

```
template <class T>
class IList
{
public:
    virtual void add(T e) = 0;
    virtual void add(int index, T e) = 0;
    virtual T removeAt(int index) = 0;
    virtual bool removeItem(T item) = 0;
    virtual bool empty() = 0;
    virtual int size() = 0;
    virtual void clear() = 0;
    virtual T get(int index) = 0;
    virtual void set(int index, T e) = 0;
    virtual int indexOf(T item) = 0;
    virtual bool contains(T item) = 0;
    virtual string toString() = 0;
};
```


Luu Quang Huan, MsC

Array implementation

Dynamically Allocated Array




```
List // Contiguous Implementation of List
  // number of used elements (mandatory)
  count <integer>

  // (Dynamically Allocated Array)
  data <Array List of <DataType> >

  capacity <integer>
End List
```


Luu Quang Huan, MsC

Array List: Implementation in C++ with Integer

```
class IntArrayList : public IList<int> {
protected:
    int *data;
    int capacity;
    int count;
public:
    IntArrayList();
    virtual ~IntArrayList();
    virtual void add(int element);
    virtual void add(int index, int element);
    virtual int removeAt(int index);
    virtual bool removeItem(int item);
    virtual bool empty();
    virtual int size();
    virtual void clear();
    virtual int get(int index);
    virtual void set(int index, int element);
    virtual int indexOf(int item);
    virtual bool contains(int item);
    virtual string toString();
    virtual void dump();
};
```


Luu Quang Huan, MsC

Array List: Implementation in C++ with Integer

```
class IntArrayList : public IList<int>
{
// ...

private:
    void checkIndex(int index);
    void ensureCapacity(int capacity);
};
```


Luu Quang Huan, MsC

Array List: Implementation in C++ with Integer

```
IntArrayList::IntArrayList()
{
    this->capacity = 10;
    this->data = new int[this->capacity];
    this->count = 0;
}

IntArrayList::~IntArrayList()
{
    delete[] this->data;
}
```


Luu Quang Huan, MsC

Array List: Implementation in C++ with Integer

```
void IntArrayList::add(int element)
{
    this->ensureCapacity(this->count + 1);

    this->data[this->count] = element;
    this->count++;
}

void IntArrayList::add(int index, int element)
{
    this->checkIndex(index);
    this->ensureCapacity(this->count + 1);

    int moveCount = this->count - index;
    if (moveCount > 0)
        memmove(this->data + index + 1,
                this->data + index,
                moveCount * sizeof(int));

    this->data[index] = element;
    this->count++;
}
```


Luu Quang Huan, MsC

Array List: Implementation in C++ with Integer

```
int IntArrayList::removeAt(int index) {
    this->checkIndex(index);
    int elementToRemove = this->data[index];

    int moveCount = this->count - index - 1;
    if (moveCount > 0)
        memmove(this->data + index,
                this->data + index + 1,
                sizeof(int) * moveCount);

    this->count--;
    return elementToRemove;
}

bool IntArrayList::removeItem(int element) {
    for (int index = 0; index < this->count; index++) {
        if (this->data[index] == element) {
            this->removeAt(index);
            return true;
        }
    }
    return false;
}
```


Luu Quang Huan, MsC

Array List: Implementation in C++ with Integer

```
bool IntArrayList::empty()
{
    return this->count == 0;
}

int IntArrayList::size()
{
    return this->count;
}

void IntArrayList::clear()
{
    delete[] this->data;

    this->capacity = 10;
    this->data = new int[this->capacity];
    this->count = 0;
}
```


Luu Quang Huan, MsC

Array List: Implementation in C++ with Integer

```
int IntArrayList::get(int index) {
    this->checkIndex(index);

    return this->data[index];
}

void IntArrayList::set(int index, int element) {
    this->checkIndex(index);

    this->data[index] = element;
}

int IntArrayList::indexOf(int element) {
    for (int index = 0; index < this->count; index++) {
        if (this->data[index] == element) {
            return index;
        }
    }

    return -1;
}

bool IntArrayList::contains(int element) {
    return this->indexOf(element) != -1;
}
```


Luu Quang Huan, MsC

Array List: Implementation in C++ with Integer

```
string IntArrayList::toString() {
    stringstream ss;
    ss << "[";
    for (int index = 0; index < count - 1; index++) {
        ss << data[index] << ", ";
    }
    if (count > 0) ss << data[count - 1] << "]";
    else ss << "]";

    return ss.str();
}

void IntArrayList::dump() {
    string line(50, '=');
    cout << line << endl;
    cout << "Integer list's information:" << endl;
    cout << "- Capacity: " << this->capacity << endl;
    cout << "- Size: " << this->count << endl;
    cout << "- Data: " << this->toString() << endl;
    cout << line << endl;
}
```


Luu Quang Huan, MsC

Array List: Implementation in C++ with Integer

```
void IntArrayList::checkIndex(int index) {
    if (index < 0 || index >= this->count)
        throw std::out_of_range(
                    "Index is out of range");
}

void IntArrayList::ensureCapacity(int capacity) {
    if (capacity > this->capacity) {
        int newCapacity = this->capacity * 3 / 2;
        int *newData = new int[newCapacity];
        memmove(newData, this->data,
                         this->count * sizeof(int));
        this->capacity = newCapacity;
        delete[] this->data;

        this->data = newData;
    }
}
```


Contiguous Implementation of List

In processing a contiguous list with n elements:

- Insert and Remove operate in time approximately proportional to n (require physical shifting).
- Clear, Empty, Full, Size, Replace, and Retrieve in constant time.

Luu Quang Huan, MsC

Singly linked list

Linked List

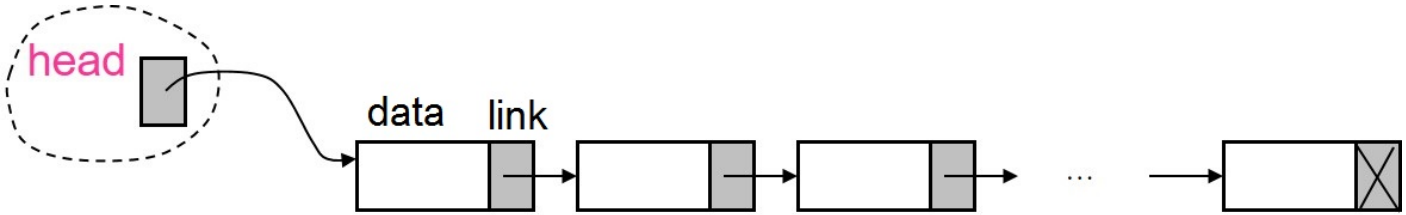


Figure: Singly Linked List

```
list // Linked Implementation of List
  head <pointer>
  tail <pointer>  // (optional)
  count <integer> // number of elements (optional)
end list
```

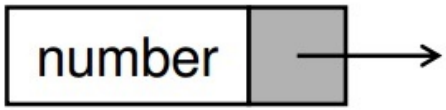

Luu Quang Huan, MsC

Nodes

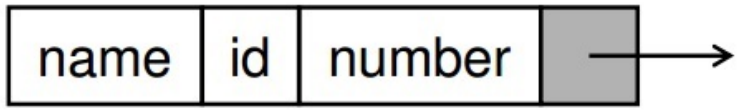
The elements in a linked list are called nodes.
A node in a linked list is a structure that has at least two fields:

- the data,
- the address of the next node.

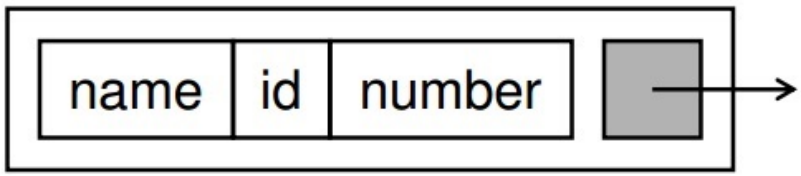
A node with one data field



A node with three data fields



A node with one structured data field



Luu Quang Huan, MsC

Nodes

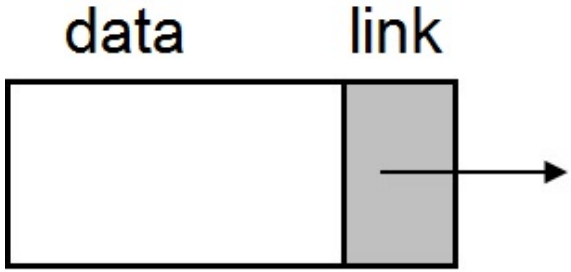


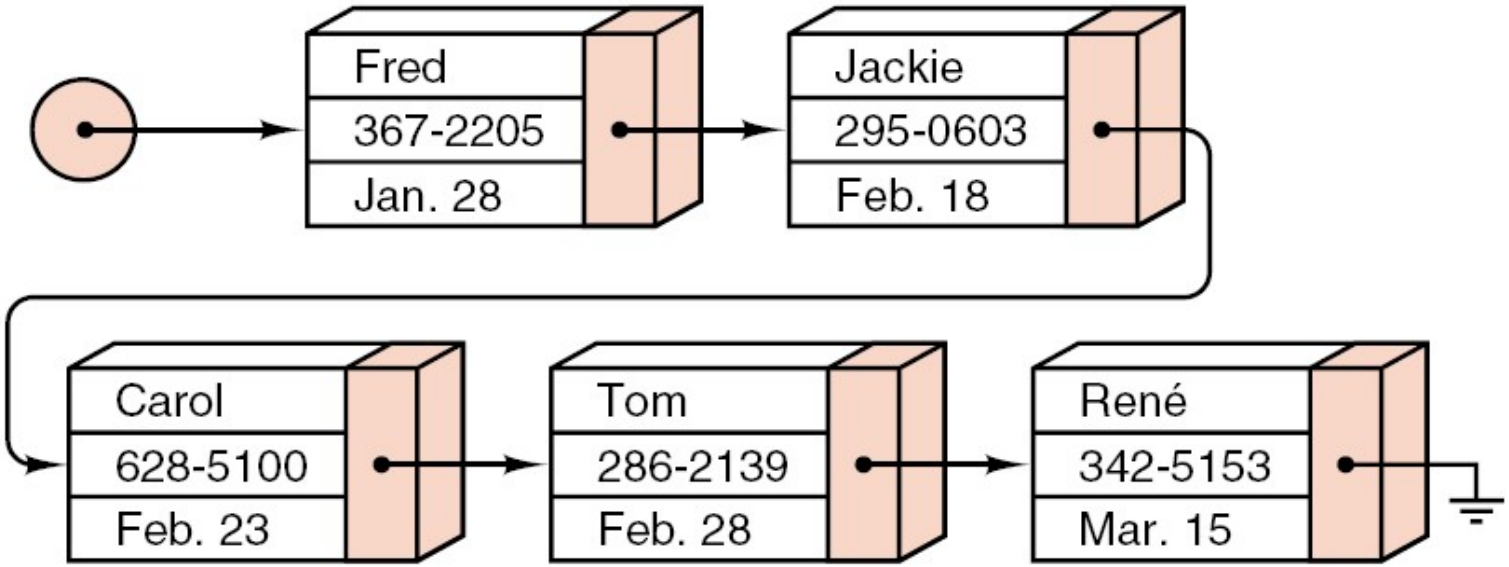
Figure: Linked list node structure

```
node
   data <dataType>
   link <pointer>
end node
```

```
// General dataType:
dataType
   key <keyType>
   field1 <...>
   field2 <...>
   ...
   fieldn <...>
end dataType
```


Luu Quang Huan, MsC

Example



Luu Quang Huan, MsC

Example

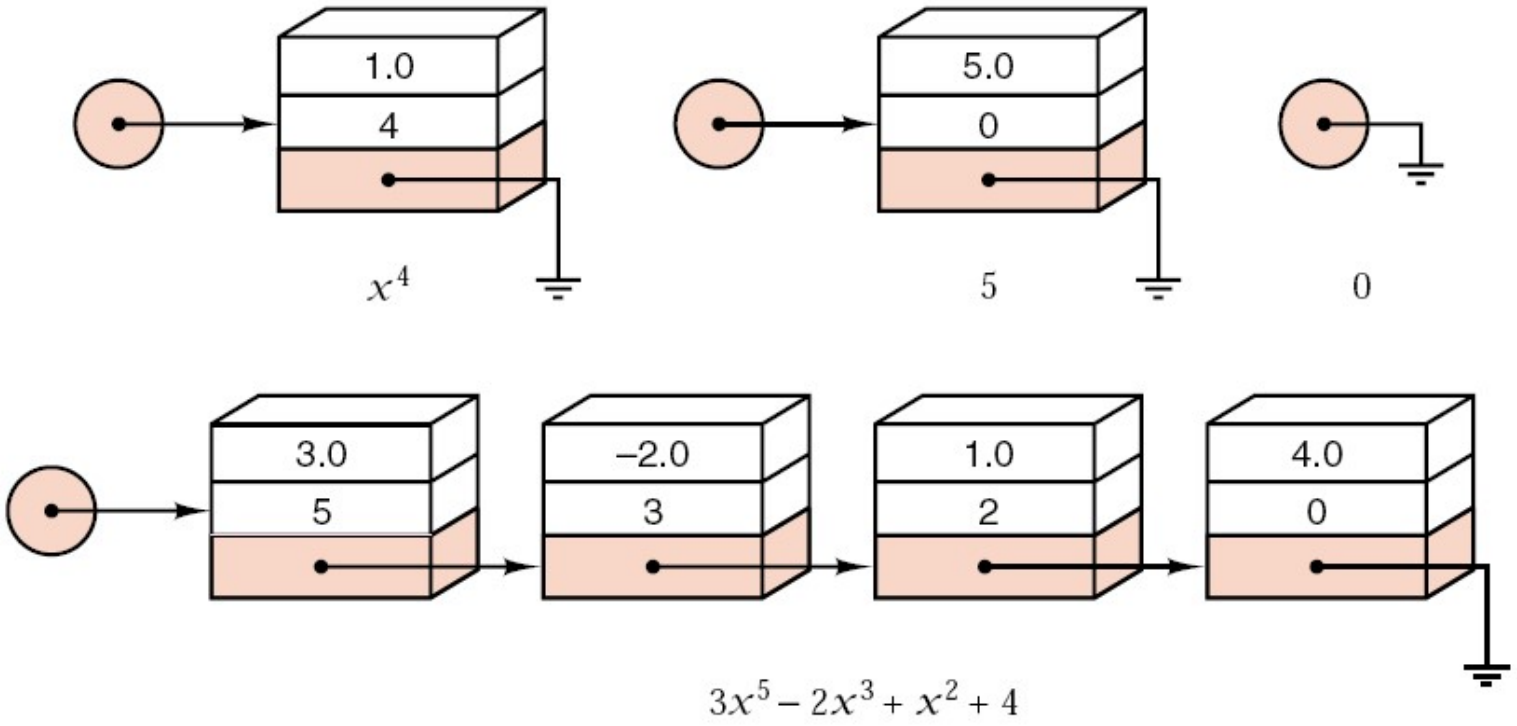


Figure: List representing polynomial

Luu Quang Huan, MsC

Implementation in C++

Example

```
node
  data <dataType>
  next <pointer>
end node
```

```
struct Node {
    int data;
    Node *next;
};
```


Luu Quang Huan, MsC

Implementation in C++ with `struct`

Example

```
struct Node {
    int data;
    Node *next;
};
```

```
struct Node {
    float data;
    Node *next;
};
```

```
template <class T>
struct Node {
    T data;
    Node<T> *next;
};
```


Luu Quang Huan, MsC

Node implementation in C++ with nested class

```
class IntSLinkedList : public IList<int> {
public:
    class Node; // Forward declaration

protected:
    Node* head;
    Node* tail;
    int count;

public:
    IntSLinkedList() :
        head(NULL), tail(NULL), count(0) {};

public:
    class Node {
    protected:
        int data;
        Node* next;

    public:
        Node() : next(NULL) {};
        Node(int data) :
            data(data), next(NULL) {};
    };
};
```

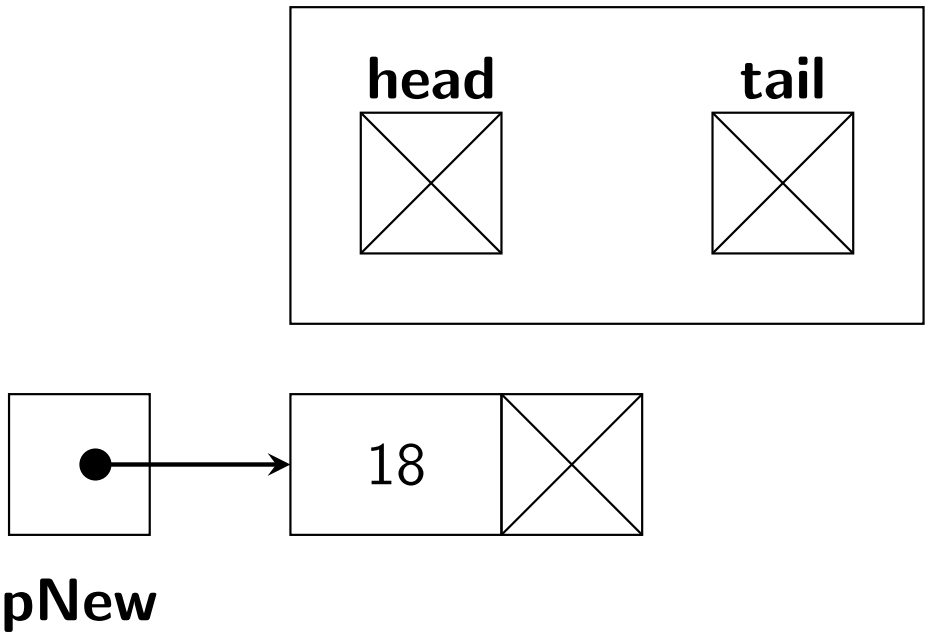

Luu Quang Huan, MsC

Linked list operations

- Create an empty linked list
- Insert an item into a linked list
- Remove an item from a linked list
- Traverse a linked list
- Destroy a linked list
- ...

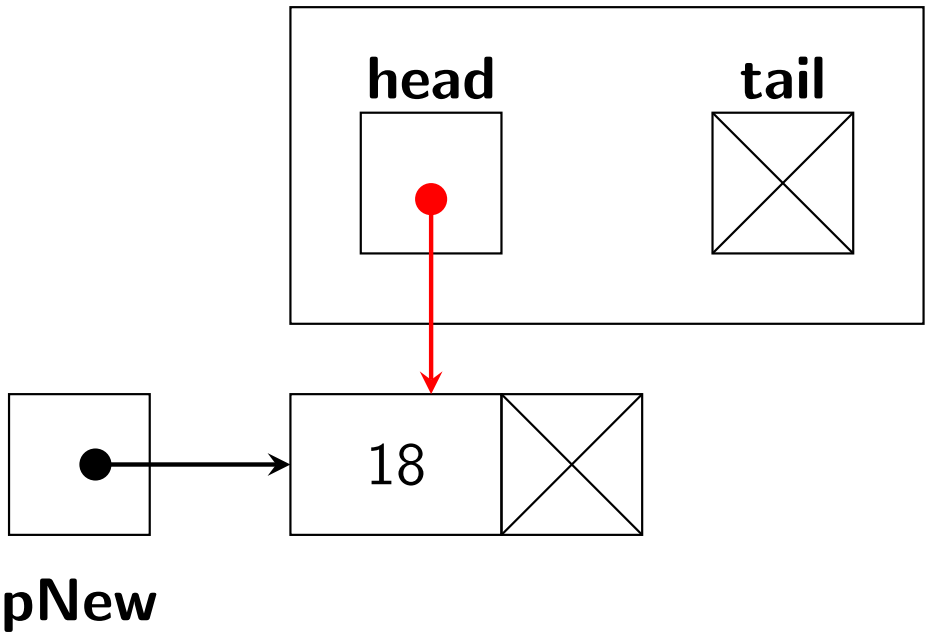
Luu Quang Huan, MsC

Insertion: Prepend to an empty list



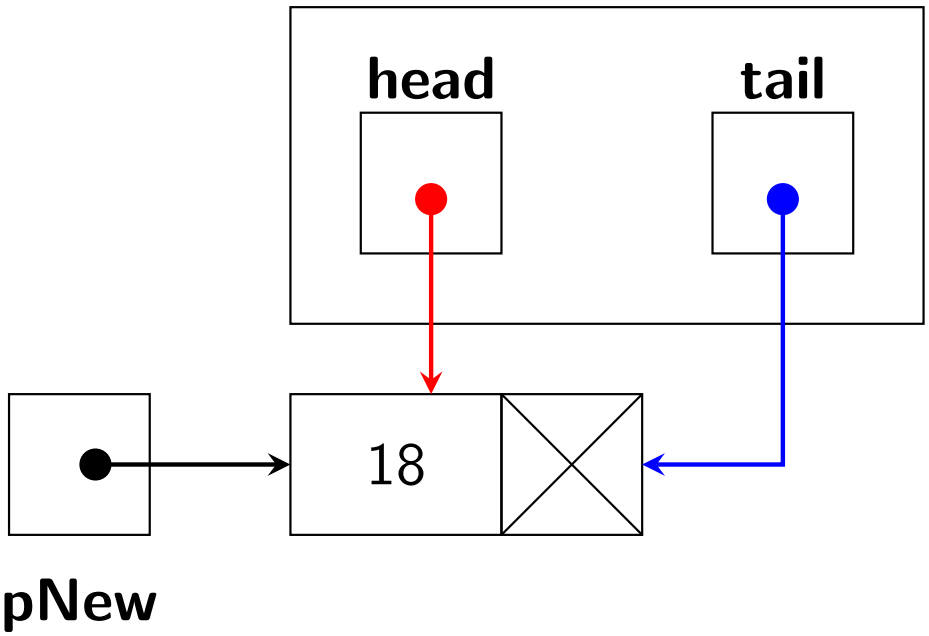
Luu Quang Huan, MsC

Insertion: Prepend to an empty list



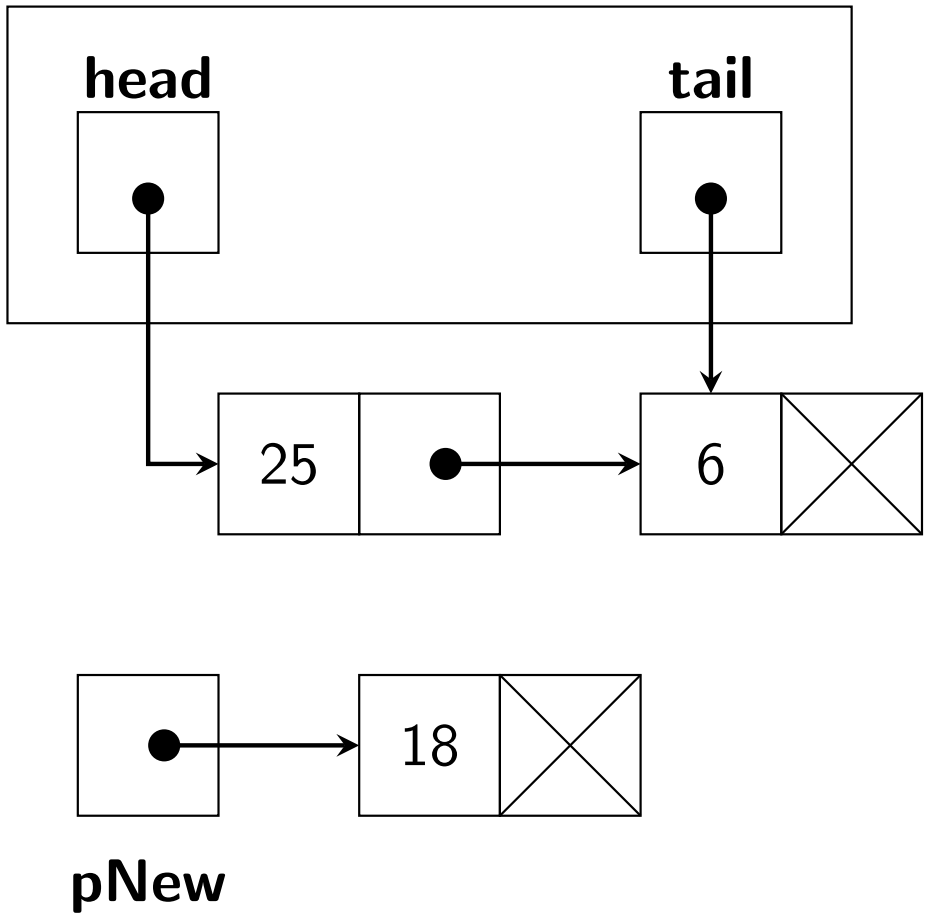
Luu Quang Huan, MsC

Insertion: Prepend to an empty list



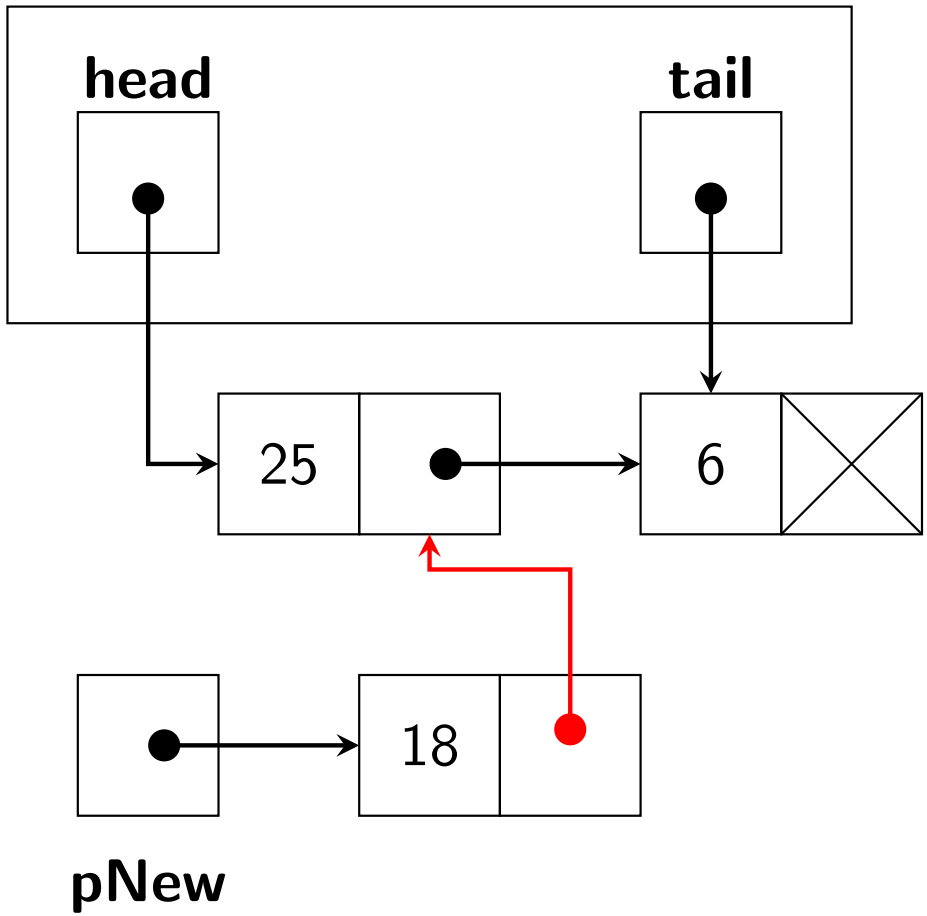
Luu Quang Huan, MsC

Insertion: Prepend to a non-empty list



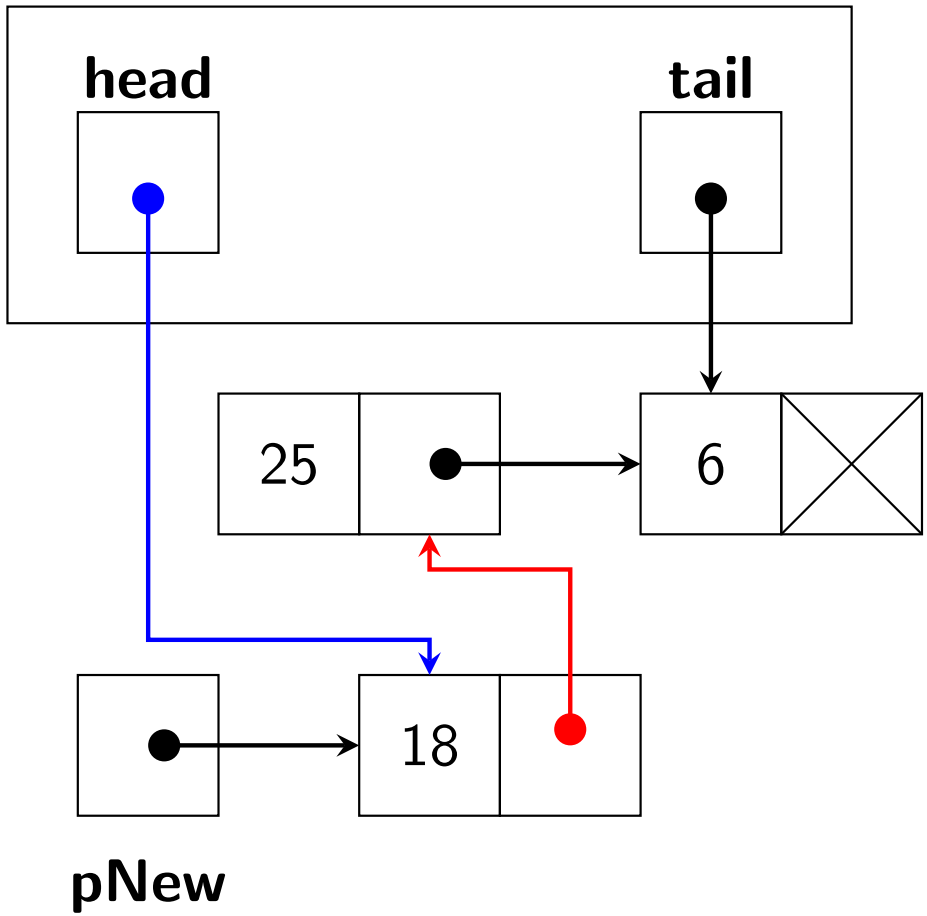
Luu Quang Huan, MsC

Insertion: Prepend to a non-empty list



Luu Quang Huan, MsC

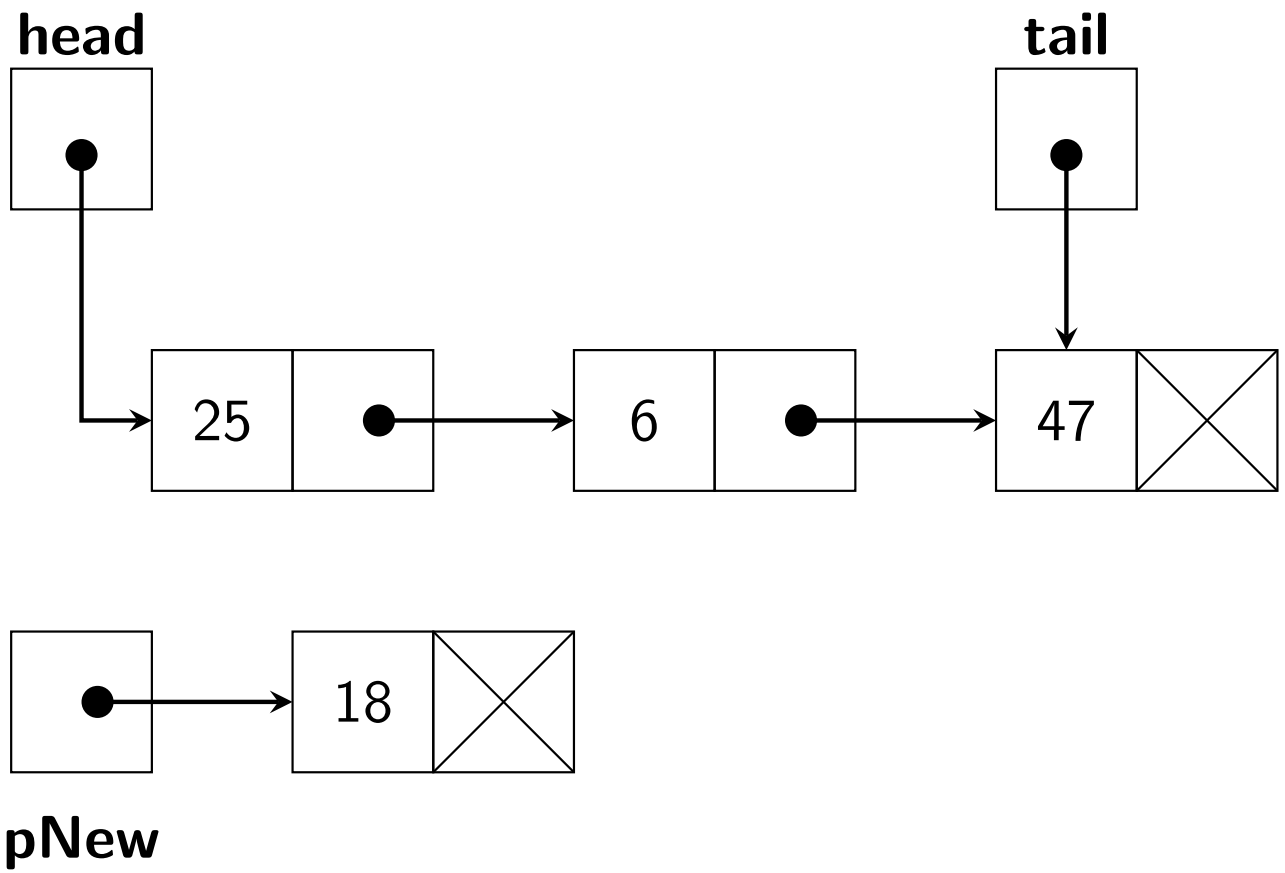
Insertion: Prepend to a non-empty list



Luu Quang Huan, MsC

Insertion: At the index i

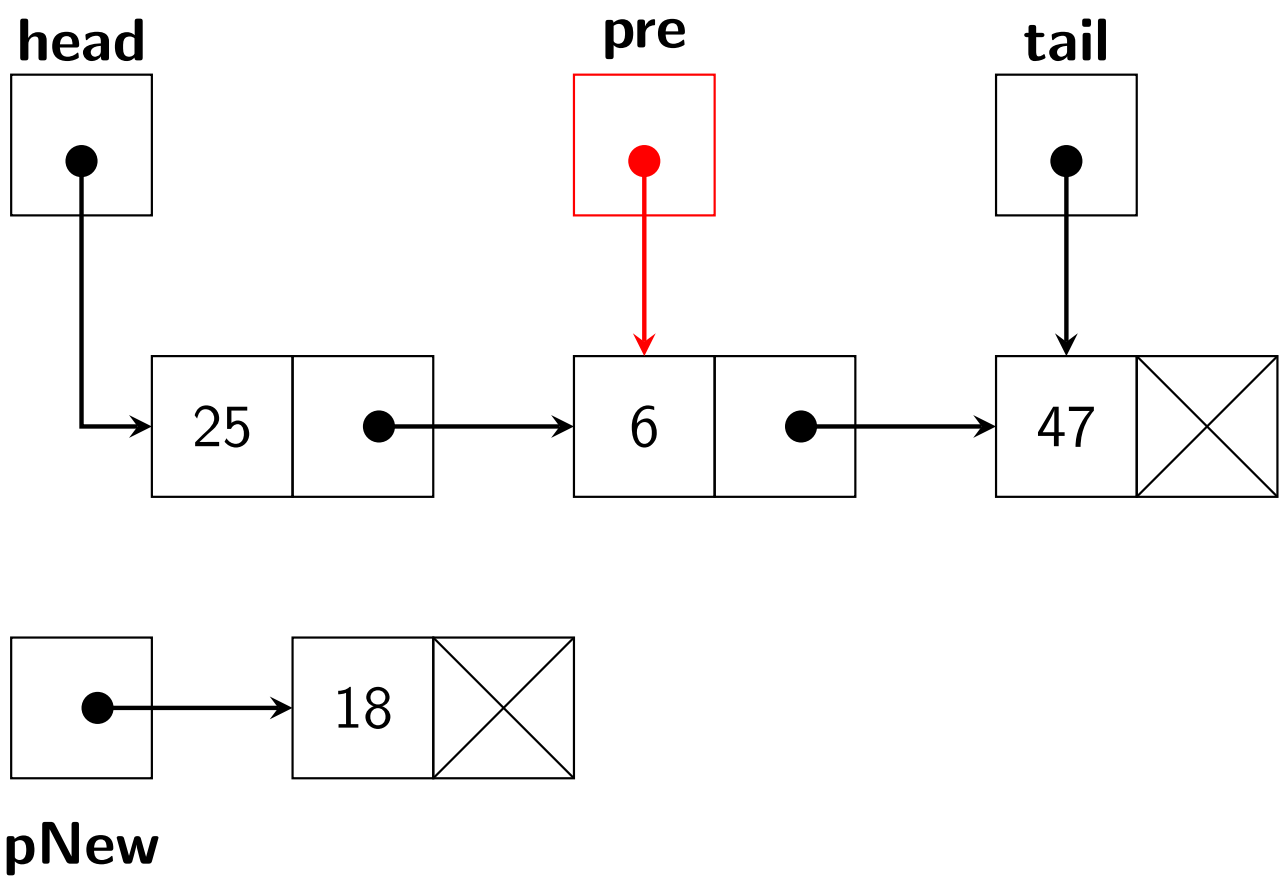
Insert **18** at index **2**.



Luu Quang Huan, MsC

Insertion: At the index i

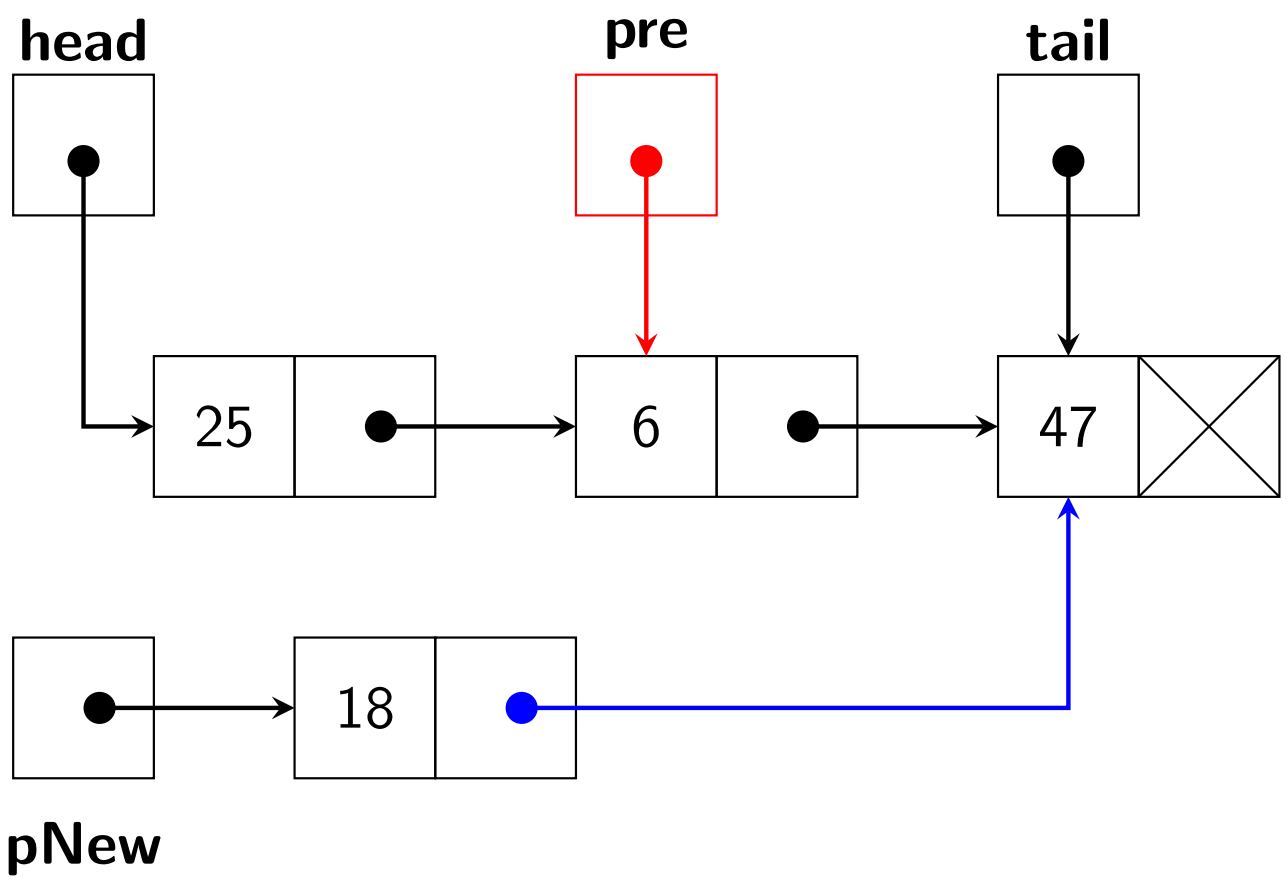
Insert **18** at index **2**.



Luu Quang Huan, MsC

Insertion: At the index i

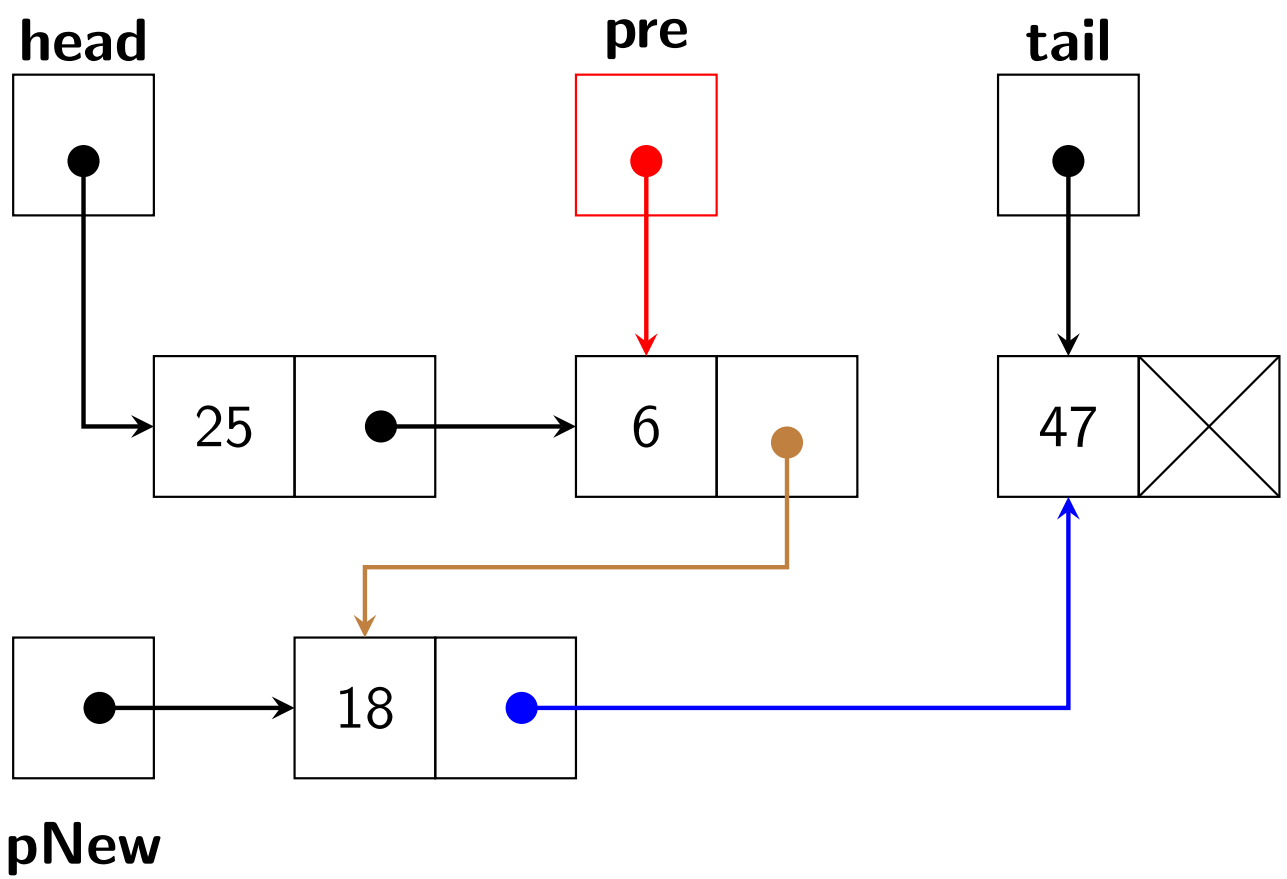
Insert **18** at index **2**.



Luu Quang Huan, MsC

Insertion: At the index i

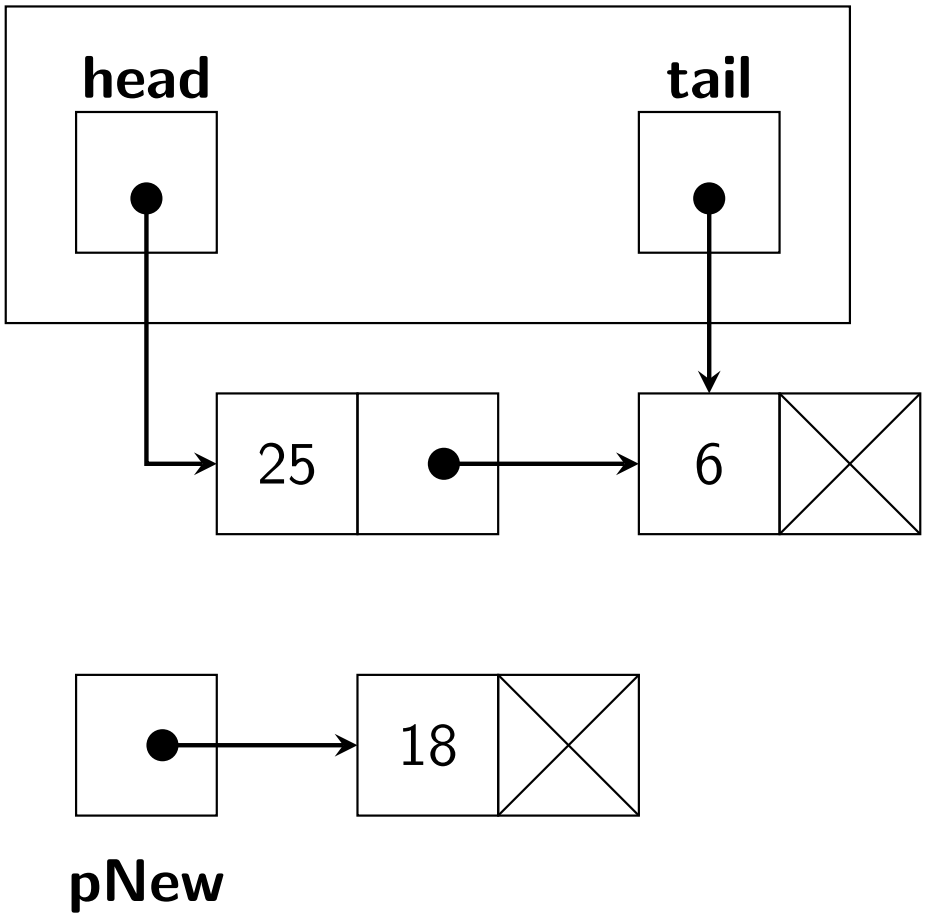
Insert **18** at index **2**.



Luu Quang Huan, MsC

Insertion: Append to the list

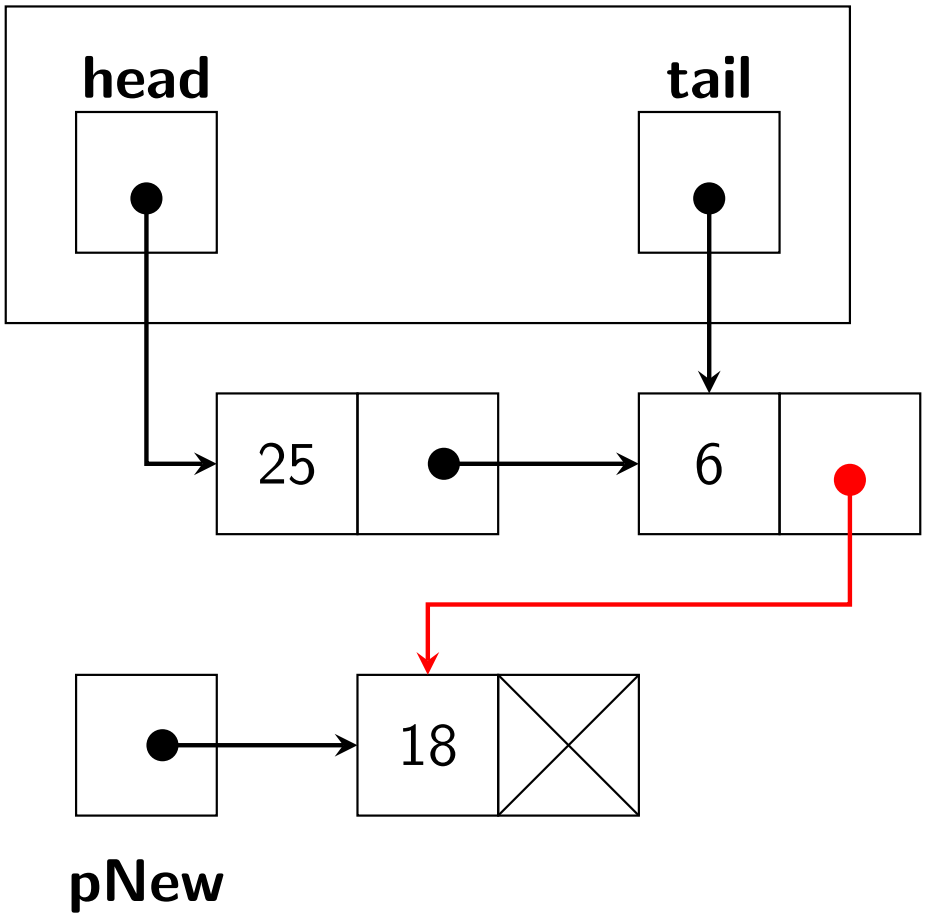
Append or insert at $i = length$.



Luu Quang Huan, MsC

Insertion: Append to the list

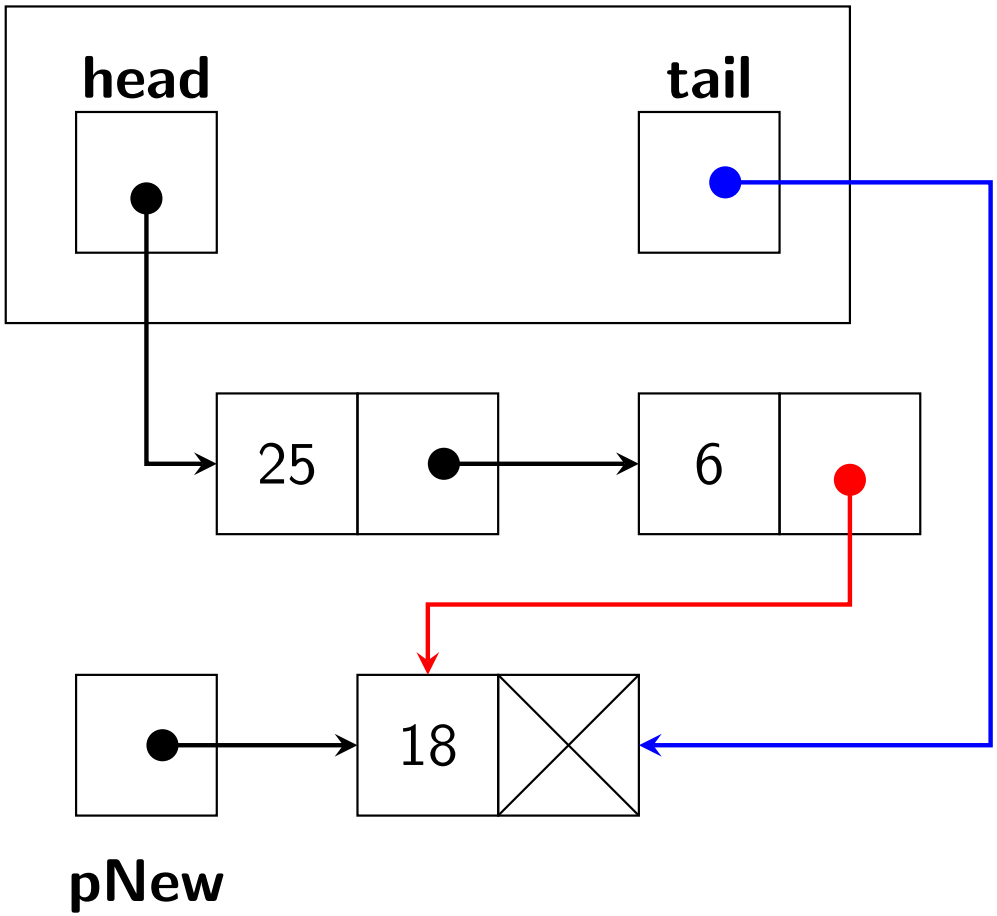
Append or insert at $i = length$.



Luu Quang Huan, MsC

Insertion: Append to the list

Append or insert at $i = length$.



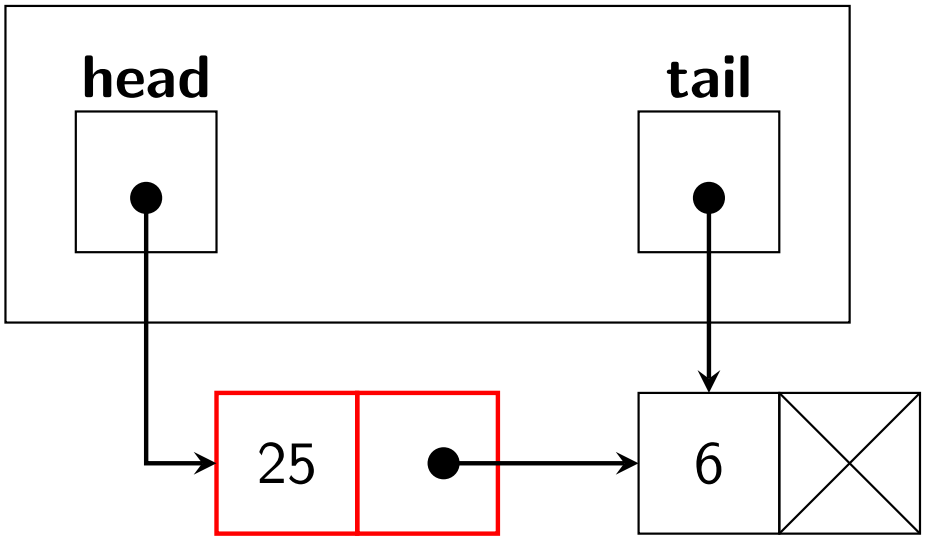
Luu Quang Huan, MsC

Insertion: Implementation in C++

```
class IntSLinkedList : public IList<int> {
// Declaration of attributes, constructor, destructor.
// Declaration of nested classes.
public:
    virtual void add(int element);
    virtual void add(int index, int element);
};
```

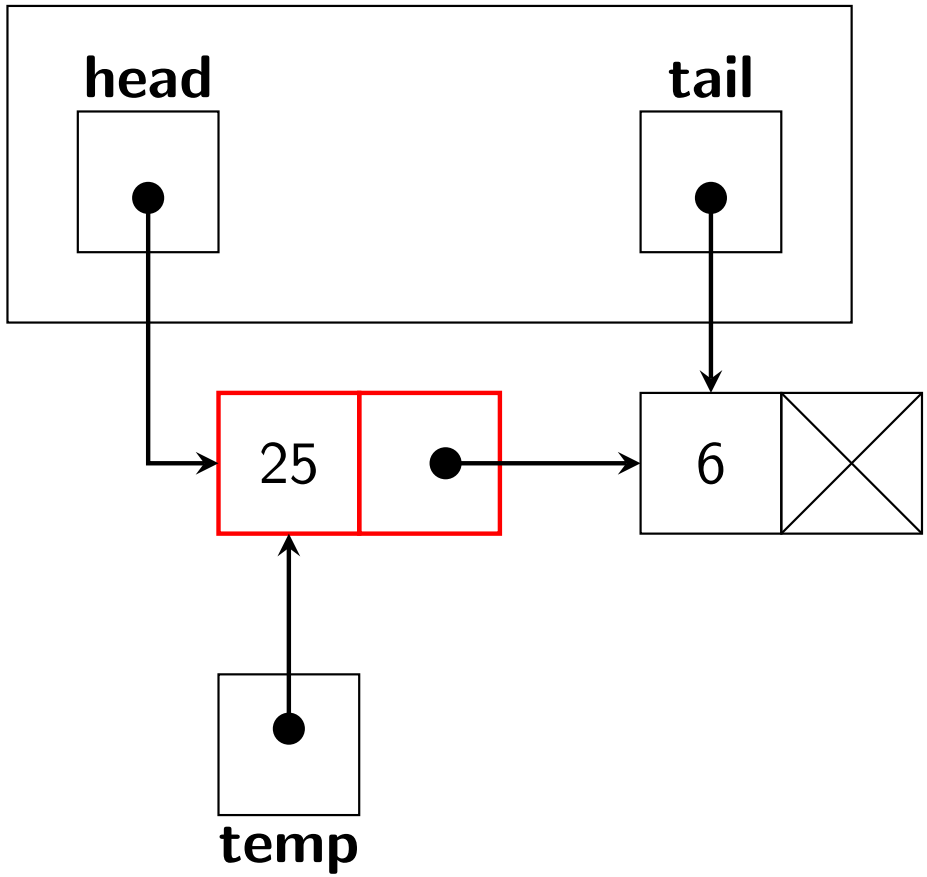

Luu Quang Huan, MsC

Removal: Delete the first element



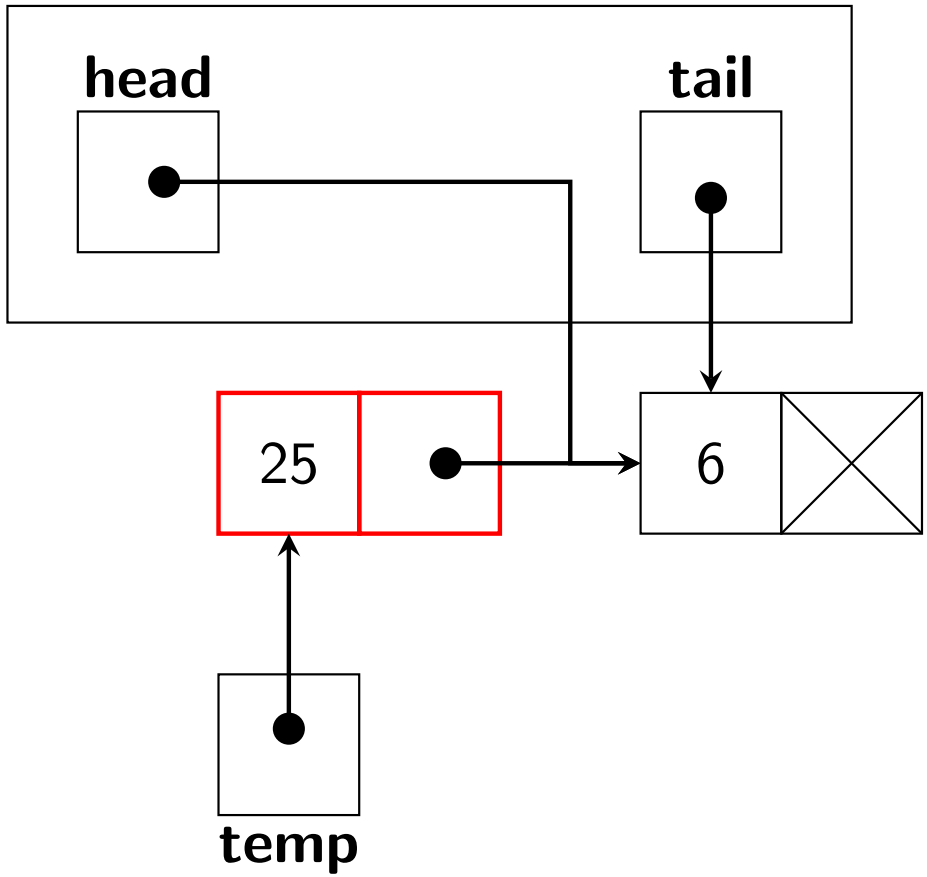
Luu Quang Huan, MsC

Removal: Delete the first element



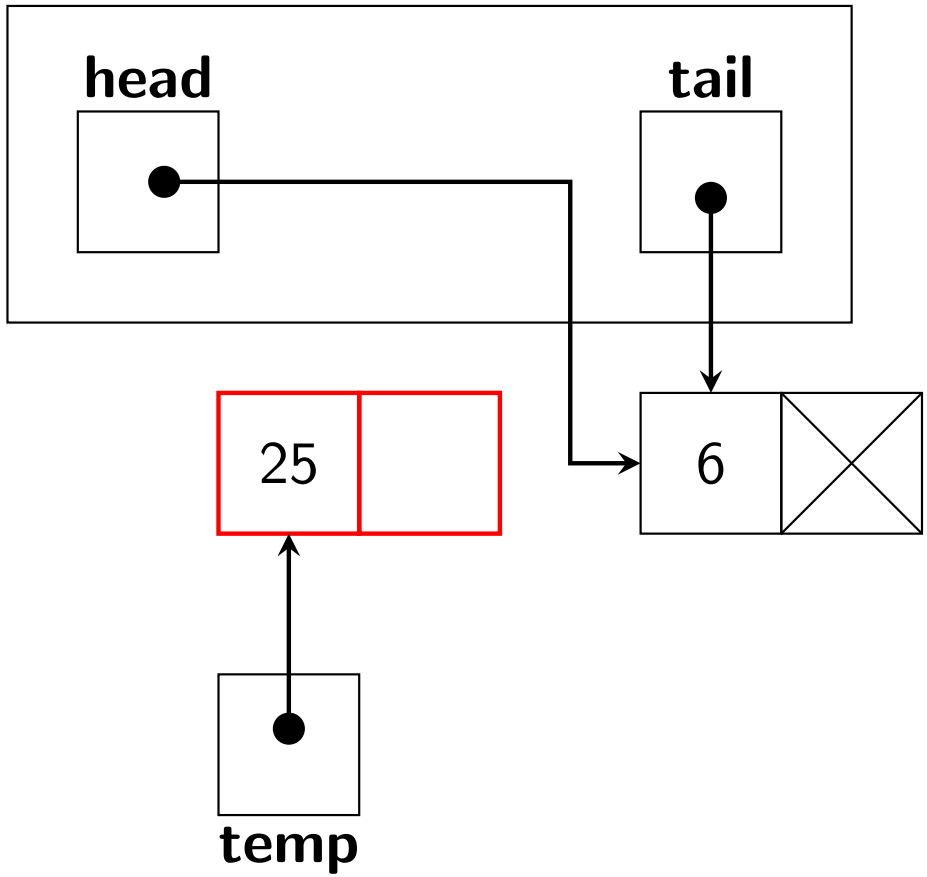
Luu Quang Huan, MsC

Removal: Delete the first element



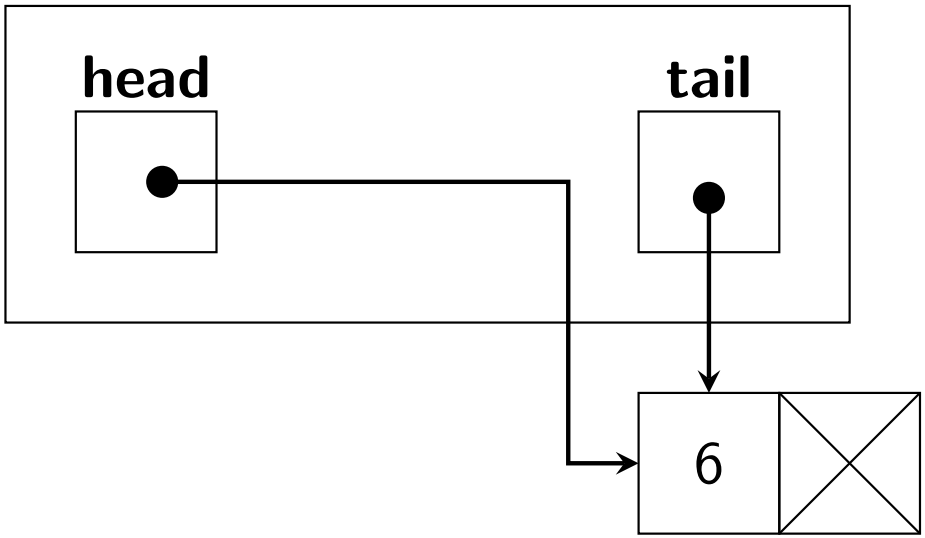
Luu Quang Huan, MsC

Removal: Delete the first element



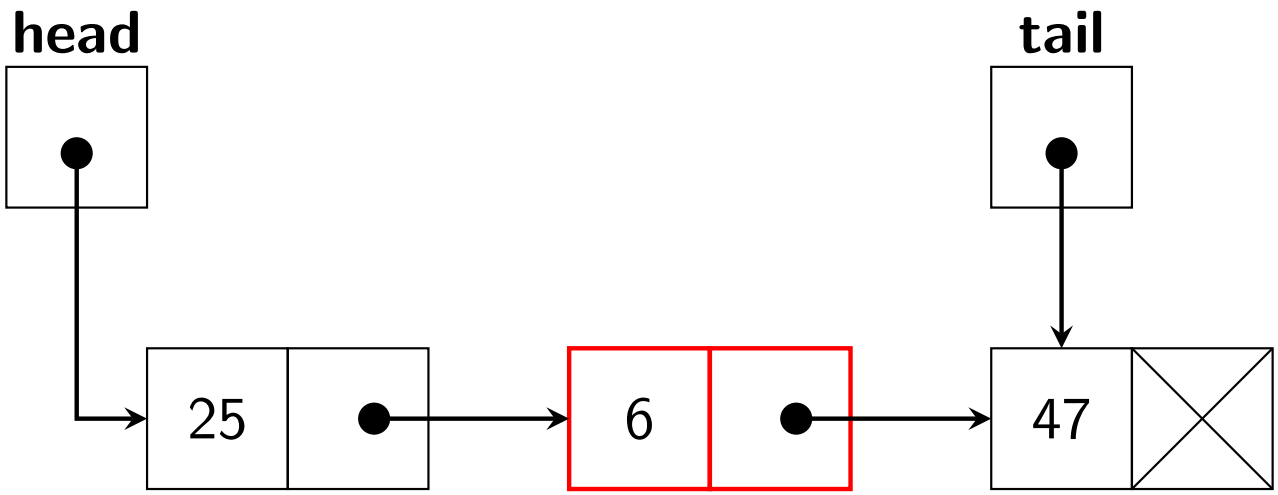
Luu Quang Huan, MsC

Removal: Delete the first element



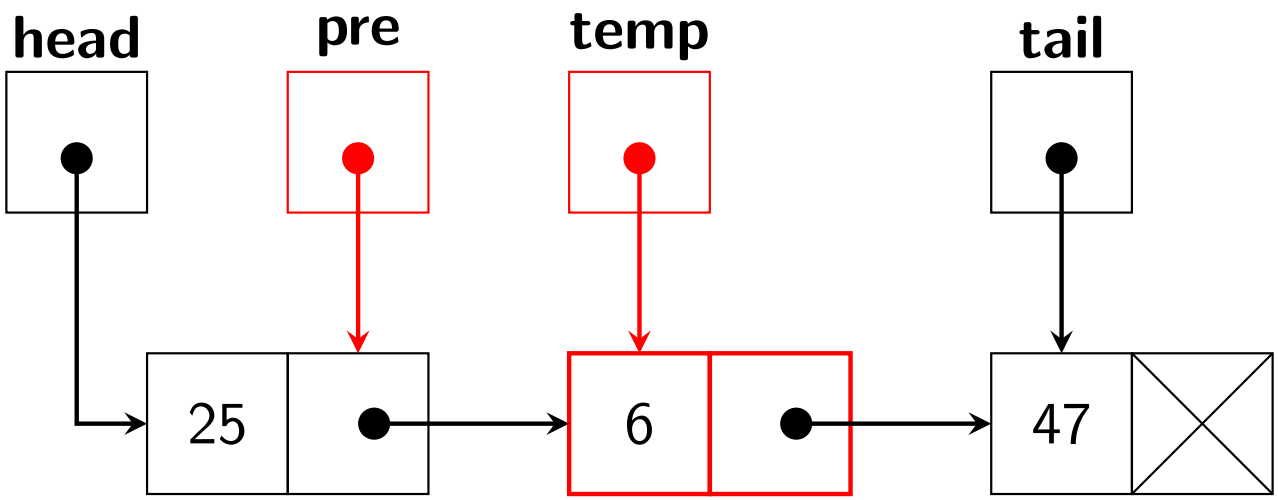
Luu Quang Huan, MsC

Removal: Delete at index i



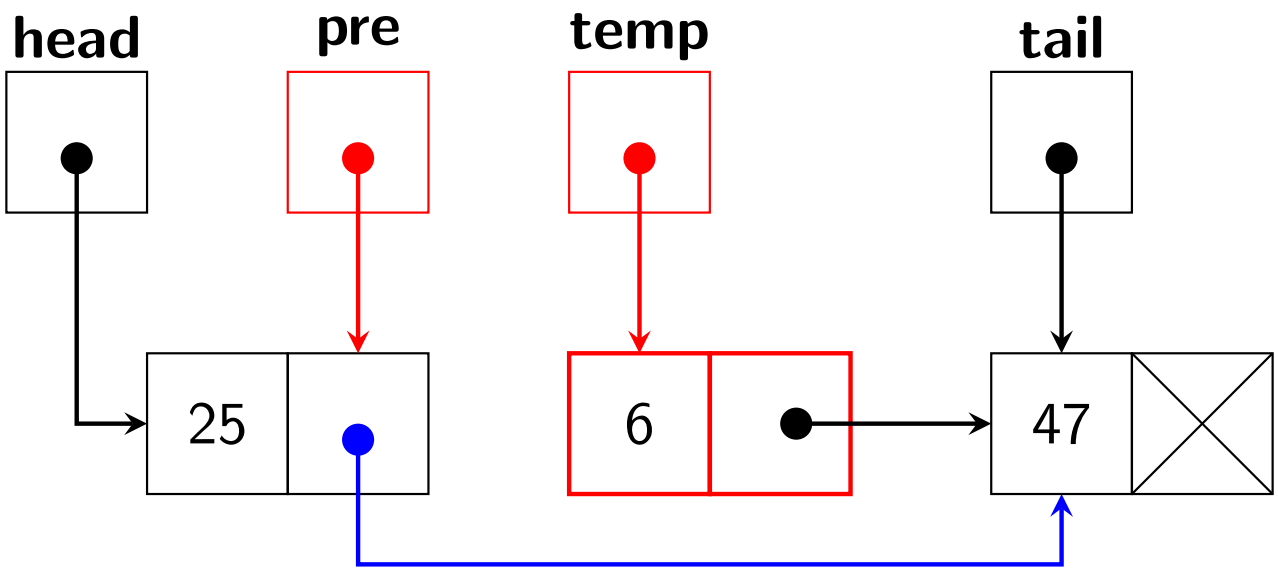
Luu Quang Huan, MsC

Removal: Delete at index i



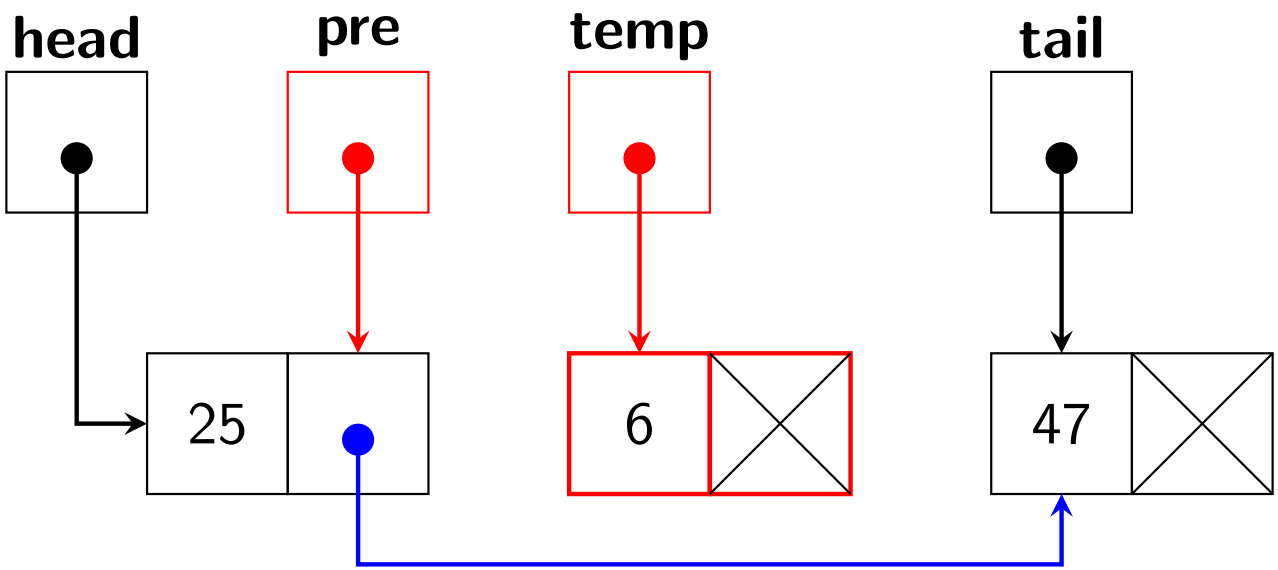
Luu Quang Huan, MsC

Removal: Delete at index i



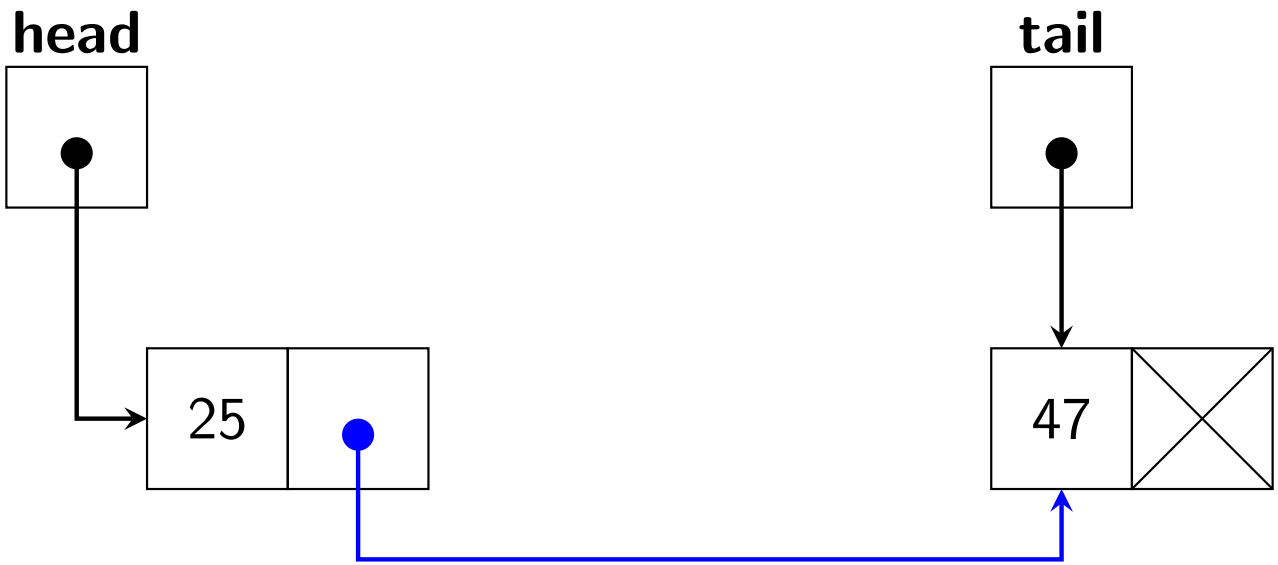
Luu Quang Huan, MsC

Removal: Delete at index i



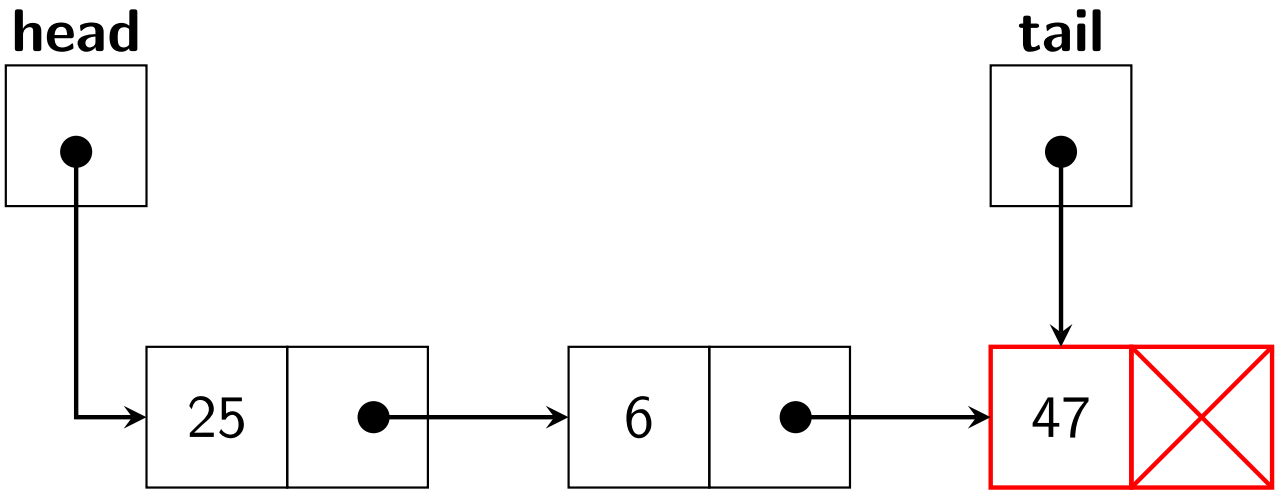
Luu Quang Huan, MsC

Removal: Delete at index i



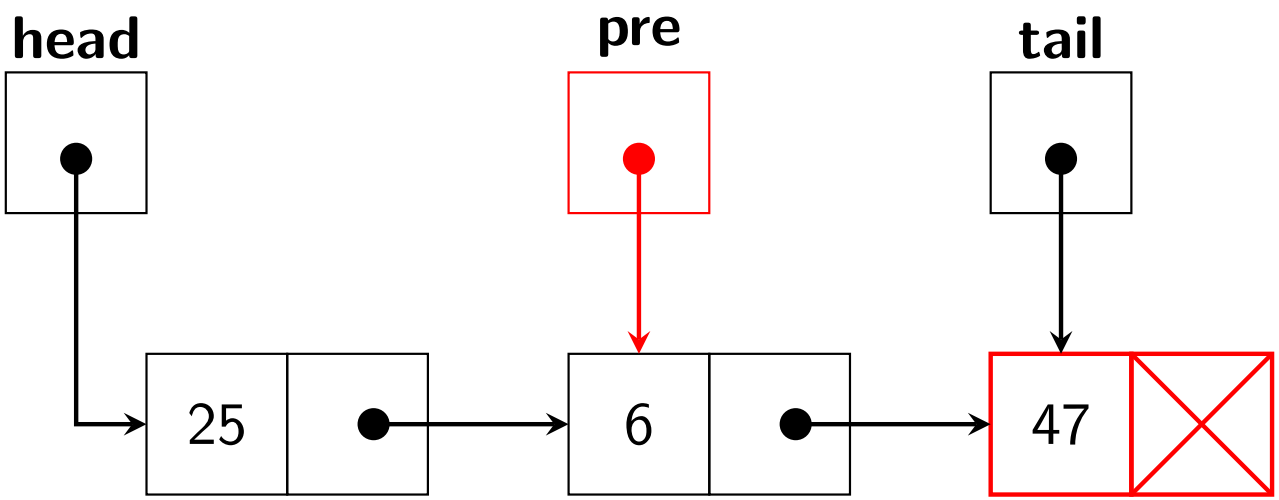
Luu Quang Huan, MsC

Removal: Delete the last element



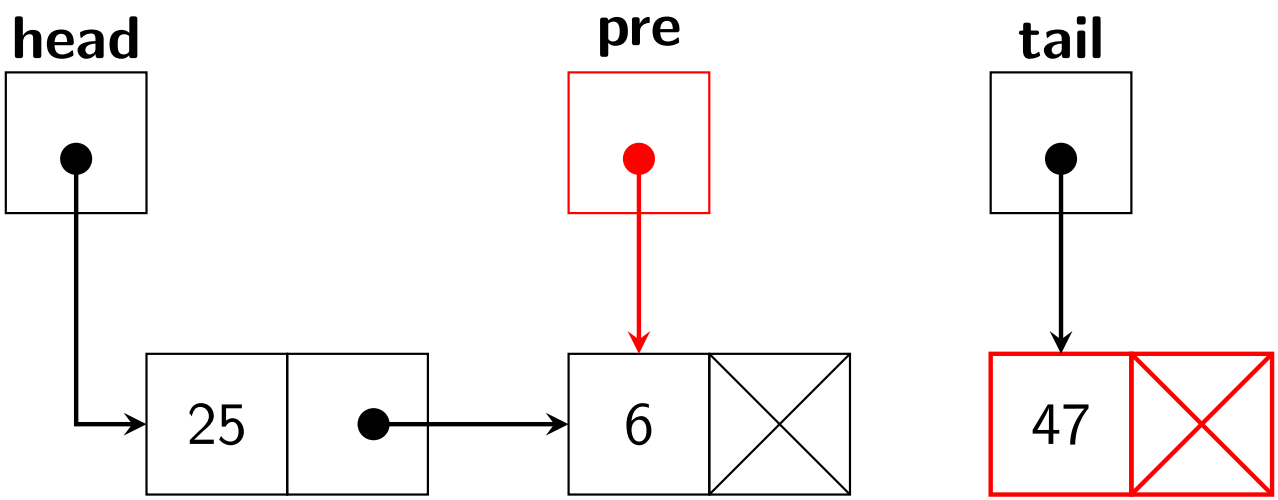
Luu Quang Huan, MsC

Removal: Delete the last element



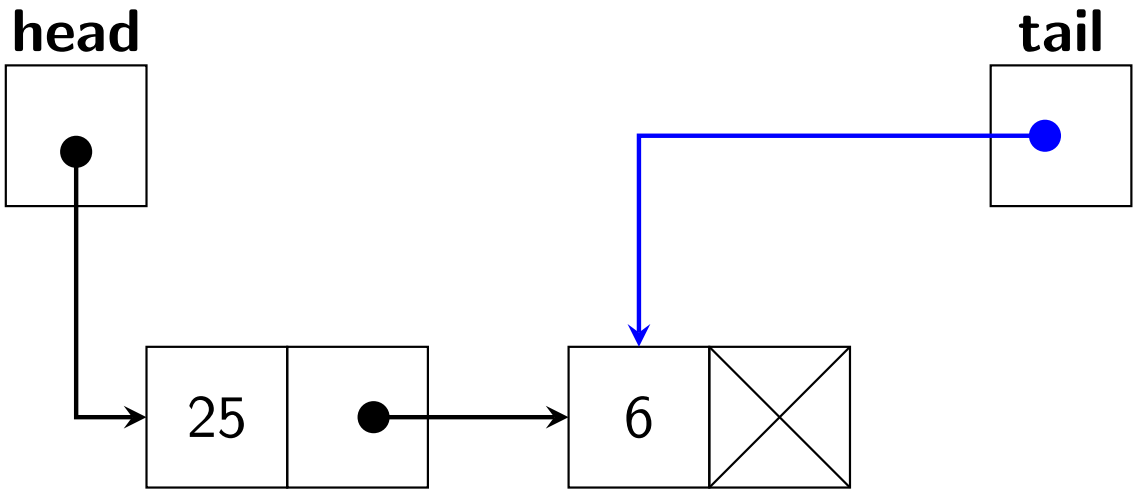
Luu Quang Huan, MsC

Removal: Delete the last element



Luu Quang Huan, MsC

Removal: Delete the last element



Luu Quang Huan, MsC

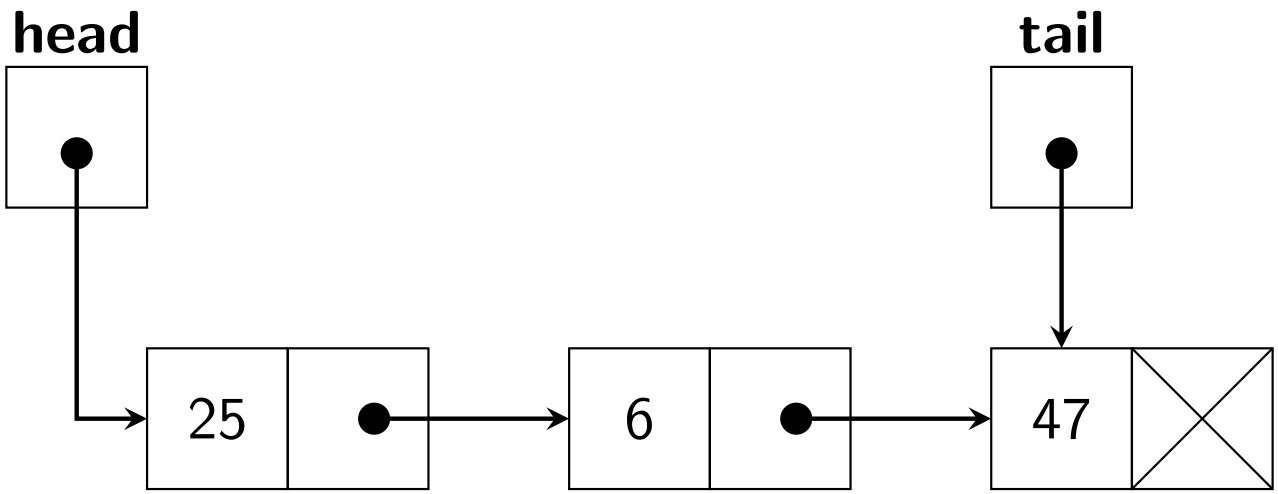
Removal: Implementation in C++

```
class IntSLinkedList : public IList<int> {
// Declaration of attributes, constructor, destructor.
// Declaration of nested classes.
public:
    virtual int removeAt(int index);
    virtual bool removeItem(int item);
};
```


Luu Quang Huan, MsC

Searching

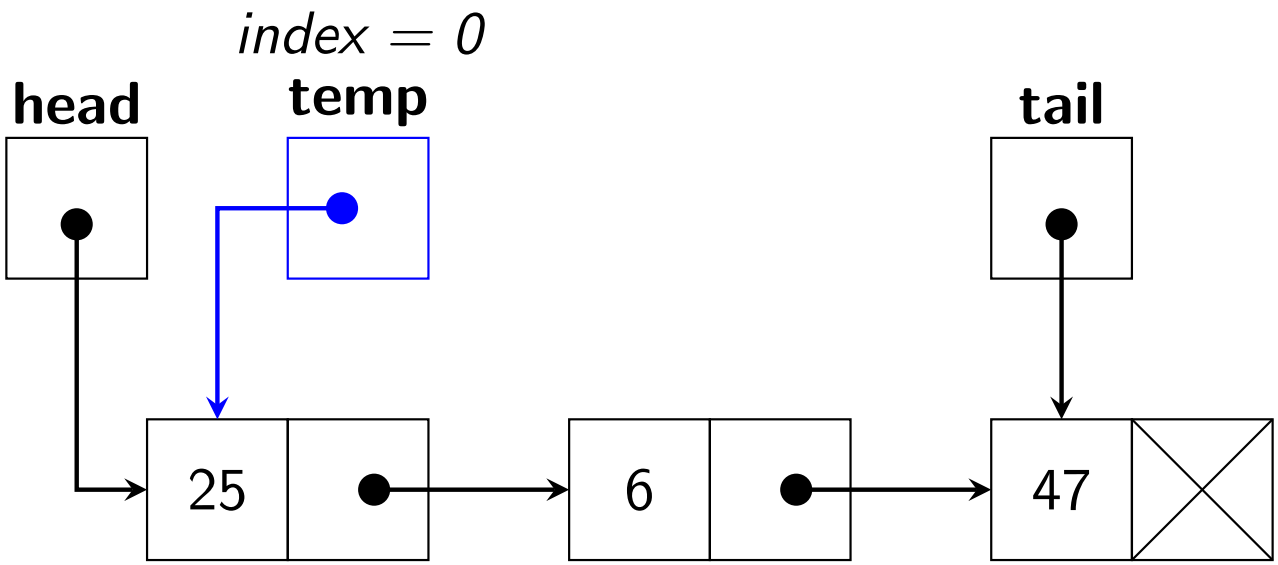
Search **element** with index **1**.



Luu Quang Huan, MsC

Searching

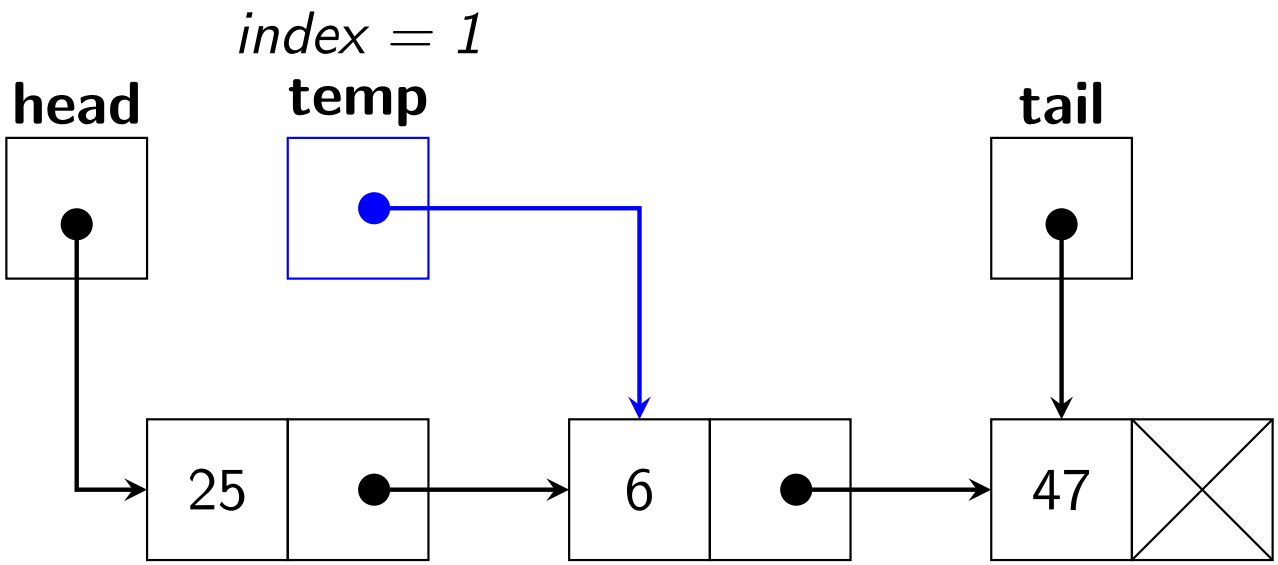
Search **element** with index **1**.



Luu Quang Huan, MsC

Searching

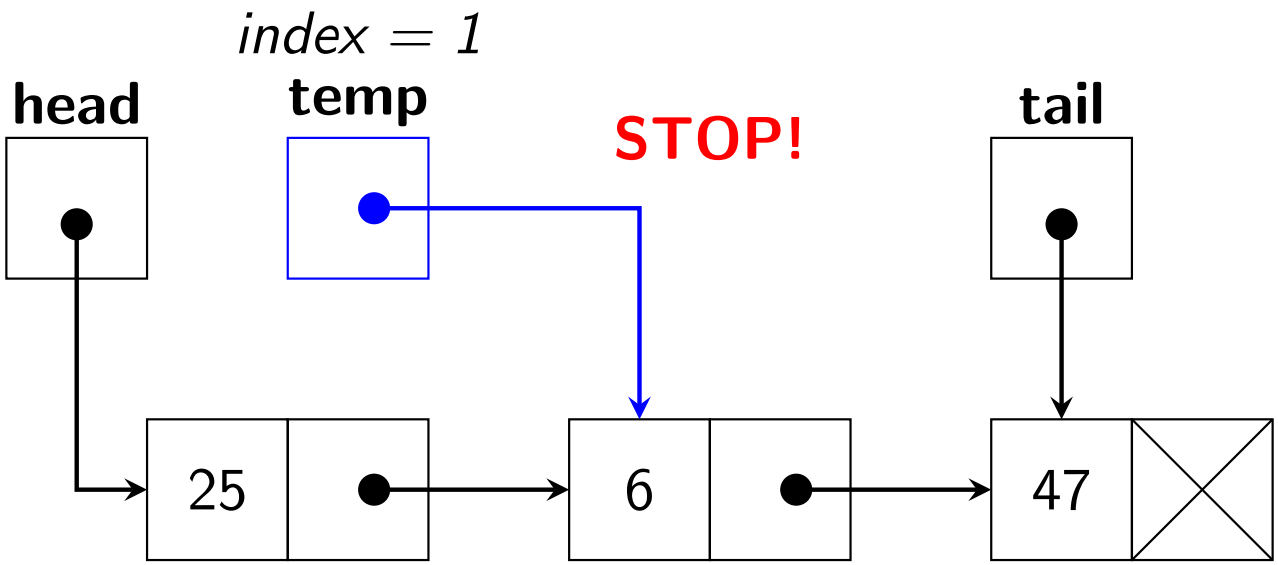
Search **element** with index **1**.



Luu Quang Huan, MsC

Searching

Search **element** with index **1**.



Luu Quang Huan, MsC

Search and other methods: Implementation in C++

```
class IntSLinkedList : public IList<int> {
// Declaration of attributes, constructor, destructor.
// Declaration of nested classes.
public:
    virtual int get(int index);
    virtual void set(int index, int element);
    virtual int indexOf(int item);
    virtual bool contains(int item);
};
```


Luu Quang Huan, MsC

Comparison of implementations of list

Luu Quang Huan, MsC

Arrays: Pros and Cons

- **Pros**: Access to an array element is fast since we can compute its location quickly.
- **Cons**:
  - If we want to insert or delete an element, we have to shift subsequent elements which slows our computation down.
  - We need a large enough block of memory to hold our array.

Luu Quang Huan, MsC

Linked Lists: Pros and Cons

- **Pros**: Inserting and deleting data does not require us to move/shift subsequent data elements.
- **Cons**: If we want to access a specific element, we need to traverse the list from the head of the list to find it which can take longer than an array access.

Luu Quang Huan, MsC

Comparison of implementations of list

- **Contiguous storage is generally preferable when**:
  - the entries are individually very small;
  - the size of the list is known when the program is written;
  - few insertions or deletions need to be made except at the end of the list; and
  - random access is important.
- **Linked storage proves superior when**:
  - the entries are large;
  - the size of the list is not known in advance; and
  - flexibility is needed in inserting, deleting, and rearranging the entries.

Luu Quang Huan, MsC

Iteration on list

Luu Quang Huan, MsC

Why iteration?

Problem on iteration

Consider a problem that you have to travel all elements in list and make change some elements like increasing elements with odd data (on integer lists).

Comparision on implementations

If only use operation GET on list:

- With array implementation: $O(n)$
- With linked list: $O(n^2)$

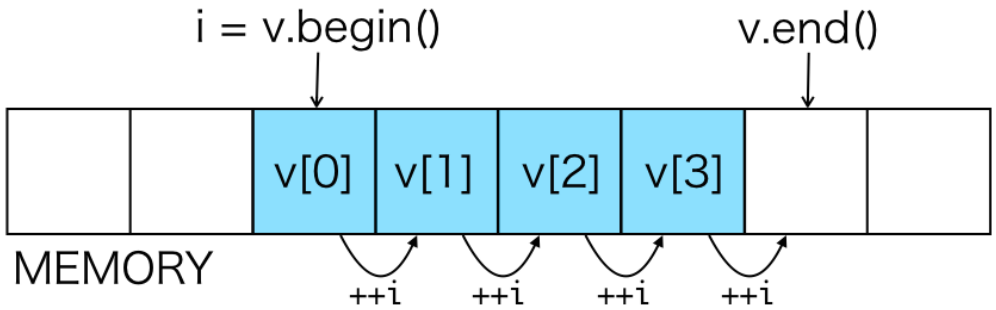
Luu Quang Huan, MsC

Definition

Iterator: Definition

Some operations on lists, most critically those to insert and remove from the middle of the list, require the notion of a position.

Iterators are used to point at the memory addresses of nodes in list.




```
v[k] == *(v.begin()+k)
```

Figure: Iterator and usage

Luu Quang Huan, MsC

Implementation in C++

```
class IntSLinkedList : public IList<int> {
    // ...
public:
    class Iterator {
    Node *pNode;
    IntSLinkedList* pList;

    public:
        Iterator(IntSLinkedList* pList = NULL,
                                bool begin = true);
        Iterator& operator=(const Iterator&
                                        iterator);
        T& operator*();
        bool operator!=(const Iterator& iterator);
        void remove();
        void set(int element);
        Iterator& operator++();
        Iterator operator++(int);
    };
}
```


Luu Quang Huan, MsC

Implementation in C++

```
class IntSLinkedList {
    //...
    public:
        Iterator begin();
        Iterator end();
};
```


Luu Quang Huan, MsC

Demo using Iterator

```
int main() {
    IntSLinkedList* list = new IntSLinkedList();
    for (int i = 0; i < 10; i++)
        list->add(i);

    for (IntSLinkedList::Iterator it = list->begin();
            it != list->end(); it++)
        cout << *it << endl;
}
```


Luu Quang Huan, MsC

THANK YOU.