List (P.1)
Array List and Singly Linked List

Data Structures and Algorithms

Dept. Computer Science
Faculty of Computer Science and Engineering
Ho Chi Minh University of Technology, VNU-HCM

Overview

Linear list concepts

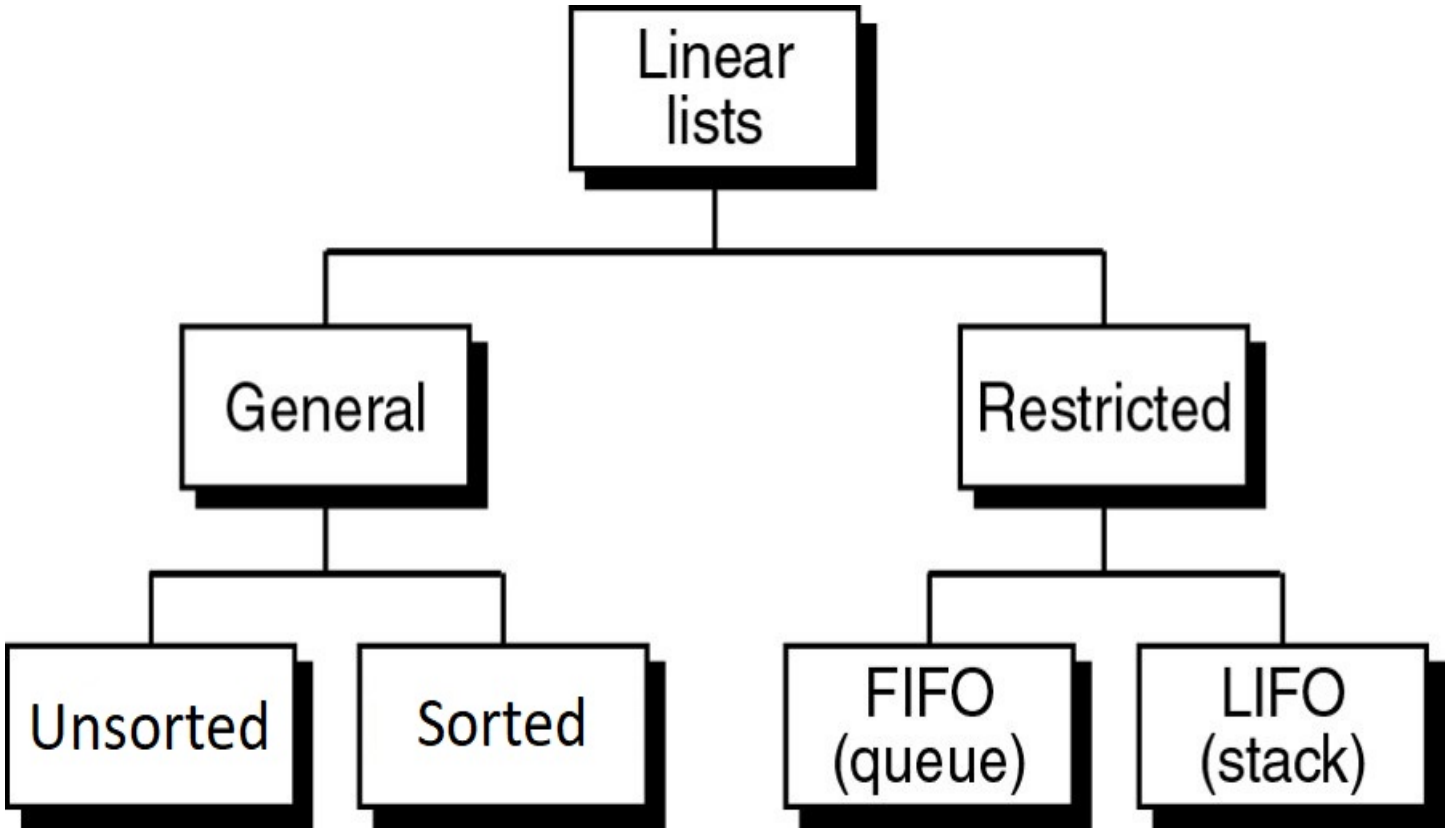
Linear list concepts

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.

Dept. Computer Science

Linear list concepts



Dept. Computer Science

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.

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).

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.

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.

List ADT

Extended operations:

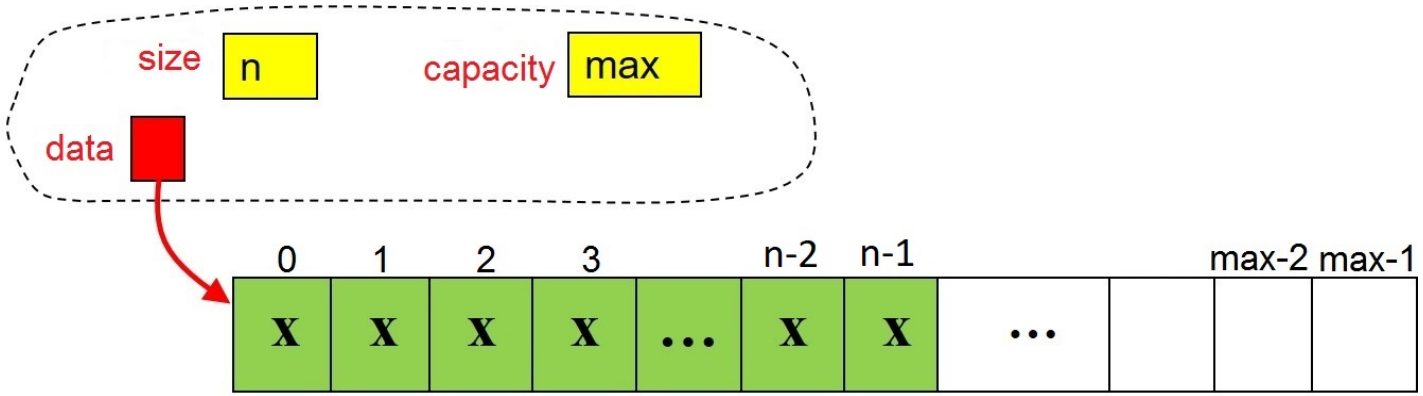
- Determine whether the list is empty or not.
- Determine whether the list is full or not.
- Find the size of the list.
- Clear the list to make it empty.
- Replace an element with another element.
- Merge two ordered list.
- Append an unordered list to another.

List ADT: Implementation in C++

```
1  template <class T>
2  class IList
3  {
4  public:
5      virtual void add(T e) = 0;
6      virtual void add(int index, T e) = 0;
7      virtual T removeAt(int index) = 0;
8      virtual bool removeItem(T item) = 0;
9      virtual bool empty() = 0;
10     virtual int size() = 0;
11     virtual void clear() = 0;
12     virtual T get(int index) = 0;
13     virtual void set(int index, T e) = 0;
14     virtual int indexOf(T item) = 0;
15     virtual bool contains(T item) = 0;
16     virtual string toString() = 0;
17 };
```


Array implementation

Dynamically Allocated Array




```
1 List // Contiguous Implementation of List
2   // number of used elements (mandatory)
3   count <integer>
4
5   // (Dynamically Allocated Array)
6   data <Array List of <DataType> >
7
8   capacity <integer>
9 End List
```


Array List: Implementation in C++ with Integer

```
1  class IntArrayList : public IList<int> {
2  protected:
3      int *data;
4      int capacity;
5      int count;
6  public:
7      IntArrayList();
8      virtual ~IntArrayList();
9      virtual void add(int element);
10     virtual void add(int index, int element);
11     virtual int removeAt(int index);
12     virtual bool removeItem(int item);
13     virtual bool empty();
14     virtual int size();
15     virtual void clear();
16     virtual int get(int index);
17     virtual void set(int index, int element);
18     virtual int indexOf(int item);
19     virtual bool contains(int item);
20     virtual string toString();
21     virtual void dump();
22 };
```


Array List: Implementation in C++ with Integer

```
1 class IntArrayList : public IList<int>
2 {
3 // ...
4
5 private:
6     void checkIndex(int index);
7     void ensureCapacity(int capacity);
8 };
```


Array List: Implementation in C++ with Integer

```
1  IntArrayList::IntArrayList()
2  {
3      this->capacity = 10;
4      this->data = new int[this->capacity];
5      this->count = 0;
6  }
7
8  IntArrayList::~IntArrayList()
9  {
10     delete[] this->data;
11 }
```


Array List: Implementation in C++ with Integer

```
1  void IntArrayList::add(int element)
2  {
3      this->ensureCapacity(this->count + 1);
4
5      this->data[this->count] = element;
6      this->count++;
7  }
8
9  void IntArrayList::add(int index, int element)
10 {
11     this->checkIndex(index);
12     this->ensureCapacity(this->count + 1);
13
14     int moveCount = this->count - index;
15     if (moveCount > 0)
16         memmove(this->data + index + 1,
17                 this->data + index,
18                 moveCount * sizeof(int));
19
20     this->data[index] = element;
21     this->count++;
22 }
```


Array List: Implementation in C++ with Integer

```
1  int IntArrayList::removeAt(int index) {
2      this->checkIndex(index);
3      int elementToRemove = this->data[index];
4
5      int moveCount = this->count - index - 1;
6      if (moveCount > 0)
7          memmove(this->data + index,
8                  this->data + index + 1,
9                  sizeof(int) * moveCount);
10
11     this->count--;
12     return elementToRemove;
13 }
14
15 bool IntArrayList::removeItem(int element) {
16     for (int index = 0; index < this->count; index++)
17     {
18         if (this->data[index] == element) {
19             this->removeAt(index);
20             return true;
21         }
22     }
23     return false;
24 }
```


Array List: Implementation in C++ with Integer

```
1  bool IntArrayList::empty()
2  {
3      return this->count == 0;
4  }
5
6  int IntArrayList::size()
7  {
8      return this->count;
9  }
10
11 void IntArrayList::clear()
12 {
13     delete[] this->data;
14
15     this->capacity = 10;
16     this->data = new int[this->capacity];
17     this->count = 0;
18 }
```


Array List: Implementation in C++ with Integer

```
1  int IntArrayList::get(int index) {
2      this->checkIndex(index);
3
4      return this->data[index];
5  }
6
7  void IntArrayList::set(int index, int element) {
8      this->checkIndex(index);
9
10     this->data[index] = element;
11 }
12
13 int IntArrayList::indexOf(int element) {
14     for (int index = 0; index < this->count; index++)
15         if (this->data[index] == element)
16             return index;
17     return -1;
18 }
19
20 bool IntArrayList::contains(int element) {
21     return this->indexOf(element) != -1;
22 }
```


Dept. Computer Science

Array List: Implementation in C++ with Integer

```
1  string IntArrayList::toString() {
2      stringstream ss;
3      ss << "[";
4      for (int index = 0; index < count - 1; index++)
5          ss << data[index] << ", ";
6
7      if (count > 0) ss << data[count - 1] << "]";
8      else ss << "]";
9
10     return ss.str();
11 }
12
13 void IntArrayList::dump() {
14     string line(50, '=');
15     cout << line << endl;
16     cout << "Integer list's information:" << endl;
17     cout << "- Capacity: " << this->capacity << endl;
18     cout << "- Size: " << this->count << endl;
19     cout << "- Data: " << this->toString() << endl;
20     cout << line << endl;
21 }
```


Array List: Implementation in C++ with Integer

```
1  void IntArrayList::checkIndex(int index) {
2      if (index < 0 || index >= this->count)
3          throw std::out_of_range(
4                      "Index is out of range");
5  }
6
7  void IntArrayList::ensureCapacity(int capacity) {
8      if (capacity > this->capacity) {
9          int newCapacity = this->capacity * 3 / 2;
10         int *newData = new int[newCapacity];
11         memmove(newData, this->data,
12                          this->count * sizeof(int));
13         this->capacity = newCapacity;
14         delete[] this->data;
15
16         this->data = newData;
17     }
18 }
```


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.

Singly linked list

Linked List

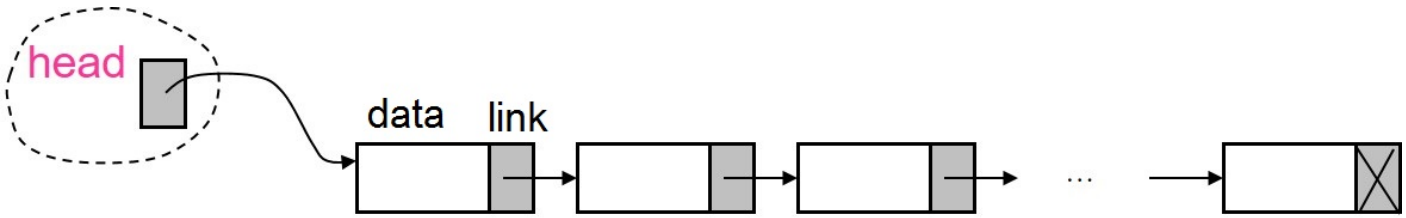


Figure: Singly Linked List

Structure

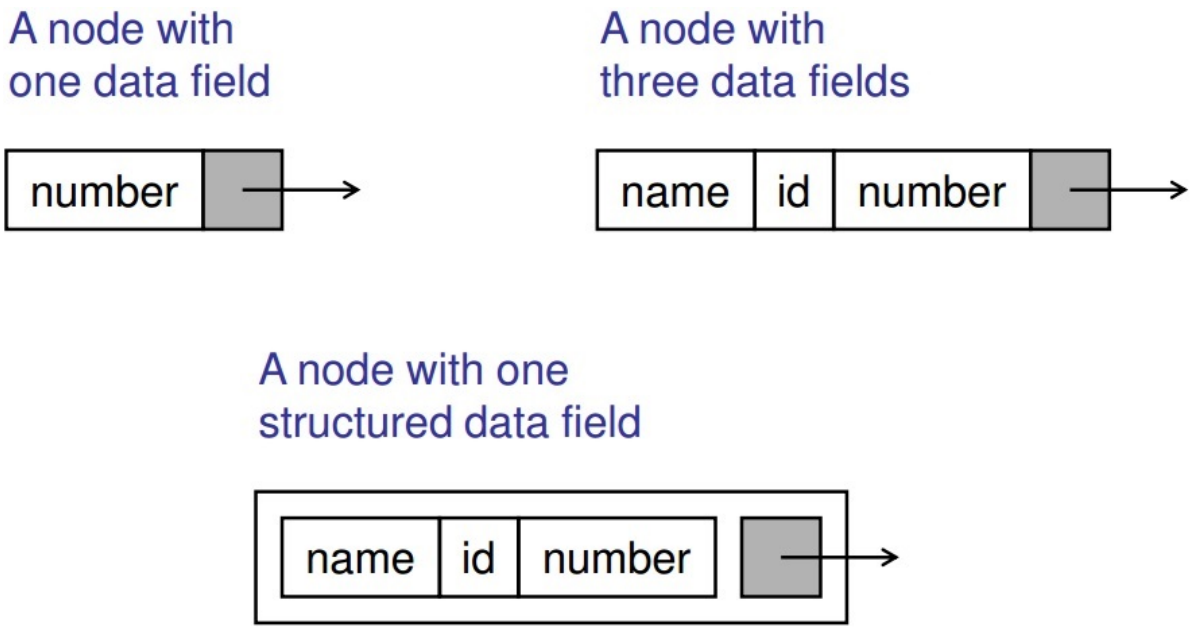
```
list // Linked Implementation of List
    head <pointer>
    tail <pointer>  // (optional)
    count <integer> // number of elements (optional)
end list
```


Nodes

Definition

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.



Dept. Computer Science

Nodes

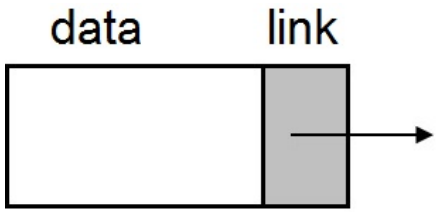
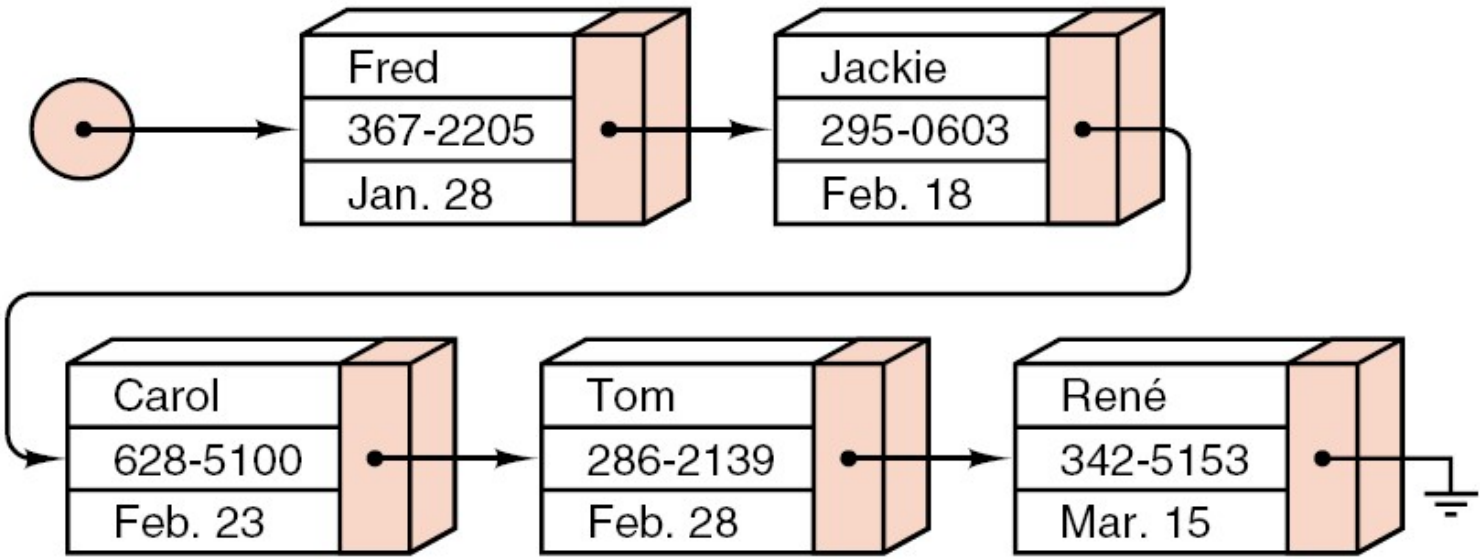


Figure: Linked list node structure

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

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


Example



Example

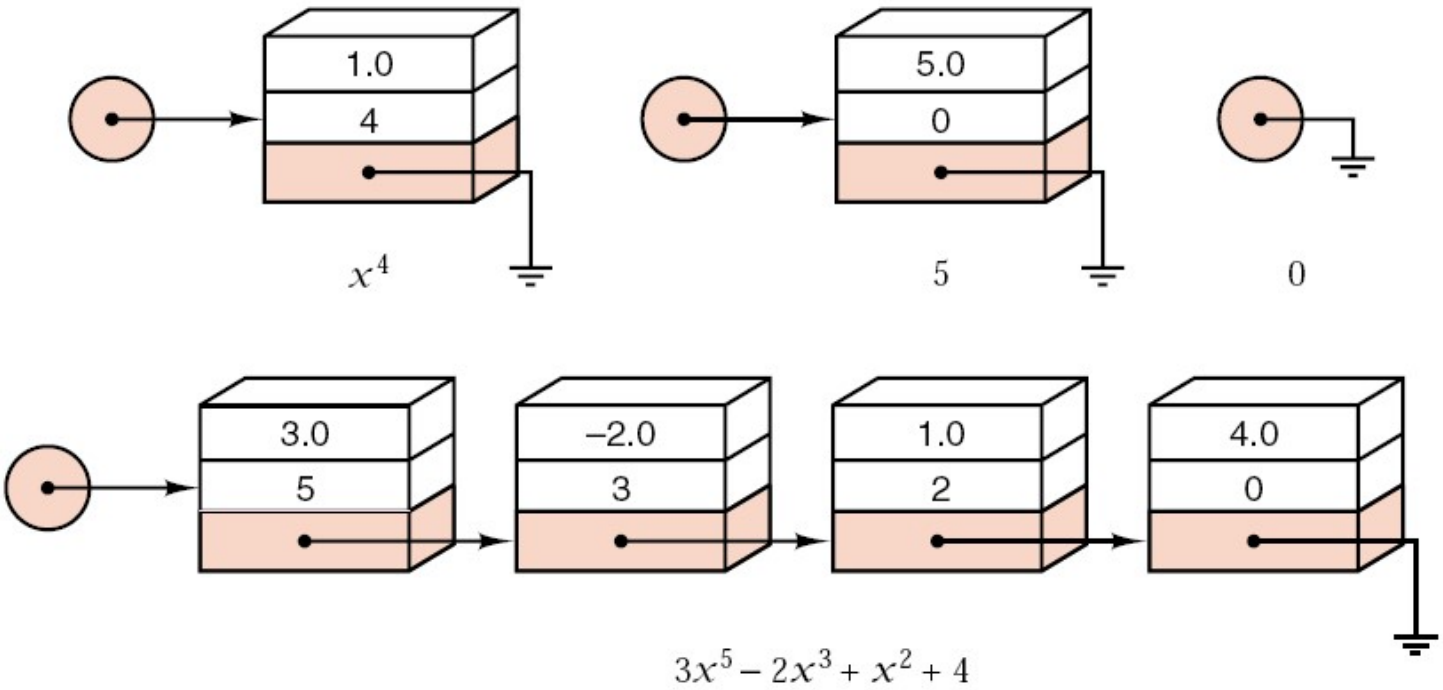


Figure: List representing polynomial

Implementation in C++

Example

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

```
1 struct Node {
2     int data;
3     Node *next;
4 };
```


Implementation in C++ with `struct`

Example

```
1 struct Node {
2     int data;
3     Node *next;
4 };
```

```
1 struct Node {
2     float data;
3     Node *next;
4 };
```

In general, with `template`:

```
1 template <class T>
2 struct Node {
3     T data;
4     Node<T> *next;
5 };
```


Node implementation in C++ with nested class

```
1  class IntSLinkedList : public IList<int> {
2  public:
3      class Node; // Forward declaration
4
5  protected:
6      Node* head;
7      Node* tail;
8      int count;
9
10 public:
11     IntSLinkedList() :
12         head(NULL), tail(NULL), count(0) {};
13
14 public:
15     class Node {
16     protected:
17         int data;
18         Node *next;
19
20     public:
21         Node(int data = 0) {
22             int data = 0;
23             this->next = NULL;
24         }
25     }
26 };
```


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
- ...

Insertion: Implementation in C++

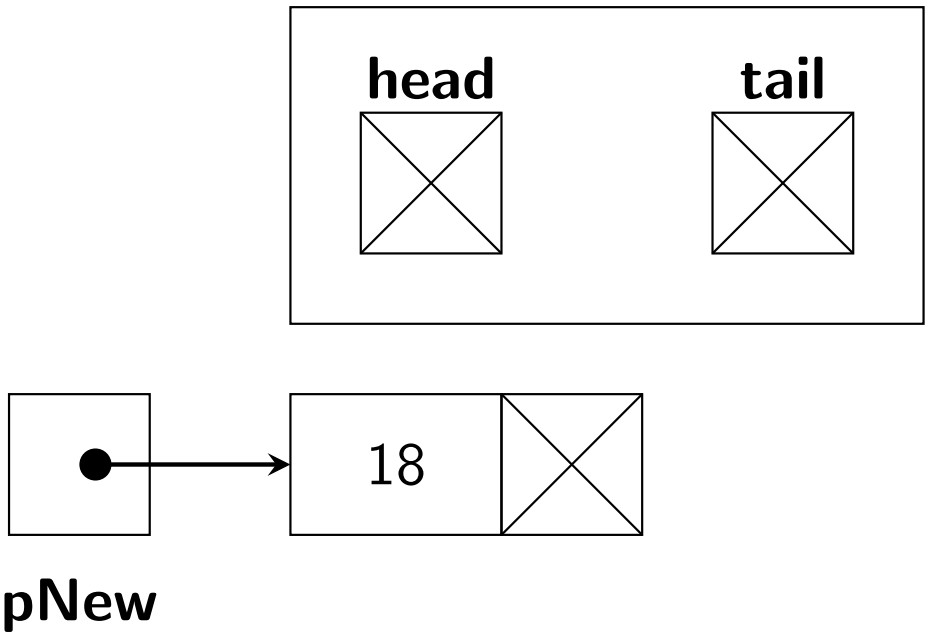
```
1 class IntSLinkedList : public IList<int> {
2 // Declaration of attributes.
3 // Declaration of constructor, destructor.
4 // Declaration of nested classes.
5 public:
6     virtual void add(int element);
7     virtual void add(int index, int element);
8 };
```


Insert an item into a linked list

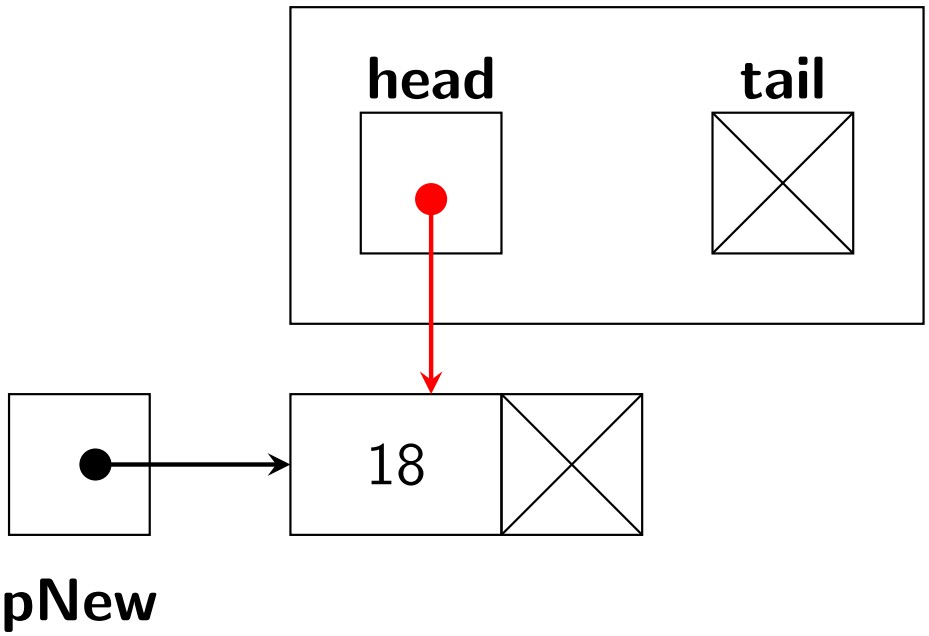
Cases with insertion:

1. Case 1: Empty list
2. Case 2: Non-empty list
   - Prepend an item into a list
   - Append an item into a list
   - Insert an item at a specific position in list

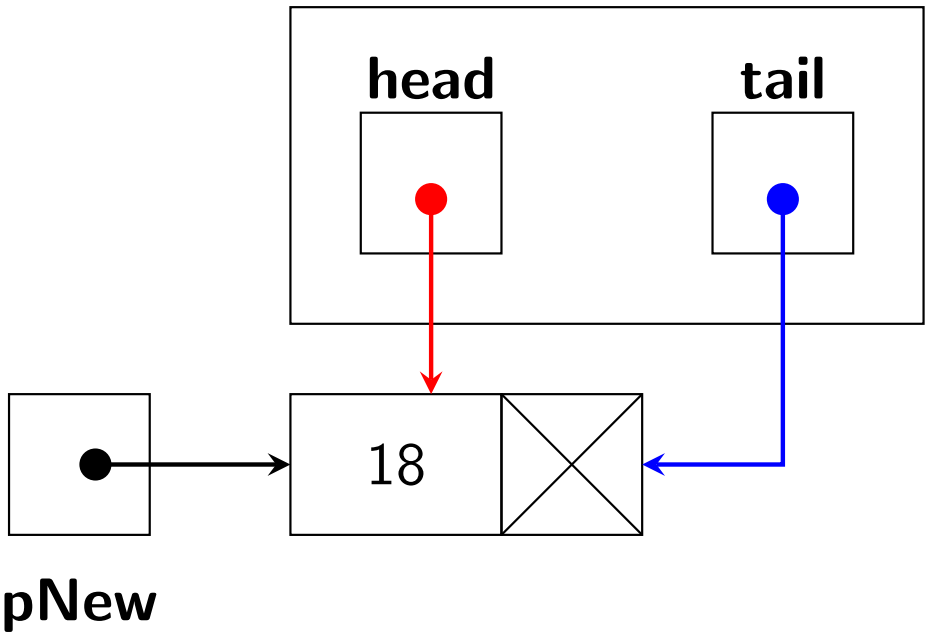
Insertion: Prepend to an empty list



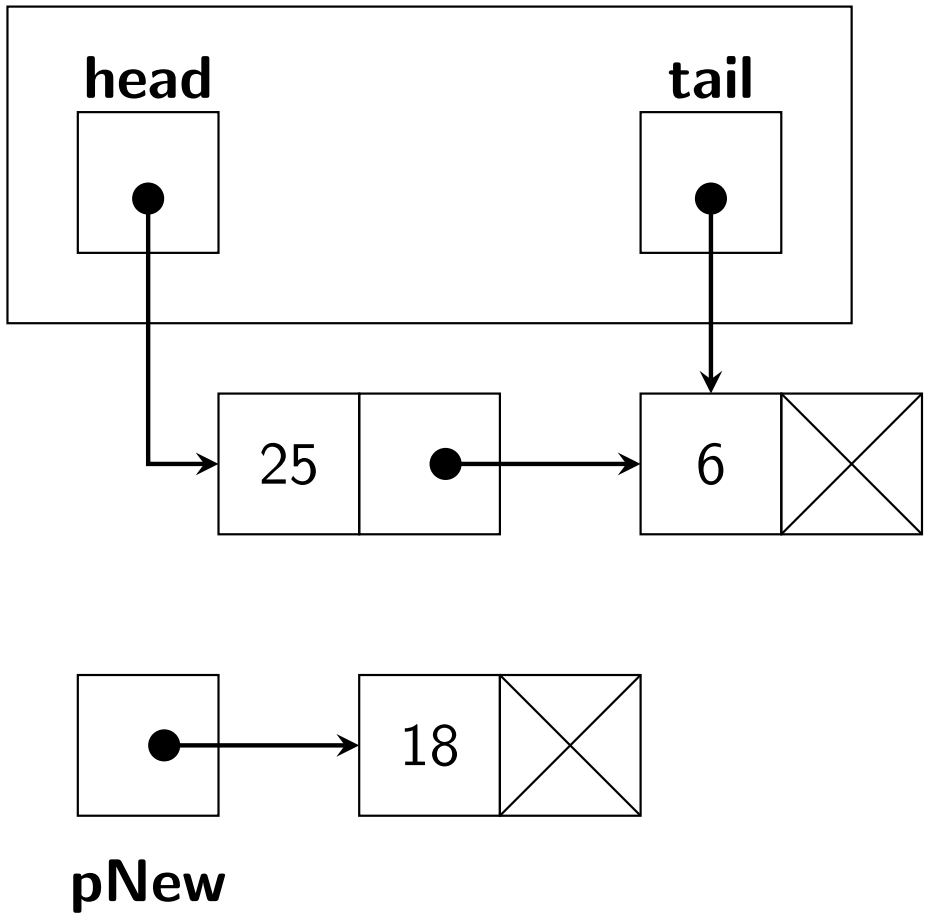
Insertion: Prepend to an empty list



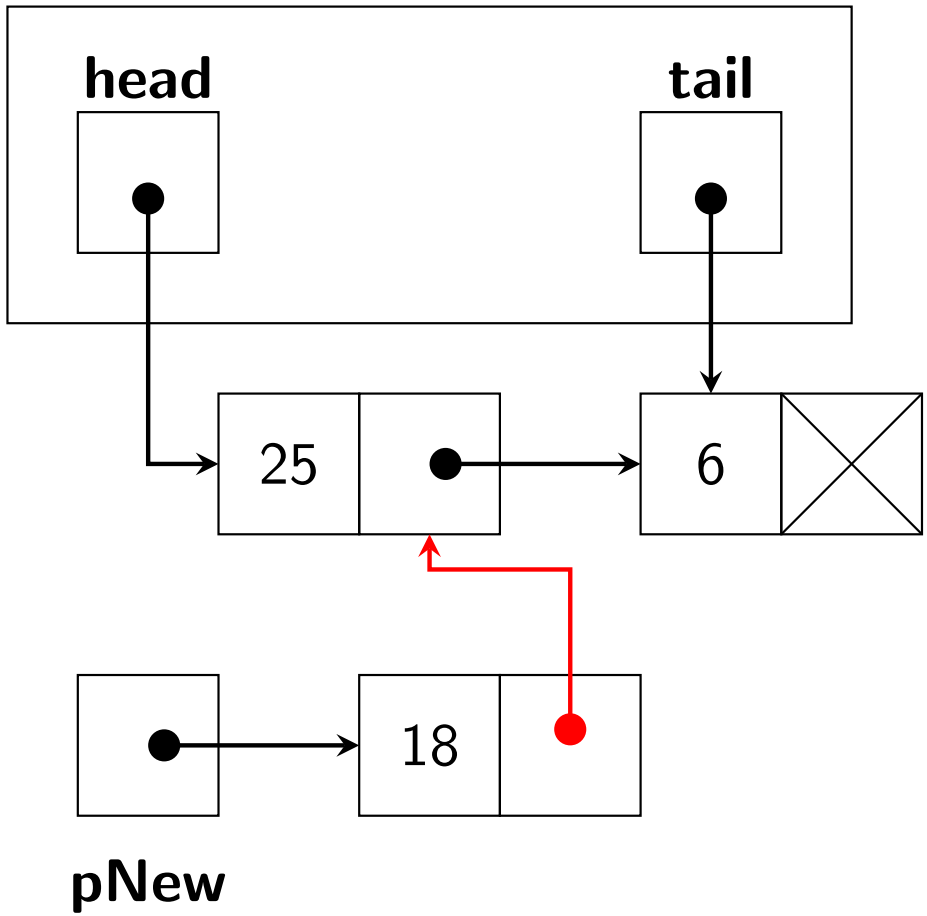
Insertion: Prepend to an empty list



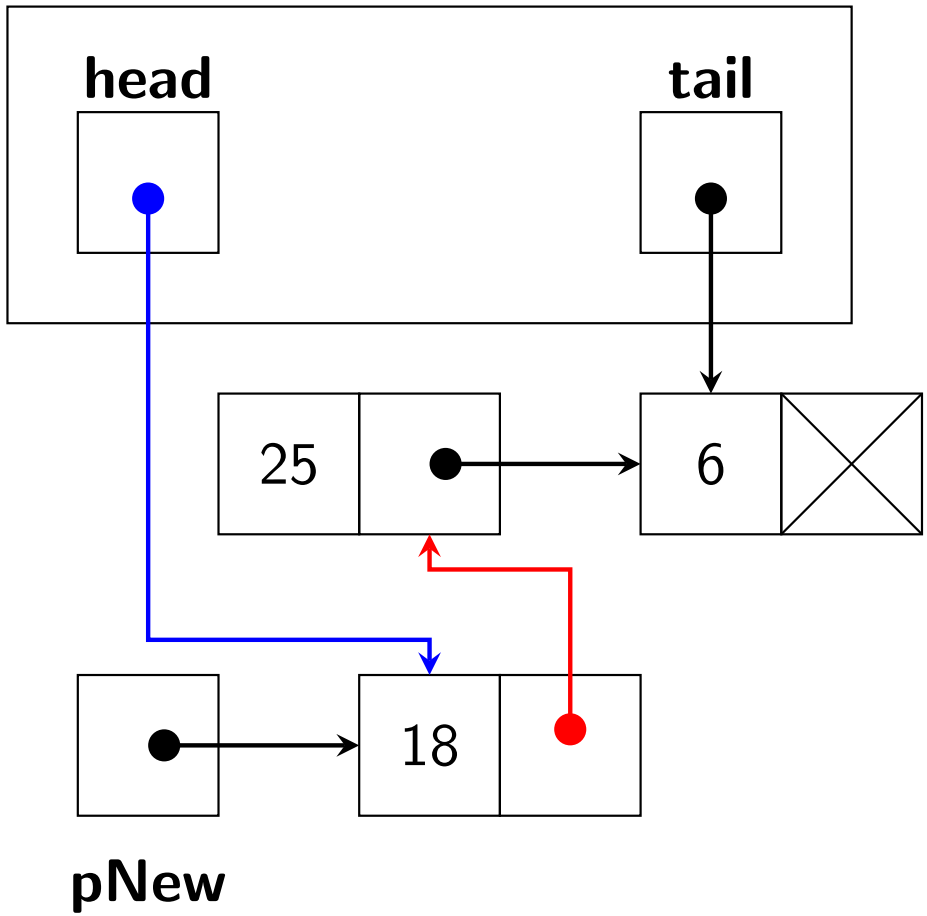
Insertion: Prepend to a non-empty list



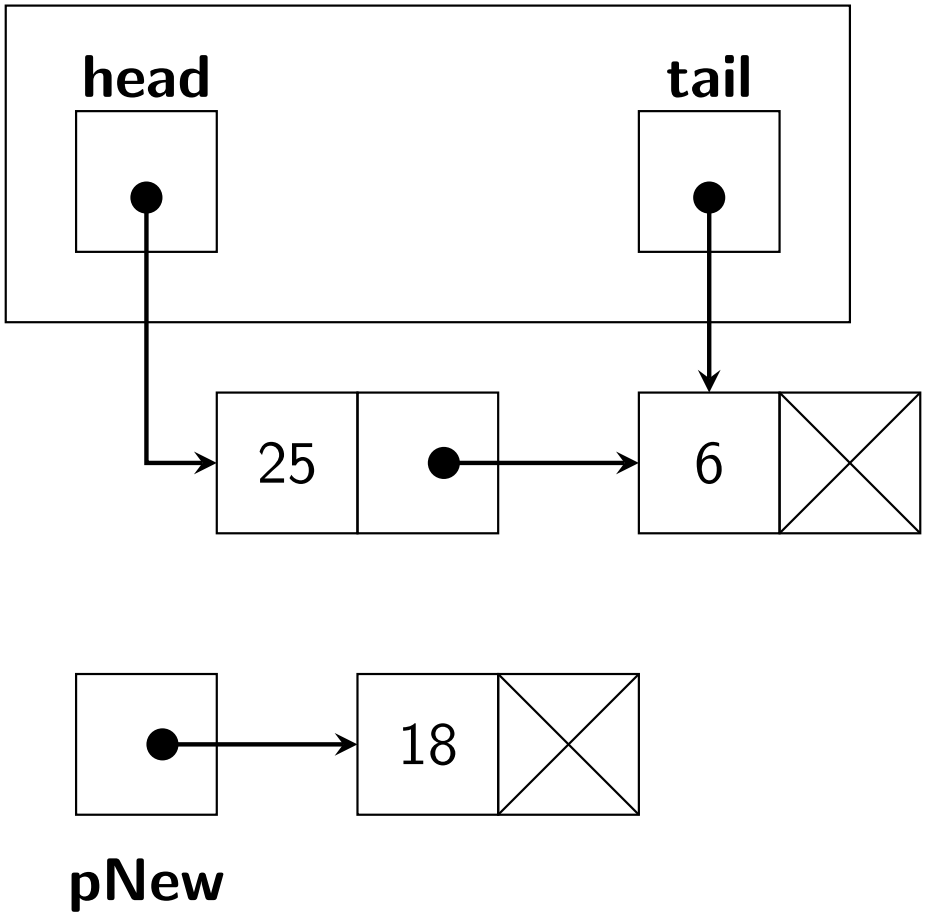
Insertion: Prepend to a non-empty list



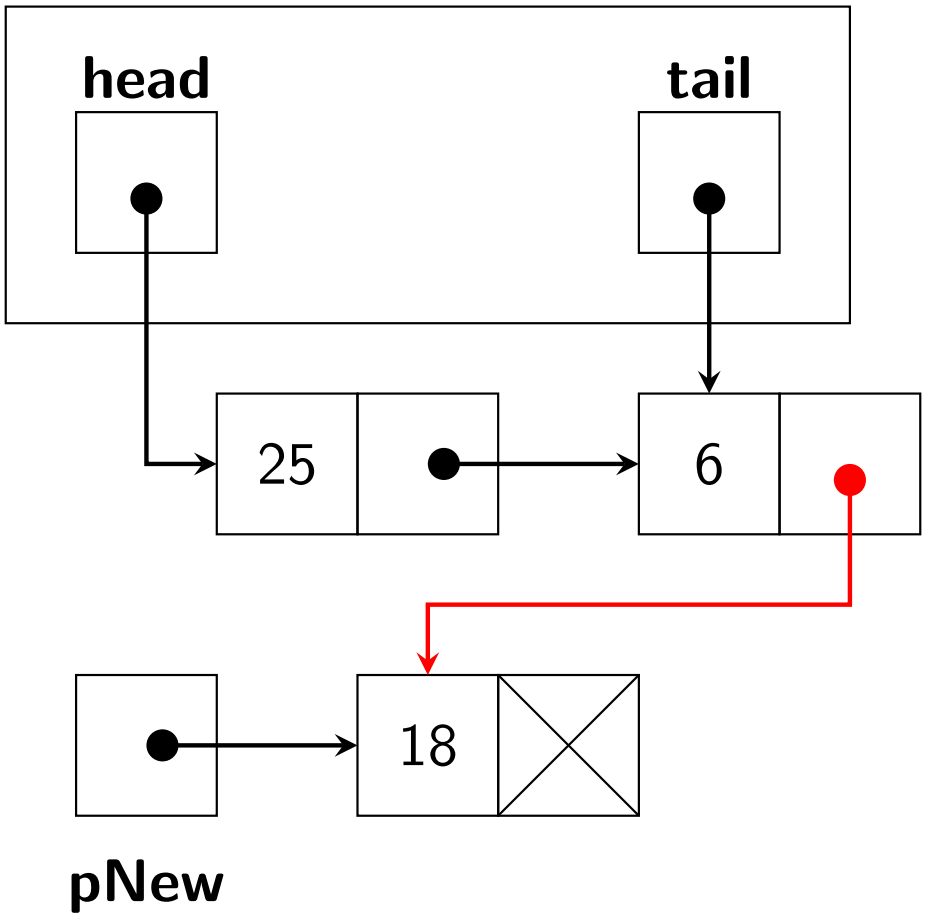
Insertion: Prepend to a non-empty list



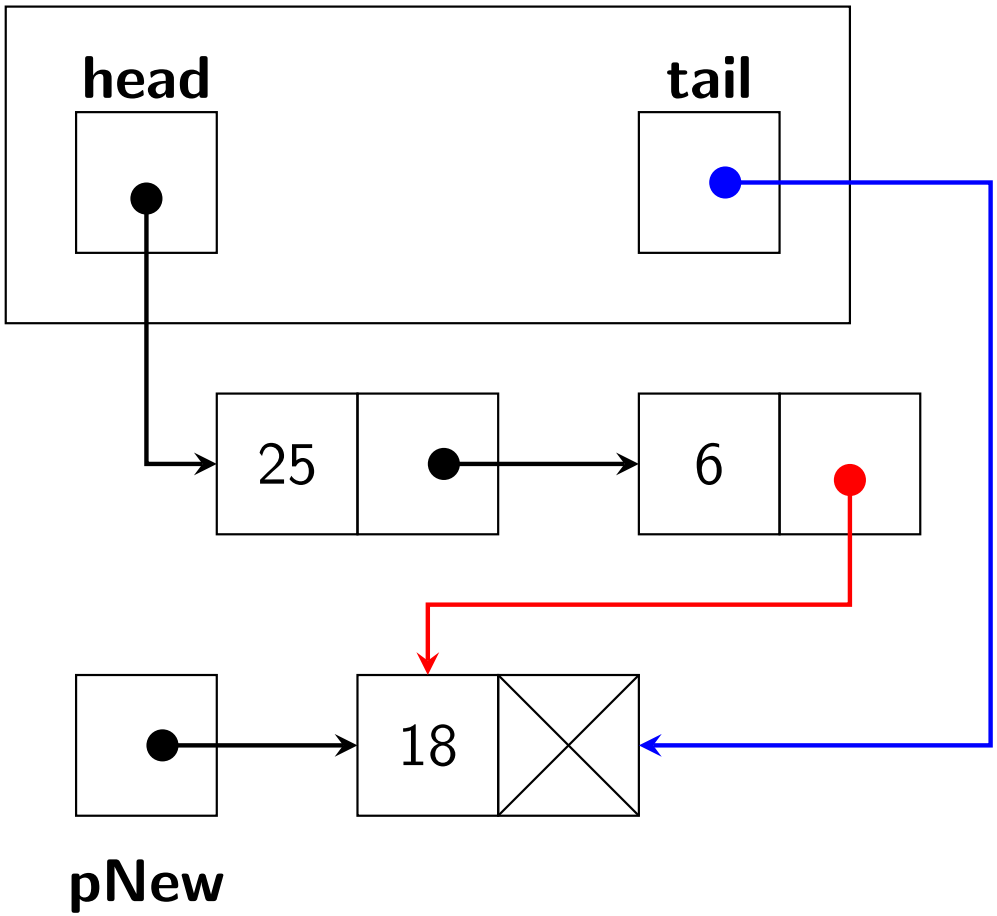
Insertion: Append to the list



Insertion: Append to the list

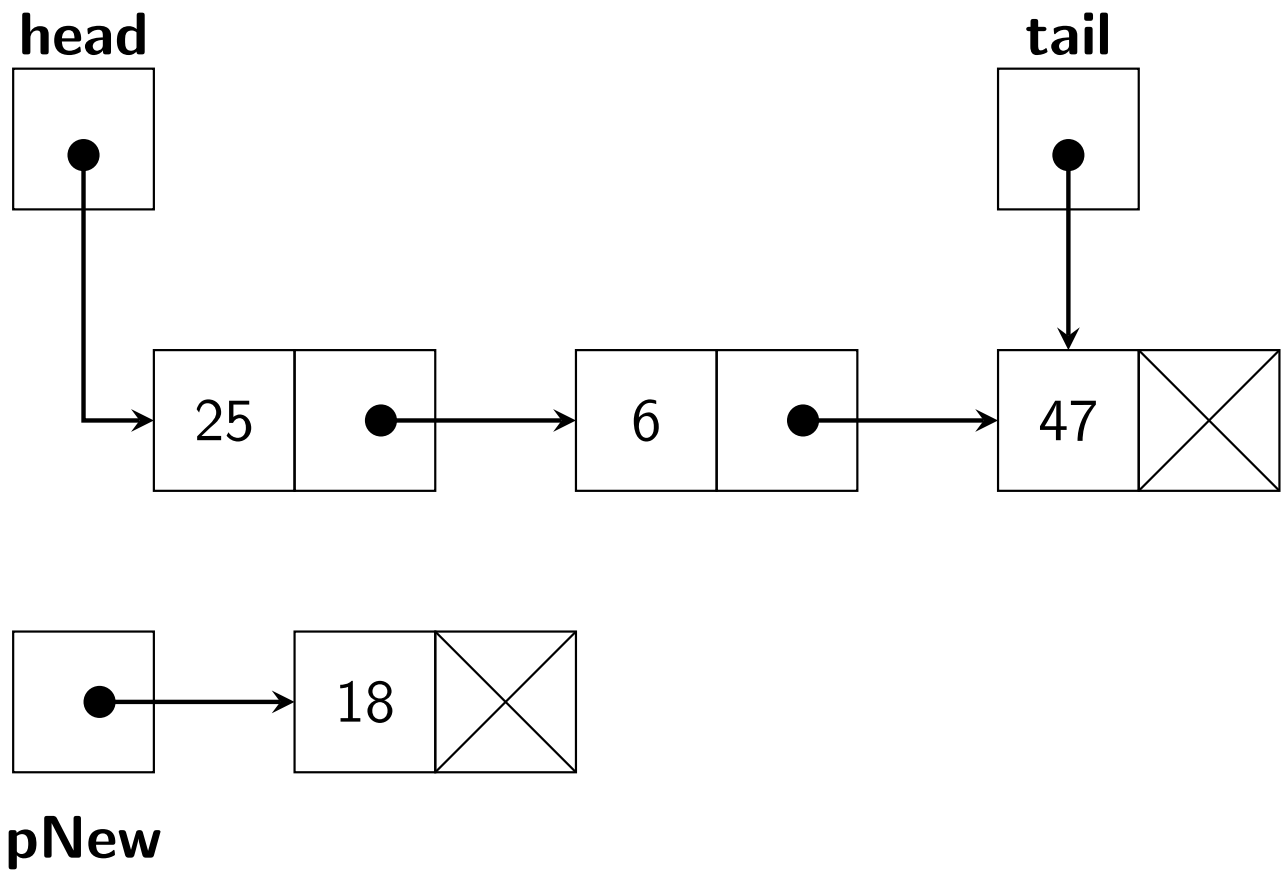


Insertion: Append to the list



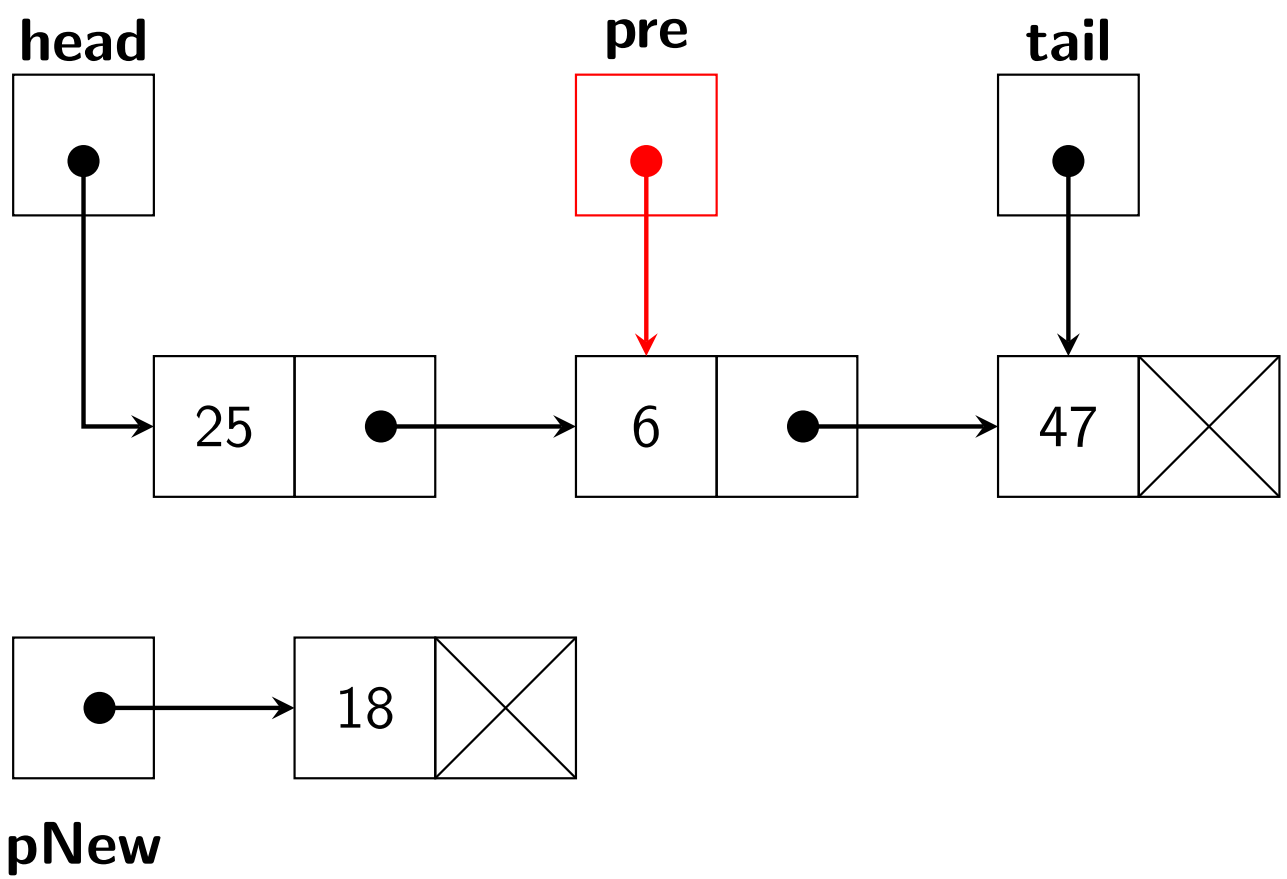
Insertion: At the index i

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



Insertion: At the index i

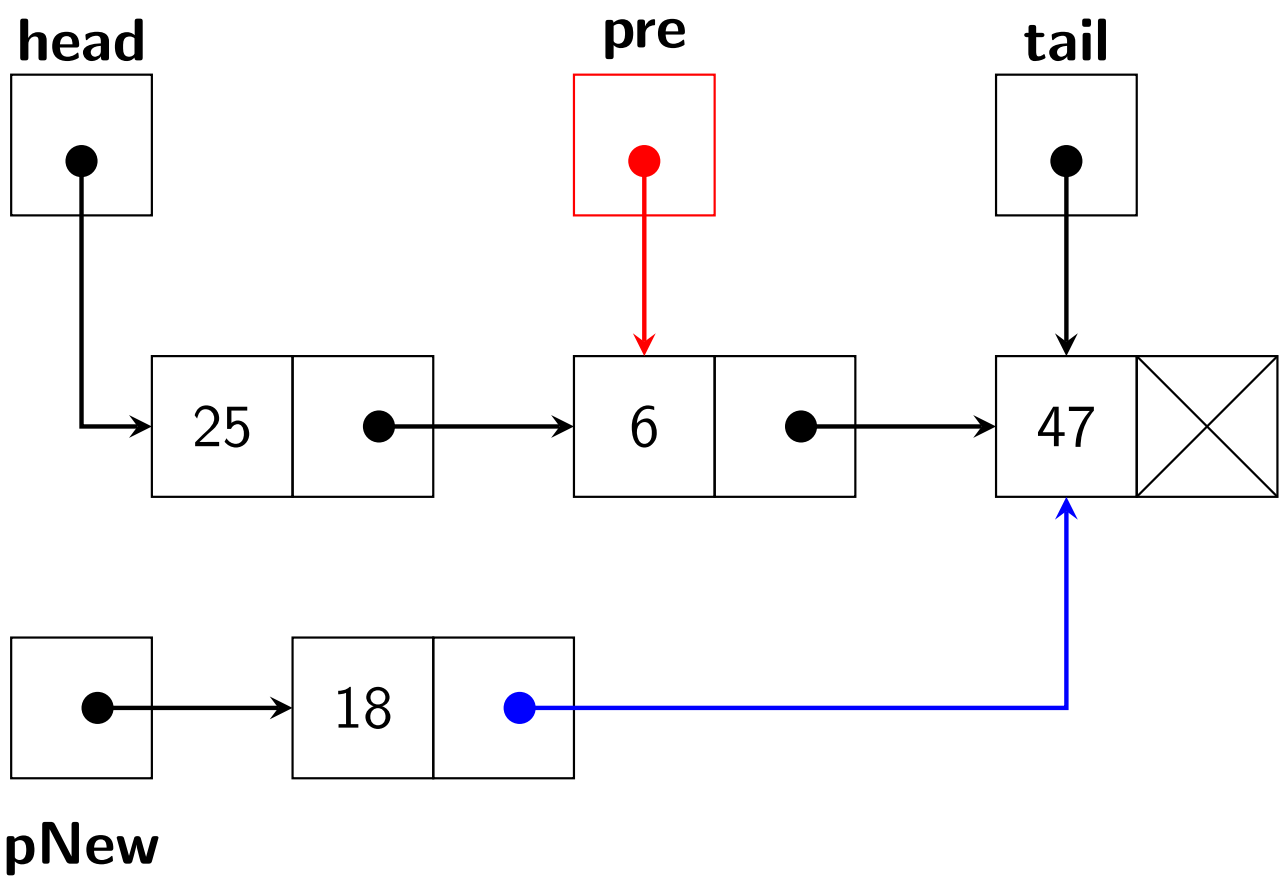
Insert **18** at index **2**.



Dept. Computer Science

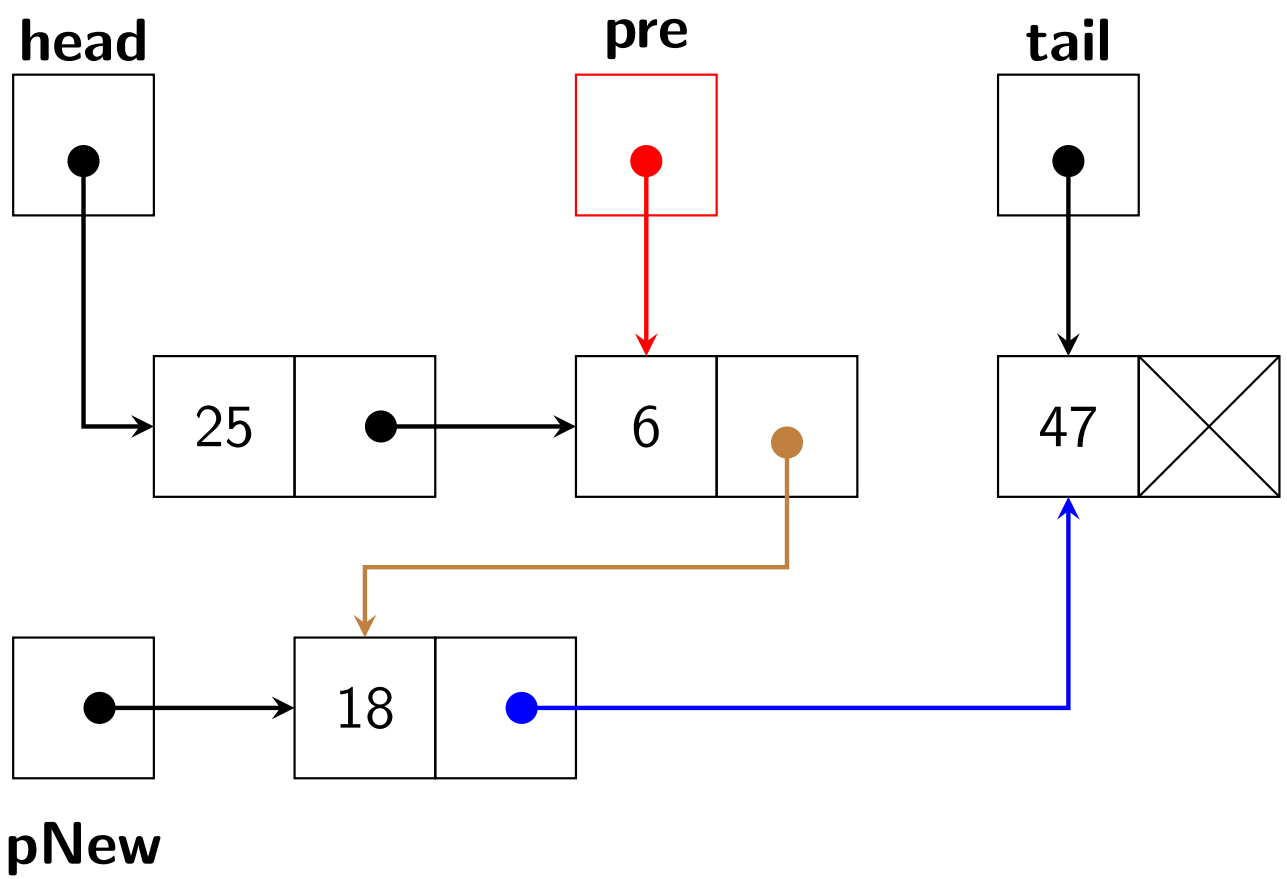
Insertion: At the index i

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



Insertion: At the index i

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

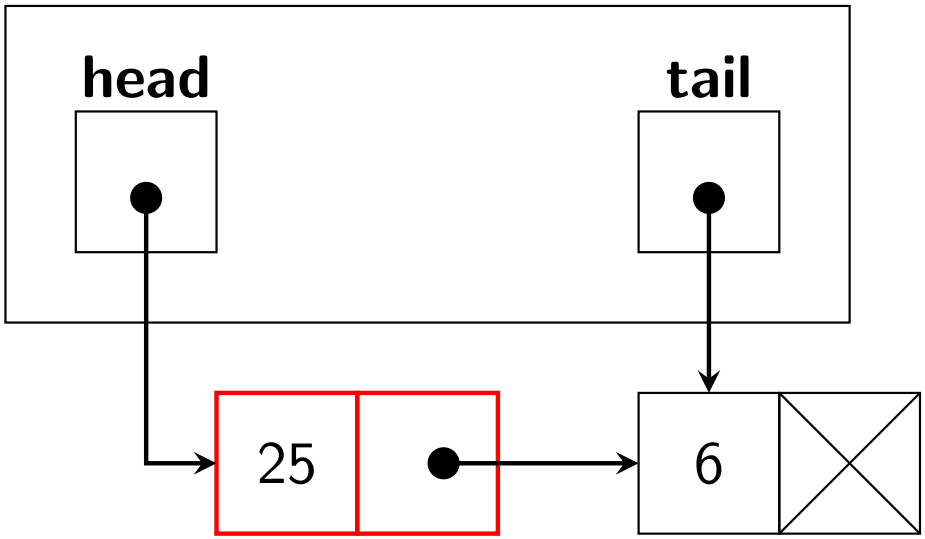


Dept. Computer Science

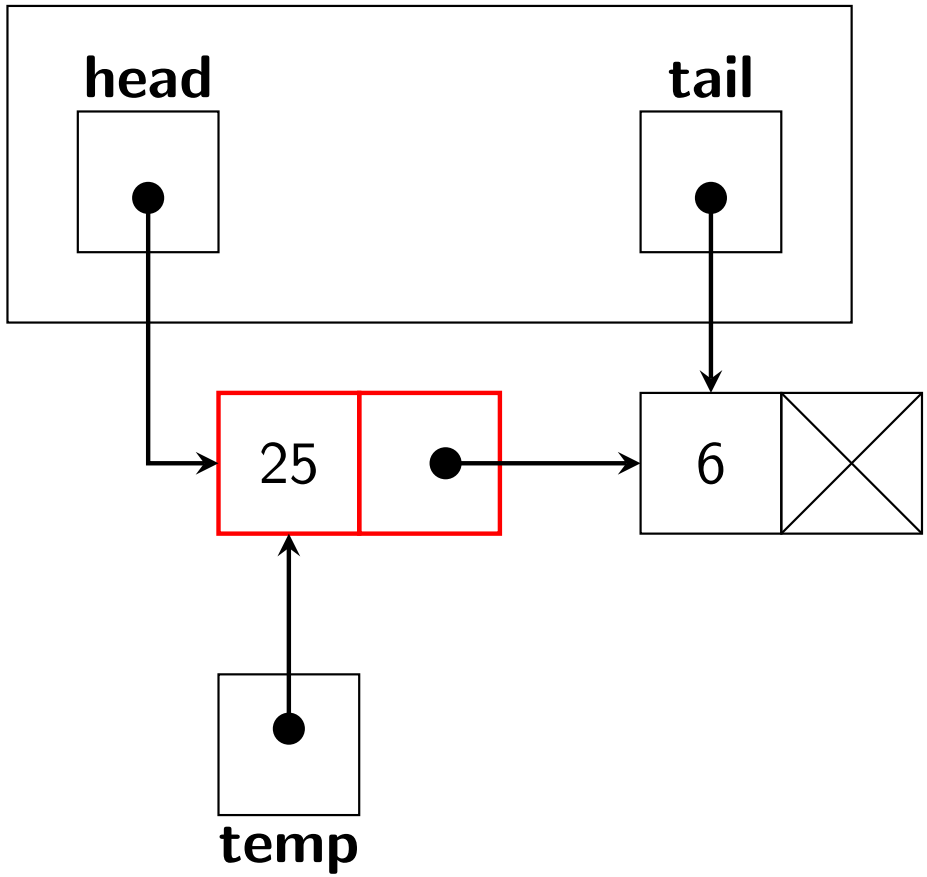
Removal: Implementation in C++

```
1 class IntSLinkedList : public IList<int> {
2 // Declaration of attributes, constructor, destructor.
3 // Declaration of nested classes.
4 public:
5     virtual int removeAt(int index);
6     virtual bool removeItem(int item);
7 };
```

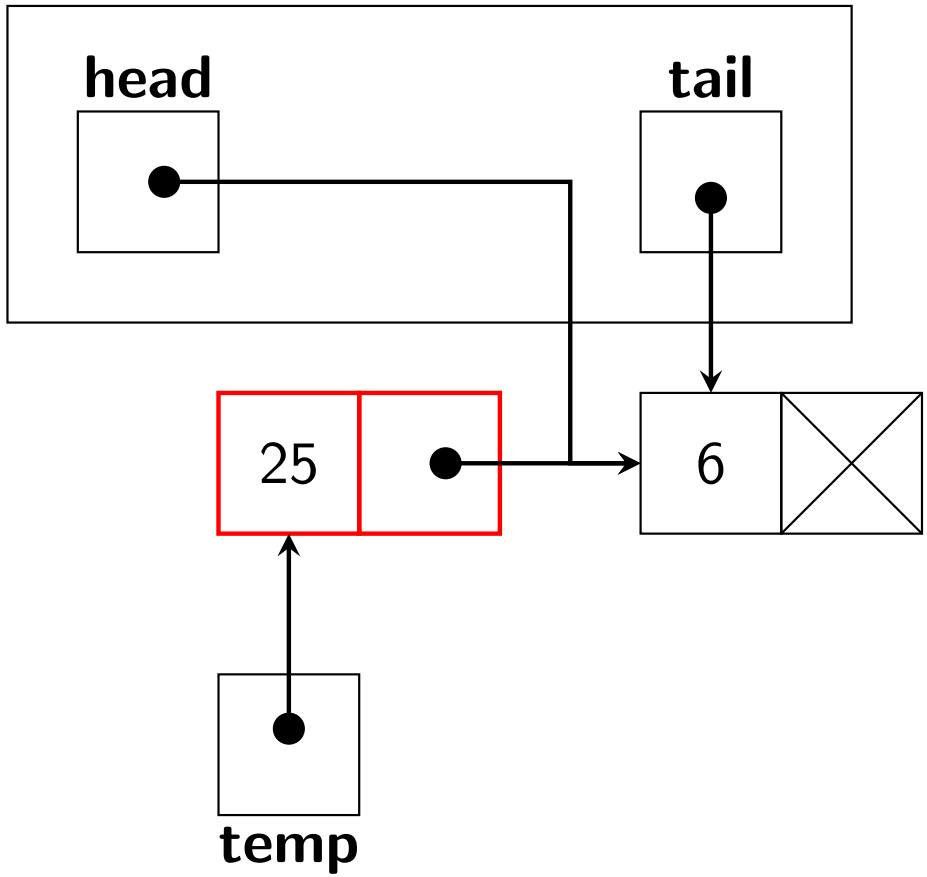

Removal: Delete the first element



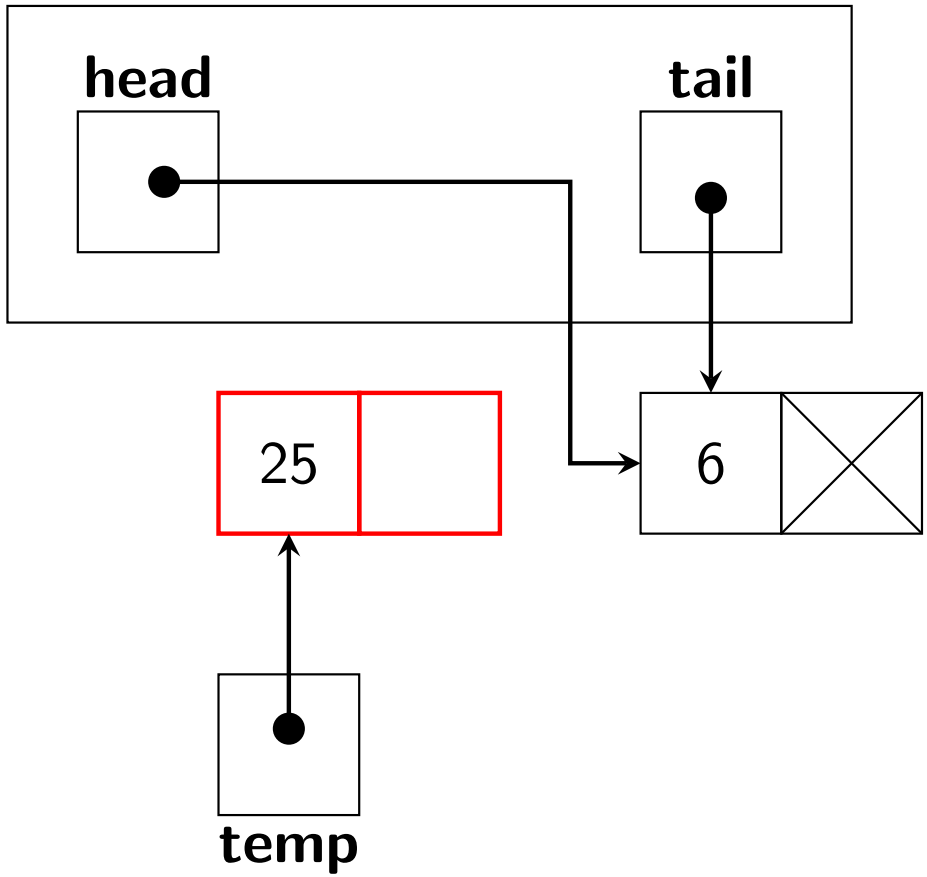
Removal: Delete the first element



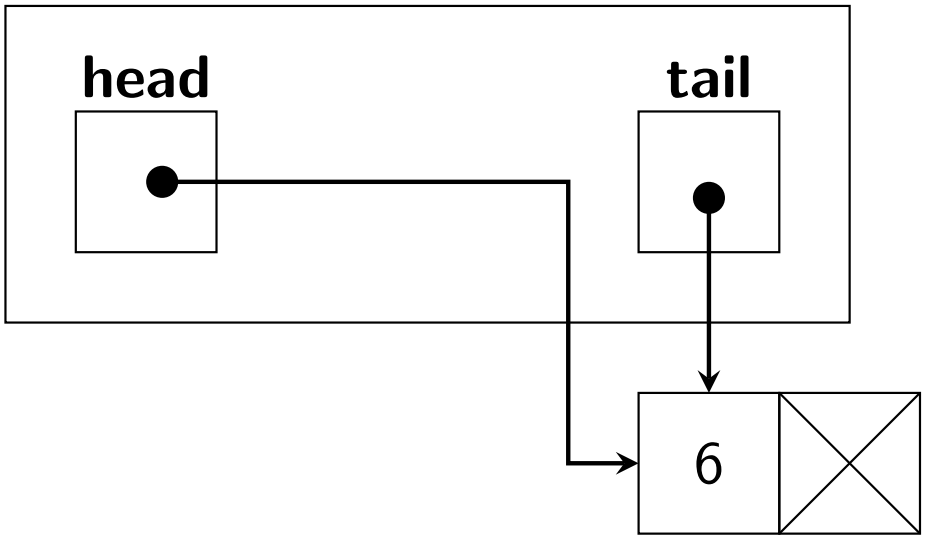
Removal: Delete the first element



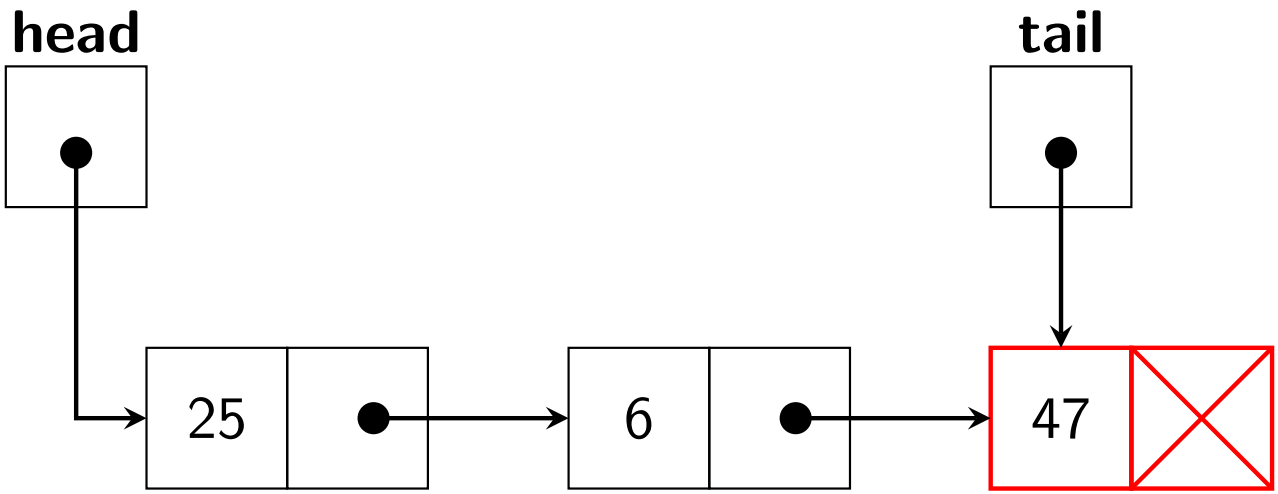
Removal: Delete the first element



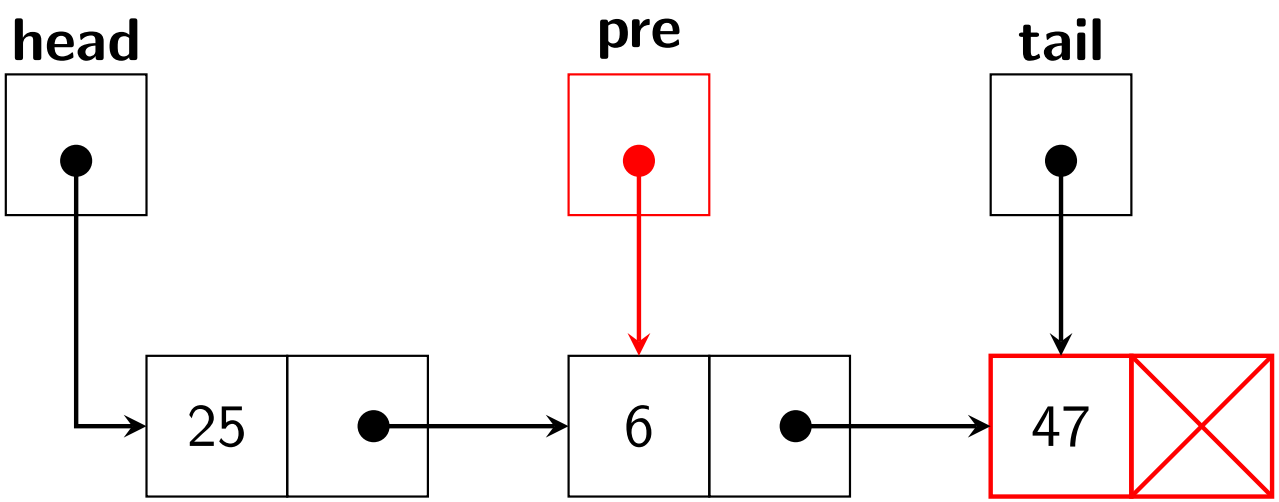
Removal: Delete the first element



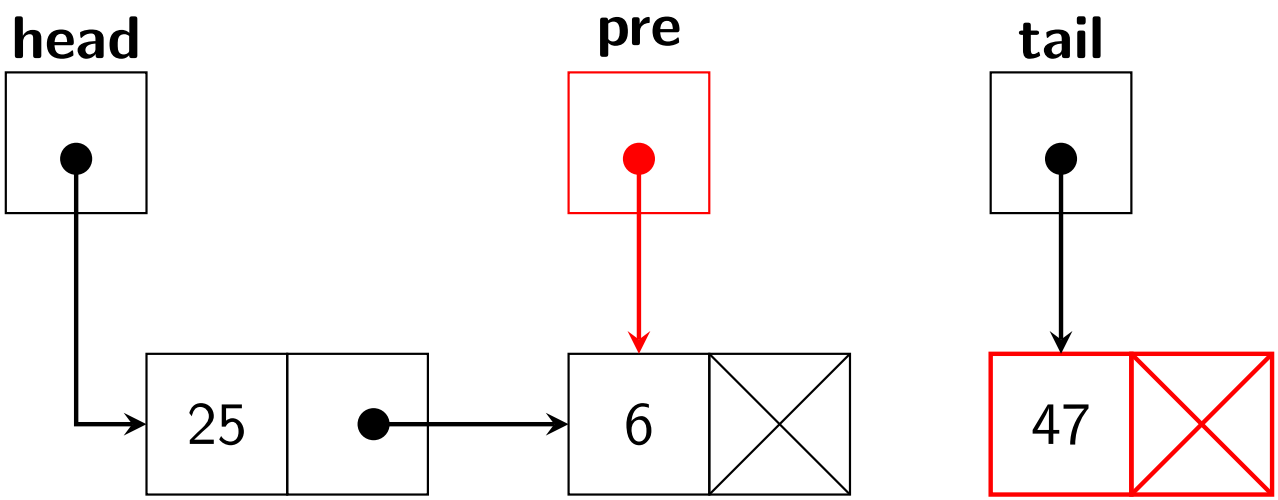
Removal: Delete the last element



Removal: Delete the last element

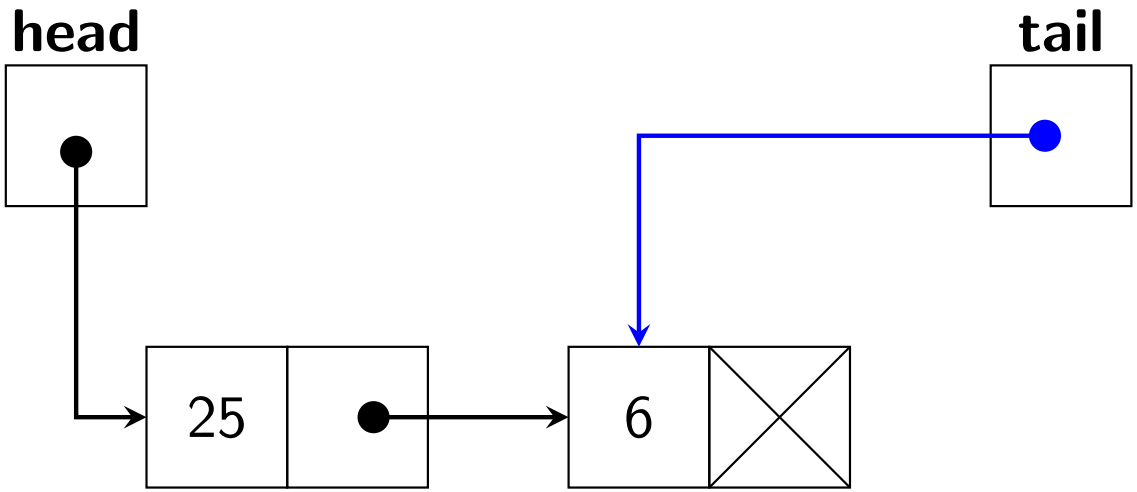


Removal: Delete the last element

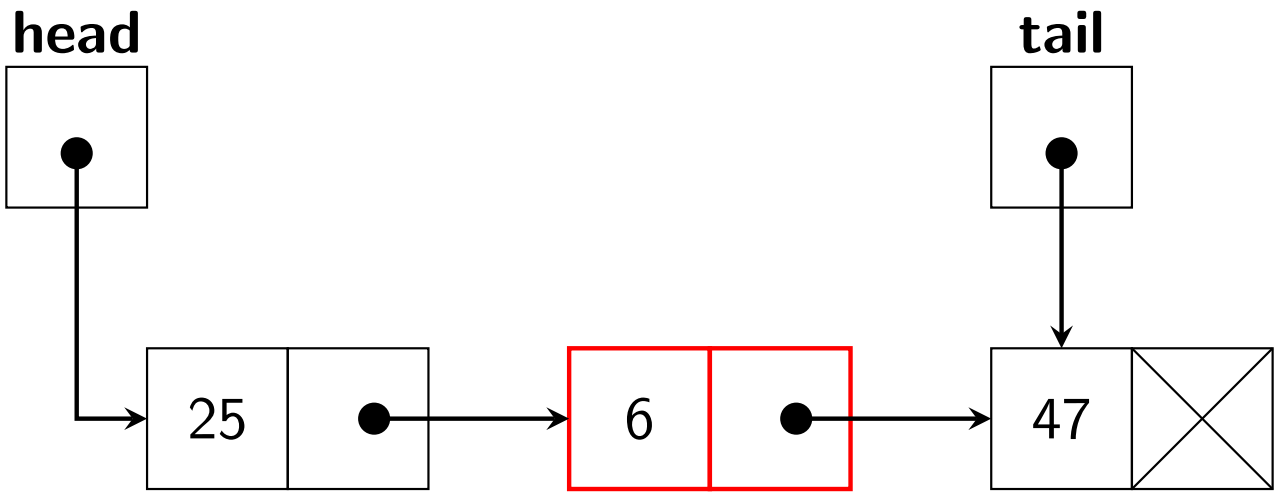


Dept. Computer Science

Removal: Delete the last element

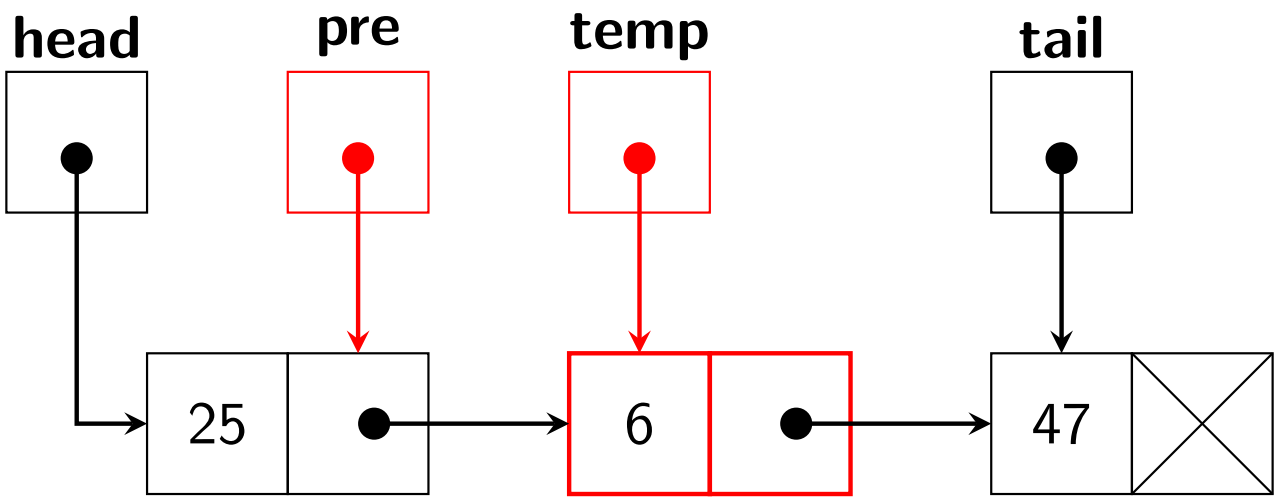


Removal: Delete at index i



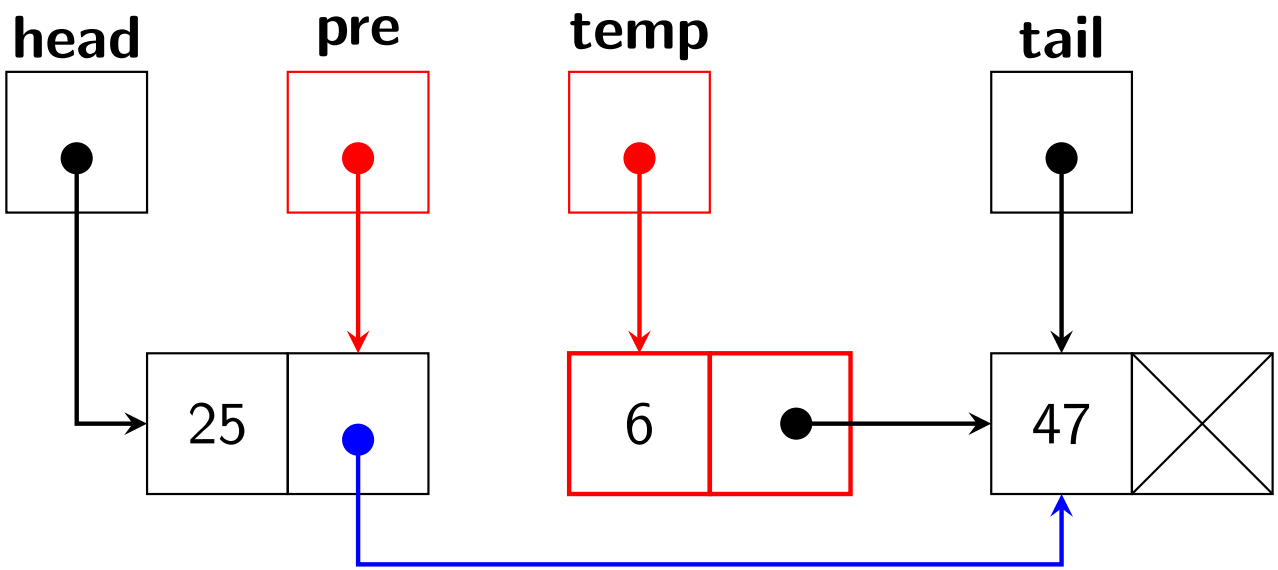
Dept. Computer Science

Removal: Delete at index i



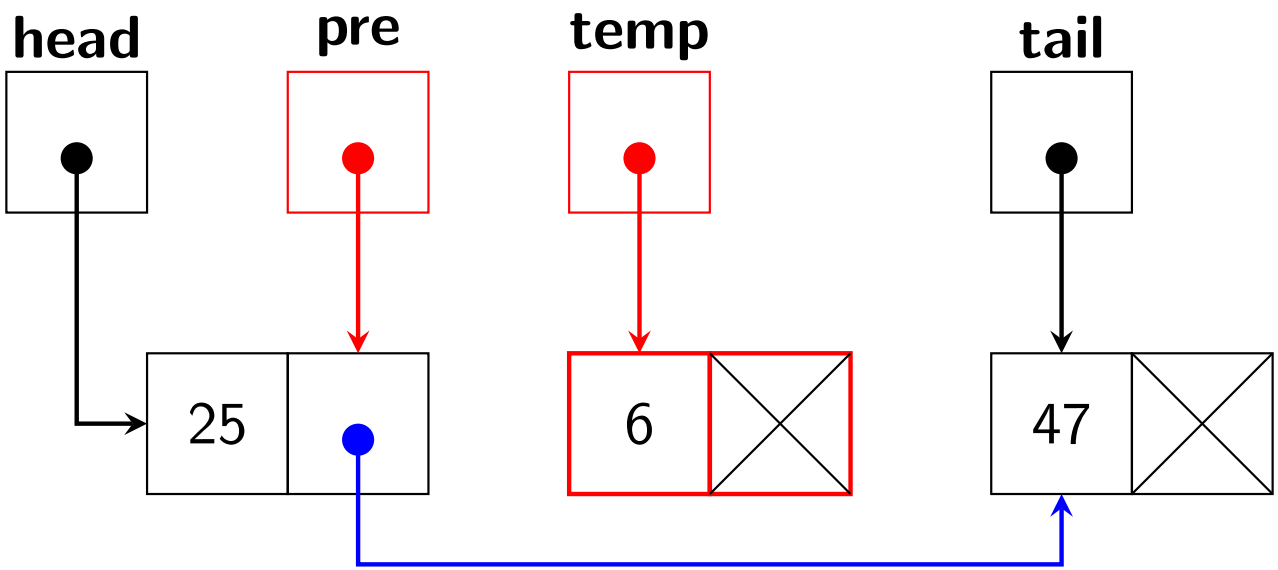
Dept. Computer Science

Removal: Delete at index i



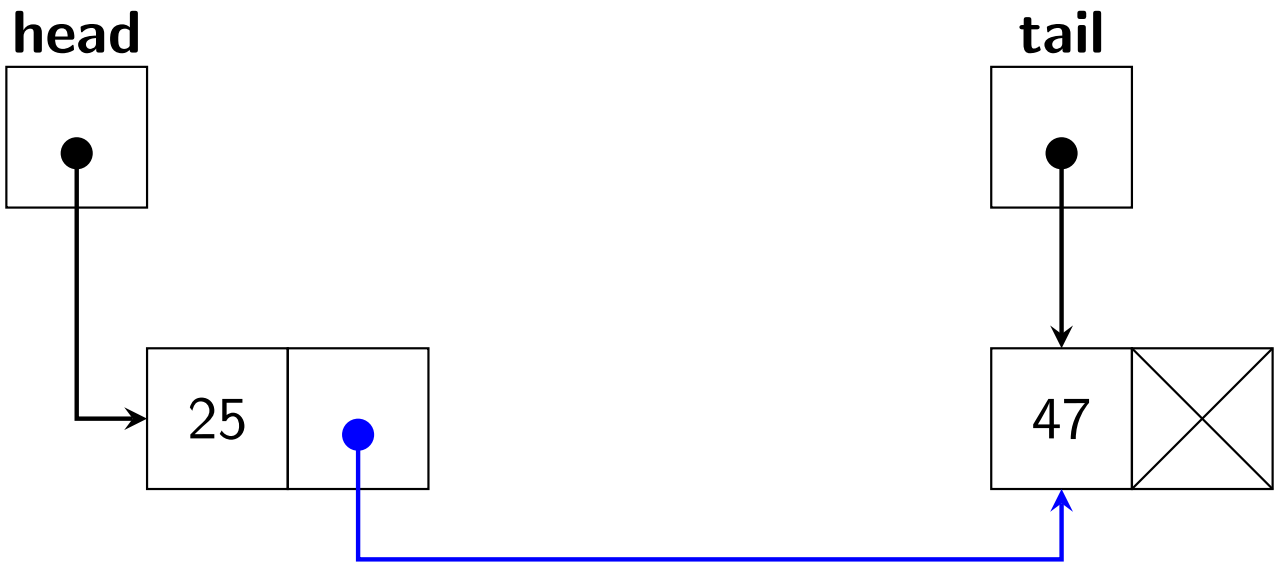
Dept. Computer Science

Removal: Delete at index i



Dept. Computer Science

Removal: Delete at index i

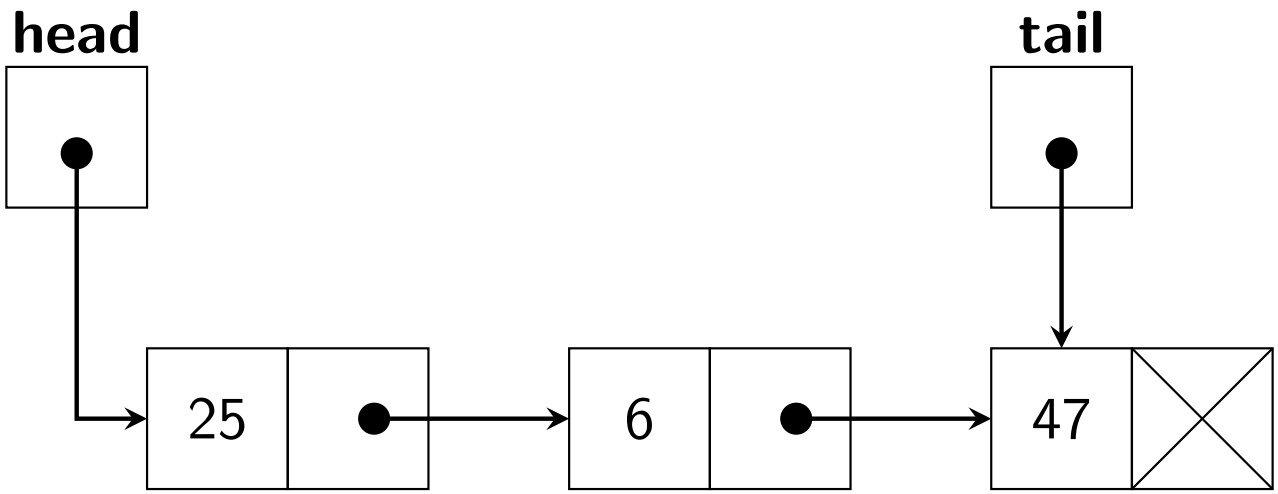


Search and other methods: Implementation in C++

```
1  class IntSLinkedList : public IList<int> {
2  // Declaration of attributes.
3  // Declaration of constructor, destructor.
4  // Declaration of nested classes.
5  public:
6      virtual int get(int index);
7      virtual void set(int index, int element);
8      virtual int indexOf(int item);
9      virtual bool contains(int item);
10 };
```

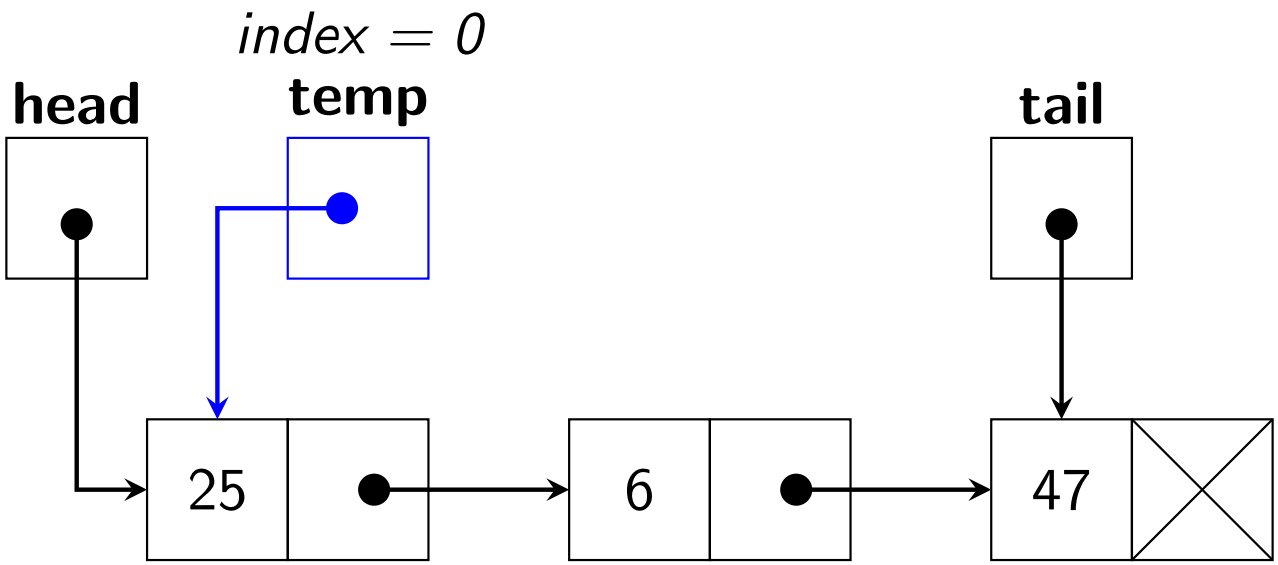

Searching

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



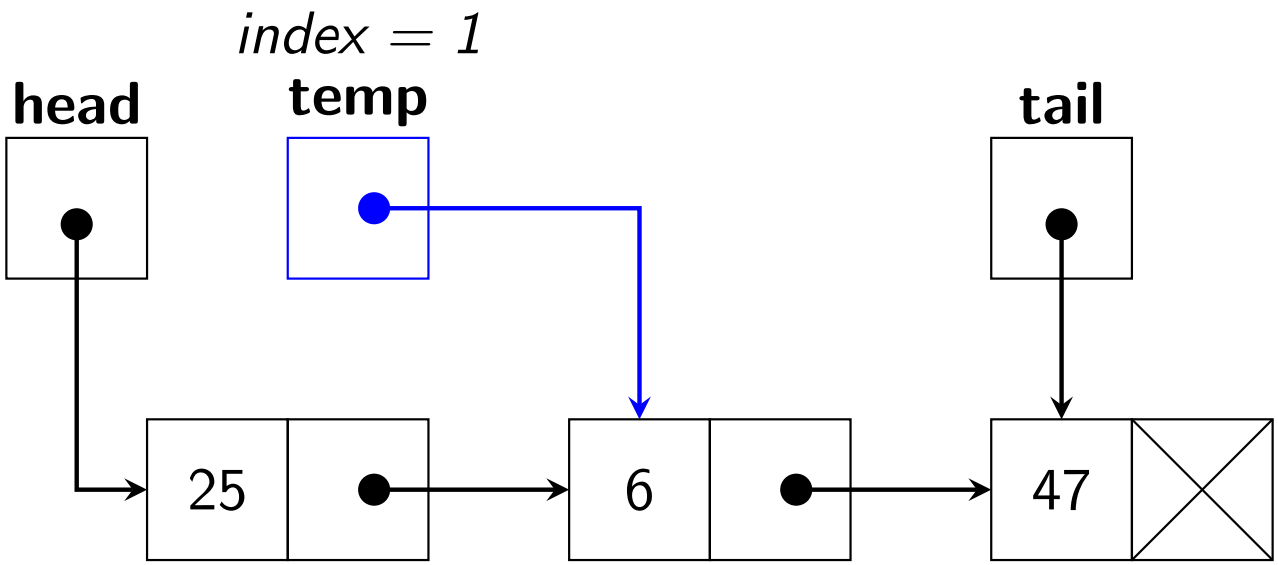
Searching

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



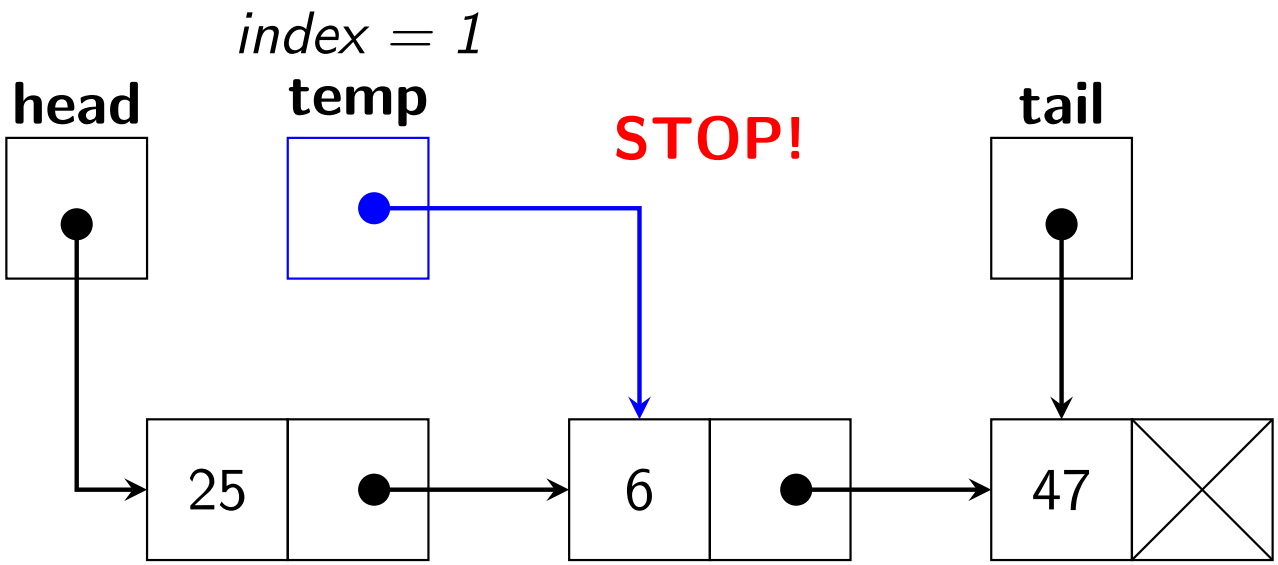
Searching

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



Searching

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



Dept. Computer Science

Comparison of implementations of list

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.

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.

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.

THANK YOU.