Data Structures
Lists and List ADT

CS 225
Brad Solomon & G Carl Evans

August 25, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

Define the functions and operations of the List ADT

Discuss list implementation strategies

Explore how to code and use a linked list

Practice fundamentals of C++ in the context of lists

Pointer-to-constant vs constant pointer

```
int x = 3;
int y = 2;
// *** A ***
const int* a = &x;

a = &y;

// *** B ***
const int* b = &x;

*b = y;

// *** C ***
int* const c = &x;

c = &y;

// *** D ***
int* const d = &x;

*d = y;
```

```
(const int)* a = &x;

(const int)* b = &x;

(int*) const c = &x;

(int*) const d = &x;
```

What types of “stuff” do we want in our list?

Templates

template1.cpp

```


T maximum(T a, T b) {
  T result;
  result = (a > b) ? a : b;
  return result;
}
```

Abstract Data Types

A way of describing a data type as a combination of:

- **Data** being stored by the data type
- **Operations** that can be performed on the data type

The actual implementation details of the ADT aren’t relevant!

List ADT (What do we want our list to do?)

List ADT

A list is an **ordered** collection of items

  Items can be either **heterogeneous** or **homogenous**

  The list can be of a **fixed size** or is **resizable**

A minimal set of operations (that can be used to create all others):

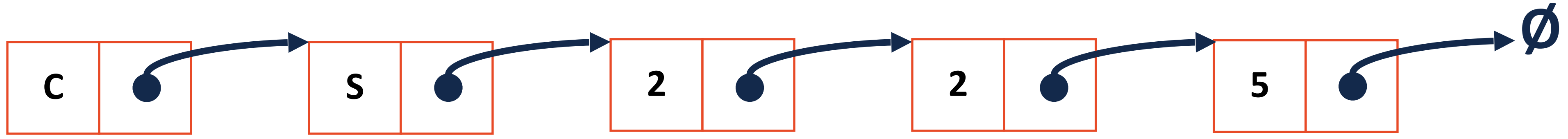
1. Insert
2. Delete
3. isEmpty
4. getData
5. Create an empty list

List Implementations

1.

2.

Linked List



List.h

```
class ListNode {
  T & data;
  ListNode * next;
  ListNode(T & data) : data(data), next(NULL) { }
};
```

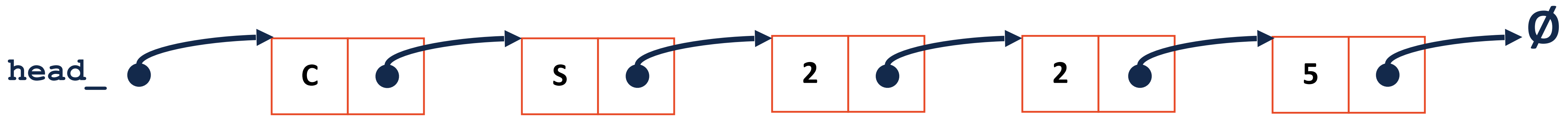
List.h

```cpp
#pragma once

template <class T>
class List {
  public:
    /* ... */
  private:
    class ListNode {
      T & data;
      ListNode * next;
      ListNode(T & data) :
       data(data), next(NULL) { }
    };

    ListNode *head_;

    /* ... */

};


#include "List.hpp" // **A**
```

List.hpp

```cpp
#include "List.h" // **B**

template <typename T>
void List<T>::insertAtFront(const T& t)  {

















}
```

Linked List: `insertAtFront(data)`



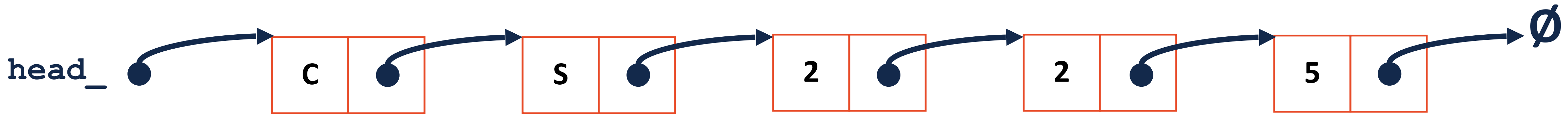
List.h

```cpp
#pragma once

template <class T>
class List {
  public:
    /* ... */
  private:
    class ListNode {
      T & data;
      ListNode * next;
      ListNode(T & data)  :
       data(data), next(NULL) { }
    };

    ListNode *head_;

    /* ... */

};


#include "List.hpp"
```

List.hpp

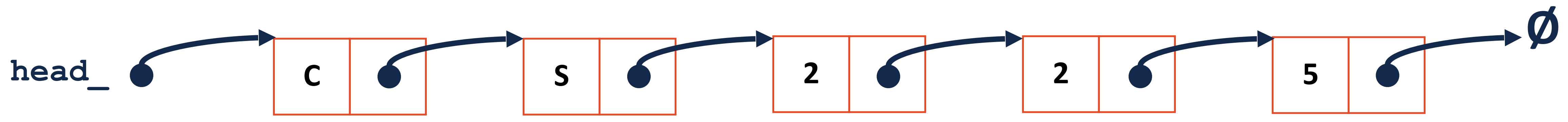
```cpp


template <typename T>
void List<T>::insertAtFront(const T& t)
{



  ListNode *tmp = new ListNode(data);



  tmp->next = head_;



  head_ = tmp;





}
```

Linked List: `insert(data, index)`



Linked List: `_index(index)`

What should the return type of `_index()` be?



```
[template <class T>]
```

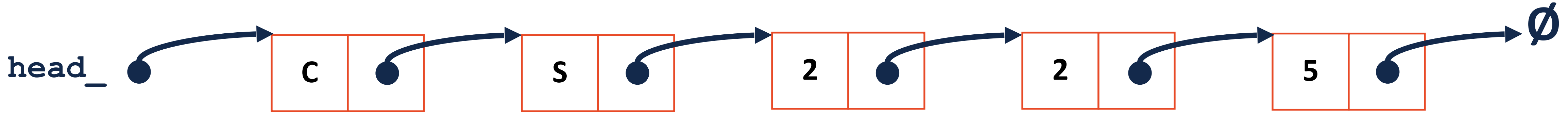
(A) `T &`

(B) `ListNode`

(C) `ListNode *`

(D) `ListNode *&`

Linked List: `_index(index)`



List.hpp

```
template <typename T>
typename List<T>::ListNode *& List<T>::_index(unsigned index){
      return _index(index, head_)
}
```

```
template <typename T>
typename List<T>::ListNode *& List<T>::_index(unsigned index, ListNode *& root){









}
```

List.hpp

```
template <typename T>
typename List<T>::ListNode *& List<T>::_index(unsigned index){
      return _index(index, head_)
}
```

```
template <typename T>
typename List<T>::ListNode *& List<T>::_index(unsigned index, ListNode *& root){

if (index == 0 || node == nullptr){
    return node;
}

return _index(index - 1, root -> next);


}
```

List.hpp

```cpp
// Iterative Solution:
template <typename T>
typename List<T>::ListNode *& List<T>::_index(unsigned index) {
  if (index == 0) { return head; }
  else {
    ListNode *thru = head;
    for (unsigned i = 0; i < index - 1; i++) {
       thru = thru->next;
    }
    return thru->next;
  }
}
```

What is the running time for iterative index?

What is the running time for recursive index?

List.hpp

```
template <typename T>
void List<T>::insertAtFront(const T& t)
{
  ListNode *tmp = new ListNode(data);

  tmp->next = head_;

  head_ = tmp;

}
```

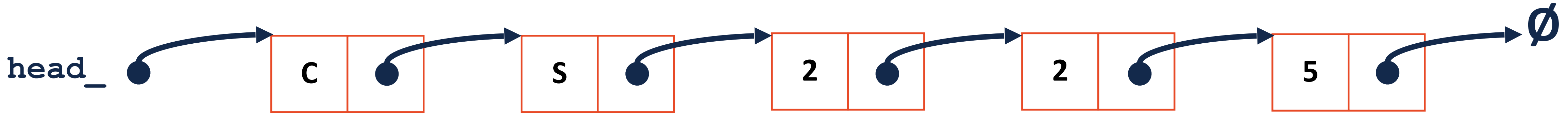
```
template <typename T>
void List<T>::insert(const T & data,
unsigned index) {



  ListNode *& curr = _index(index);



  ListNode * tmp = new ListNode(data);



  tmp->next = curr;



  curr = tmp;
}
```

List Random Access [ ]

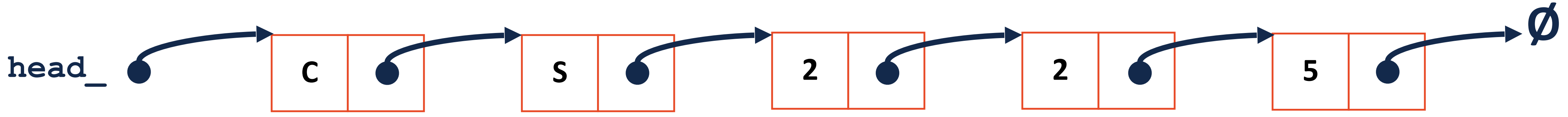
Given a list L, what operations can we do on L[ ]?

List.hpp

```
template <typename T>
T & List<T>::operator[](unsigned index) {










}
```



Linked List: `find(data)`



Given a Linked List, what should find return?