String Algorithms and Data Structures
Lists and Lists ADT

CS 225
Brad Solomon

February 4, 2022

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

Define the list abstract data type

Discuss implementation strategies for lists

Develop code for core functions in ADT

Abstract Data Types

A conceptual model that defines how we can interact with an object

Says nothing about implementation details!

Abstract Data Types

A conceptual model that defines how we can interact with an object

Says nothing about implementation details!

PROLOGUE

Everyone my age remembers where they were and what they were doing when they first heard about the contest. I was sitting in my hideout watching cartoons when the news bulletin broke in on my video feed, announcing that James Halliday had died during the night.

I'd heard of Halliday, of course. Everyone had. He was the videogame designer responsible for creating the OASIS, a massively multiplayer online game that had gradually evolved into the globally networked virtual reality most of humanity now used on a daily basis. The unprecedented success of the OASIS had made Halliday one of the wealthiest people in the world.

At first, I couldn't understand why the media was making such a big deal of the billionaire's death. After all, the people of Planet Earth had other concerns.

List ADT

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

Today’s List ADT

```
List();

void insert(const T &data);

void remove();

T getData(unsigned i) const;

bool isEmpty() const;
```

Coding the List ADT

```
class ListString{
  public:
    List();
    void insert(const std::string &data);
    void remove();
    std::string getData(unsigned i) const;
    bool isEmpty() const;
};

```

```
class ListInt{
  public:
    List();
    void insert(const int &data);
    void remove();
    int getData(unsigned i) const;
    bool isEmpty() const;
};

```

Templates

Recipes for code that the compiler uses to make our code

Uses types as variables

template1.hpp

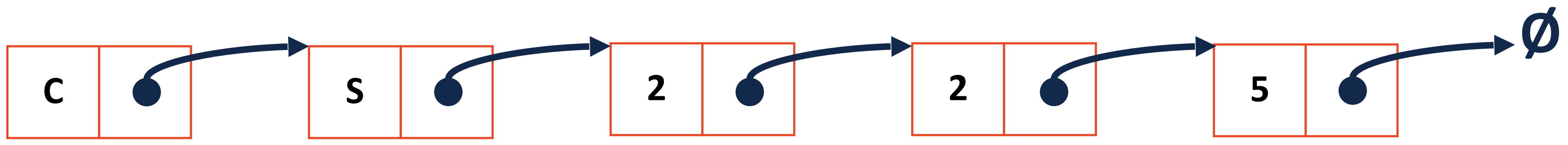
```
template <typename T>
T maximum(T a, T b) {
  T result;
  result = (a > b) ? a : b;
  return result;
}

```

Coding the List ADT

```
template <typename T>
class List{
    public:
        List();
        void insert(const T &data);
        void remove();
        T getData(unsigned i) const;
        bool isEmpty() const;
};
```

List Implementations

Linked List



Linked List

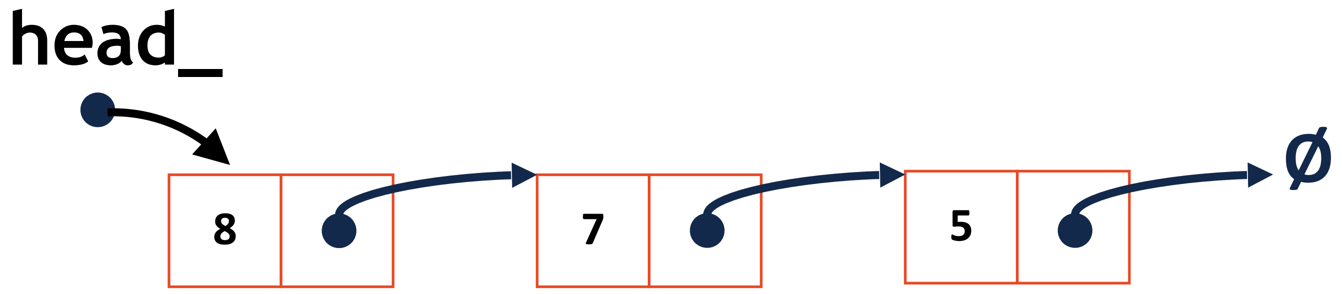
```
template <typename T>
class List {
  public:
    List();
    void insert(const T &data);
    void remove(unsigned i);
    T getData(unsigned i) const;
    bool isEmpty() const;

  private:












};
```

Linked List

```
template <typename T>
class List {
  public:
    List() : head_(nullptr) { }
    void insert(const T &data);
    void remove(unsigned i);
    T getData(unsigned i) const;
    bool isEmpty() const;

  private:
    class ListNode {
      public:
        T data;
        ListNode * next;
        ListNode(const T & data) :
        data(data), next(nullptr) { }
    };

    ListNode *head_;
};

```

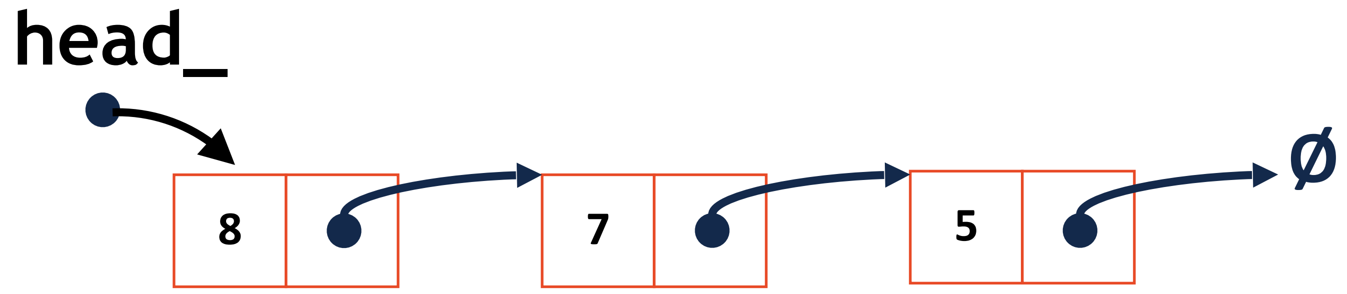
Linked List

`isEmpty()`



Linked List

Insert(1)



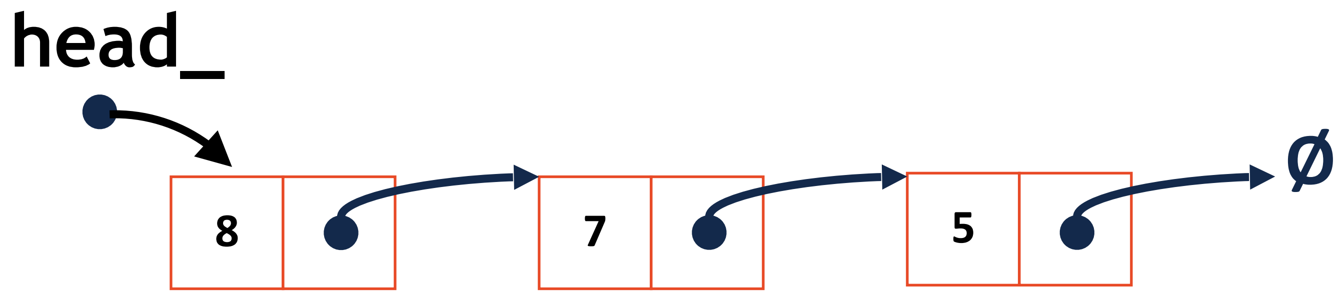
Linked List Insert (at front)

```
template <typename T>
class List {
  public:
    List() : head_(nullptr) { }
    void insert(const T &data);
    void remove(unsigned i);
    T getData(unsigned i) const;
    bool isEmpty() const;

  private:
    class ListNode {
      public:
        T data;
        ListNode * next;
        ListNode(const T & data) :
        data(data), next(nullptr) { }
    };

    ListNode *head_;
};

```

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

  ListNode *node = new ListNode(data);

  node->next = head_;

  head_ = node;

}
```

Linked List

Remove()

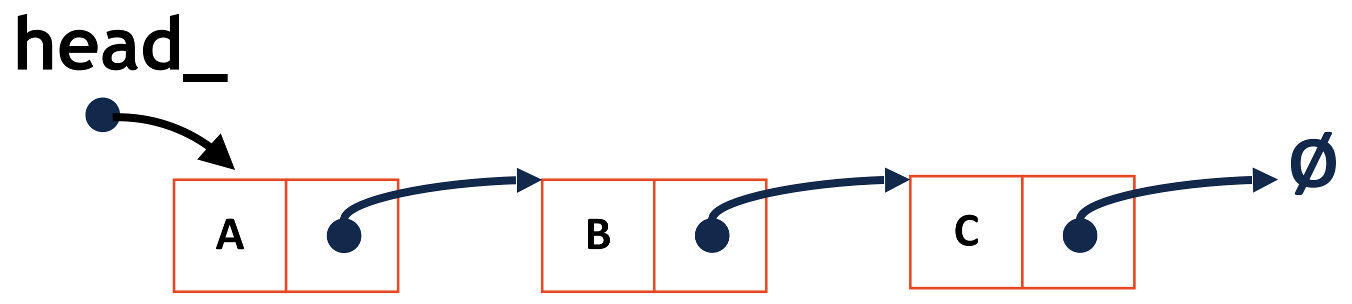


Linked List Remove (from front)

```
template<typeName T>
void List<T>::remove(){
    ListNode *tmp = _head;

    _head = _head->next;

    delete tmp;
}
```

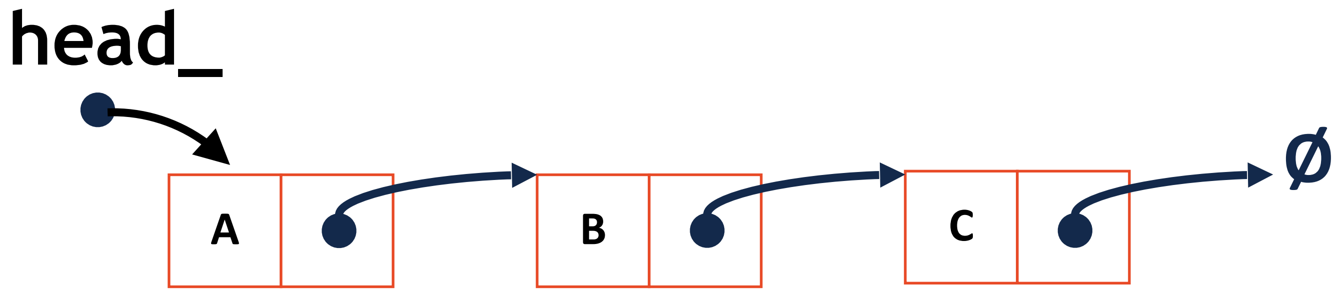
Linked List

`getData(2)`



Linked List

getData(2)



```
template <typename T>

T List<T>::getData(unsigned i) const{












}
```

Linked List getData

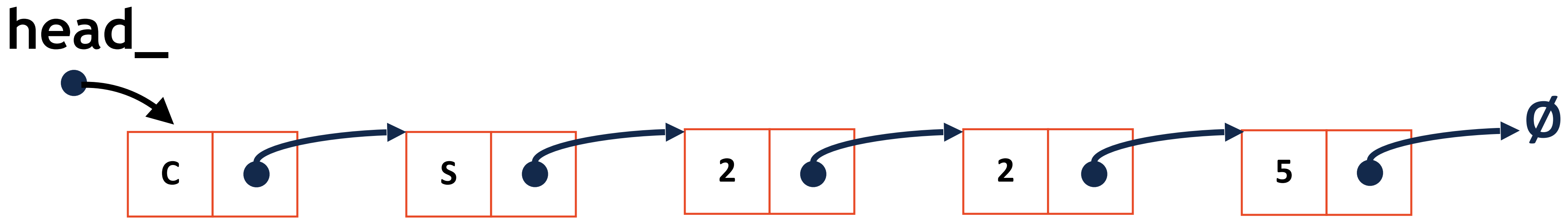
```
template <typename T>
T List<T>::getData(unsigned i) const{
    ListNode *temp = head_;
    while( i > 0 && temp->next != nullptr){
        temp = temp->next;
        i--;
    }
    if( i > 0 ){ return NULL; }
    return temp->data;
}
```

Linked List

```cpp
template <typename T>
class List {
  public:
    List() : head_(nullptr) { }
    void insert(const T &data);
    void remove();
    T getData(unsigned i) const;
    bool isEmpty() const;

  private:
    class ListNode {
      public:
        T data;
        ListNode * next;
        ListNode(const T & data) :
        data(data), next(nullptr) { }
  };

  ListNode *head_;
};

```

```cpp
void List<T>::insert(const T & data) {
  ListNode *node = new ListNode(data);
  node->next = head_;
  head_ = node;
}

void List<T>::remove(){
  ListNode *tmp = _head;
  _head = _head->next;
  delete tmp;
}

T List<T>::getData(unsigned i) const{
  ListNode *temp = head_;
  while( i > 0 && temp->next != nullptr){
    temp = temp->next;
    i--;
  }
  if( i > 0 ){ return NULL; }
  return temp->data;
}
```

A better List ADT



What if I want to add the letter "i" after "s"?

What if I want to only remove the last listNode?

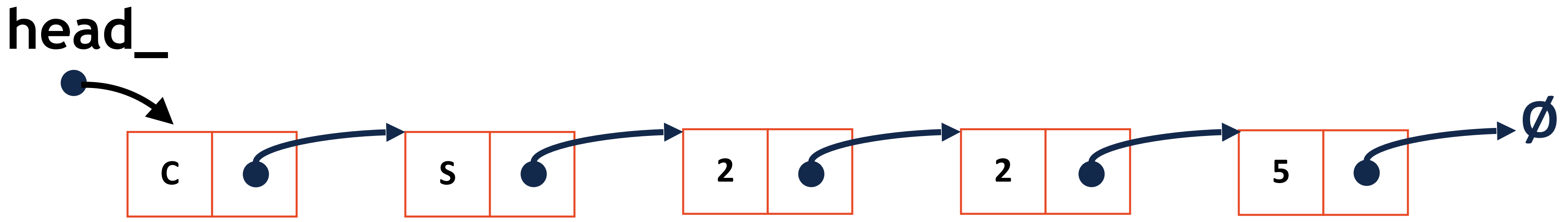
What if I want to be able to find a listNode rather than a value?

A better List ADT

```
List();

void insert(unsigned i, const T &data);

void remove(unsigned i);

ListNode*& _index(unsigned i);

T & List<T>::operator[](unsigned index);
```

Linked List

Insert(2,"I")



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





  ListNode *node = new ListNode(data);

  node->next = head_;

  head_ = node;

}
```