CS24 Week 4 Lecture 1

Kyle Dewey
Overview

- Additional use of `const` in C++
- List ADT
  - Array Lists
  - Linked Lists
Additional Use of `const`

- We’ve seen `const` already in two positions:

```c
void foo(const char* const s) {
    s[0] = 'a'; // disallowed
    s = NULL; // disallowed
}
```

What is pointed to is constant | The pointer itself is constant
Additional Use of `const`

- We can also tag whole methods with `const`, indicating that they may not change any state of the class they are called on.
- Great for accessors, as opposed to mutators.

```cpp
class Foo {
  public:
    Foo(int a) { b = a; }
    void setValue(int a) { b = a; }
    int getValue() const { return b; }
  private:
    int b;
};
```
this
this

• Allows one to refer to the object being acted upon in a method call, via a pointer

class Foo {
    private:
        int x;
    public:
        int getX() {
            return this->x;
        }
};
List ADT
Motivation

- We often work with a series of items
  - Addresses in a phone book, cards in a deck, etc.

- Arrays can be painful
  - Fixed size
  - Error-prone (e.g., index too large)
  - Repeated similar operations
Idea: A “List” ADT

- Handles the storage of elements and the addition of elements
- Holds common operations (e.g., checking if an item is contained within)
- Can protect against out-of-bounds
A List ADT

- What should the List ADT have at the logical/abstract/interface level?
A List ADT

- What should the List ADT have at the logical/abstract/interface level?
  - Basic examples: get item, add item, insert item at a position, remove item, get size
  - Many, many more examples possible
Idealized List ADT

List emptyList();
int getSize();
int getInt(int position);
bool containsInt(int item);
void addInt(int item);
void addIntAtPosition(int item, int position);
void removeFirstInt(int item);
Implementing in C++

- Classes? Constructors? Methods?
- Which methods should be marked `const`?

List emptyList();
int getSize();
int getInt(int position);
bool containsInt(int item);
void addInt(int item);
void addIntAtPosition(int item, int position);
void removeFirstInt(int item);
Implementing in C++

• Classes? Constructors? Methods?

• Which methods should be marked const?

List emptyList(); // Constructor
int getSize() const;
int getInt(int position) const;
bool containsInt(int item) const;
void addInt(int item);
void addIntAtPosition(int item, int position);
void removeFirstInt(int item);
Implementing in C++

- For now, let’s implement this via an array
- What other issues are present because of this design decision?
Implementing in C++

• For now, let’s implement this via an array
• What other issues are present because of this design decision?
  • Size of the array?
  • Accessing out-of-bounds element?
  • Adding an element in the middle?
• How might we handle each?
Implementation in C++
Array-Based List

- What sort of operations were hard because arrays were used?
Array-Based List

- What sort of operations were hard or awkward because arrays were used?
- Constructor needed an array size
- Adding an element at an arbitrary position required pushing elements to the right
- Removing an element required pushing elements to the left
Other Approaches

- How might we improve on these issues? (Fixed size, making arbitrary addition and removal easier)
Other Approaches

• How might we improve on these issues? (Fixed size, making arbitrary addition and removal easier)

• Wide variety of answers

• Approach we will take: linked lists
Fixed Size

• Observation: with arrays, we must allocate in blocks

• We must pre-allocate room, and expanding this room is obnoxious

• We would like to allocate as we go along, in a piecewise fashion
Piecewise Allocation

• How can we represent the list in a way that makes piecewise allocation possible? (Not just extending onto an array)
Piecewise Allocation

• How can we represent the list in a way that makes piecewise allocation possible? (Not just extending onto another array)
  • Piecewise implies separate chunks that hold onto single elements
• How do we keep track of chunks?
Linked Lists

- Idea: have each chunk (called a node) keep track of both a list element and another chunk

- Need to keep track of only the head node

List: 0, 1, 2, 3

Diagram:

```plaintext
0 -> 1 -> 2 -> 3 -> X
```
Node Representation

- What might a node look like in C/C++?
Node Representation

- What might a node look like in C?

```c
struct Node {
    int item;
    struct Node* next;
};
```
Node Representation

• What might a node look like in C++?

class Node {
  public:
    Node(int i, Node* n);
    int getItem() const;
    void setItem(int i);
    Node* getNext() const;
    void setNext(Node* n);
  private:
    int item;
    Node* node;
};
C++ Implementation of Linked Lists