ECE 2574: Data Structures and Algorithms - List: Array Implementations

C. L. Wyatt

Today we will look at how to use a dynamically allocated array internally to implement the List ADT.

- Warmup
- Review of the ADT
- Memory management revisited: rule of 3 and copy-swap idiom
- implementing the interface

Warmup

When should the copy constructor be written for a class?

- Always 3%
- For any class that uses dynamic allocation 62%
- For any class with a non-trivial destructor 36% (most correct)
- Never 0%

The Ordered List ADT and its abstract interface in C++

- Test if a list is empty

```
+isEmpty(): boolean
```

becomes

```
bool isEmpty() const
```

The Ordered List ADT and its abstract interface in C++

- Get the number of entries in the list

```
+getLength(): integer
```

becomes

```
std::size_t getLength() const
```

The Ordered List ADT and its abstract interface in C++

- Insert an entry at a given position in the list

```
+insert(newPosition: integer, newEntry: ItemType) :
boolean
becomes
bool insert(std::size_t position, const T& item)
```

The Ordered List ADT and its abstract interface in C++

- Remove entry at given position from the list

```
+remove(position: integer): boolean
```

becomes

```
bool remove(std::size_t position)
```

The Ordered List ADT and its abstract interface in C++

- remove all entries (clear)

```
+clear(): void
```

becomes

```
void clear()
```

The Ordered List ADT and its abstract interface in C++

- get a copy of the item at a given position

`+getEntry(position: integer): ItemType`
becomes
`T getEntry(std::size_t position) const`
with the possibility of throwing `std::range_error`.
What would be the implications of returning `T&` instead?

The Ordered List ADT and its abstract interface in C++

- replace the value of the item at a given position

`+setEntry(position: integer, newValue: ItemType): void`
becomes
`void setEntry(std::size_t position, const T& newValue)`
with the possibility of throwing `std::range_error`

Memory management: rule of 3 and copy-swap idiom

Since the dynamic array implementation requires managing memory we need to implement

- Destructor
- Copy Constructor
- Copy Assignment

A dynamically allocated array implementation of AbstractList

see inclass code.

Next Actions and Reminders

- Read CH pp. 272-286
- No warmup for Monday
- Note: Today is the last day to Drop.