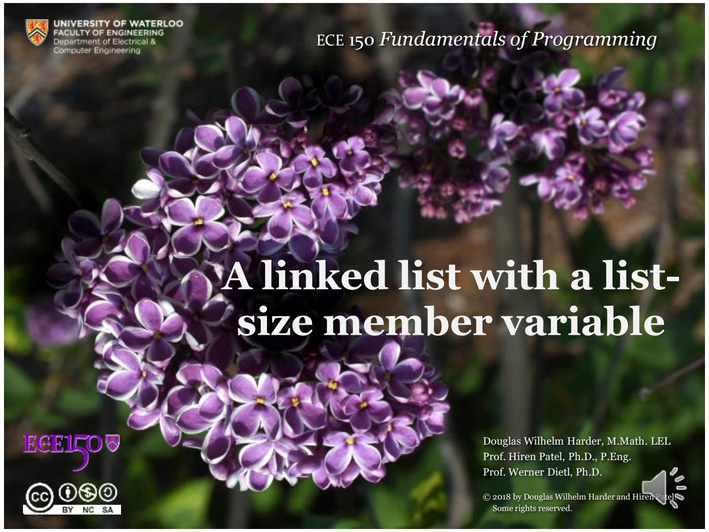


1

Outline

- In this lesson, we will:
  - Describe how to add a member variable to record the size
  - See that there are superior ways of authoring code
  - Explain how leaving an object in an inconsistent state is something that must be minimized in time

2

Initializing the list size

- Currently, it takes a loop to count the number of nodes in a linked list
  - It should be possible to track the list size with a member variable
- Such a member variable should be:
  - Initially set to zero, as the linked list is empty
  - This member variable will only ever be modified in any member function that changes the number of nodes in a linked list:

```
push_front(…);
pop_front();
clear();
```

  - The `clear()` member function already calls `pop_front()`, so we need only consider the first two

3

Initializing the list size

- First, we add the member variable

```
class Linked_list {
    public:
        // Public constructors, member functions, and destructor
    private:
        Node *p_list_head_;
        std::size_t list_size_;
};
```

- An initial linked list is empty, so the initial size is zero

```
Linked_list::Linked_list():
p_list_head_{ nullptr },
list_size_{ 0 } {
    // Empty constructor
}
```

4

Simplifying push front

- Next, we must modify any function changing the number of nodes in the linked list:

```
void Linked_list::push_front( double new_value ) {
    p_list_head_ = new Node{ new_value, p_list_head_ };
    ++list_size_;
}

void Linked_list::pop_front() {
    if ( !empty() ) {
        Node *p_old_head{ p_list_head_ };
        p_list_head_ = p_list_head_->p_next_node();
        --list_size_;
        delete p_old_head;
        p_old_head = nullptr;
    }
}
```


Popping the front when not empty

- Note that all member variables are changed, as much as possible:
  - In the order they are defined
  - In the closest possible proximity to each other
- This allows other programmers examining the class to:
  - Become more comfortable and familiar with your design
    - This helps when you make a mistake
      - For example, forgetting to update one specific member variable
    - It helps when adding more member variables:
      - Anywhere that one member variable is changing may indicate where a new member variable may have to be updated
  - Avoid making mistakes by having as few statements as possible that leave the linked list be in an inconsistent state
    - That is, where, for example, the number of actual nodes in the linked list does not match the member variable `list_size_`

Simplifying push front

- Less ideal would be to switch order at a whim, and to change some member variables almost as an afterthought

```
void Linked_list::push_front( double new_value ) {
    ++list_size_;
    p_list_head_ = new Node{ new_value, p_list_head_ };
}

void Linked_list::pop_front() {
    if ( !empty() ) {
        Node *p_old_head{ p_list_head_ };
        p_list_head_ = p_list_head_->p_next_node();
        delete p_old_head;
        p_old_head = nullptr;
        --list_size_;
    }
}
```


Simplifying push front

- You can group and highlight statements changing the state:

```
void Linked_list::push_front( double new_value ) {
    // Begin critical code:
        p_list_head_ = new Node{ new_value, p_list_head_ };
        ++list_size_;
    // End critical code
}
void Linked_list::pop_front() {
    if ( !empty() ) {
        Node *p_old_head{ p_list_head_ };

        // Begin critical code:
            p_list_head_ = p_list_head_->p_next_node();
            --list_size_;
        // End critical code

        delete p_old_head;
        p_old_head = nullptr;
    }
}
```

Conclusions

- While it may seem all code is “okay” so long as it gets the job done, some approaches are better that others
- Programmers reading your code will be thankful, even if they have no clue who you are
  - You will be thankful yourself, especially if you haven’t looked at your own code for over a month

9

Summary

- Following this lesson, you now
  - Know that adding features can be straight-forward
  - Understand that adding a feature that significantly reduces the run-time can be very beneficial
  - Understand that how you code changes is relevant:
    - It is critical to leave the object in inconsistent states for as short a period as possible

10

References

[1] https://en.wikipedia.org/wiki/Linked_list
[2] https://en.wikipedia.org/wiki/Information_hiding#Encapsulation

11

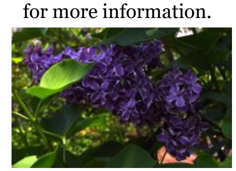
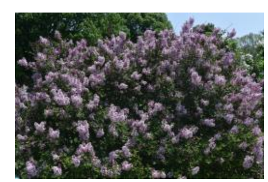
Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.


12

Disclaimer