UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

ECE 150 *Fundamentals of Programming*

Concatenating two linked lists

Douglas Wilhelm Harder, M.Math. LEL
Prof. Hiren Patel, Ph.D., P.Eng.
Prof. Werner Dietl, Ph.D.

1

Outline

- In this lesson, we will:
  - Describe how to concatenate two linked lists
  - Consider the appropriate names for the member functions
  - Walk through the steps of determining what must be done
  - Implement the push front member function
  - Observe that we can much more easily implement push back
  - Consider what happens if we concatenate a linked list with itself

2

Pushing an entire linked list

- Currently, we can push a new node at either the front or the back of a linked list
  - Suppose, however, we want to concatenate (insert) an entire linked list either at the front or the back
- The appropriate member functions would be:

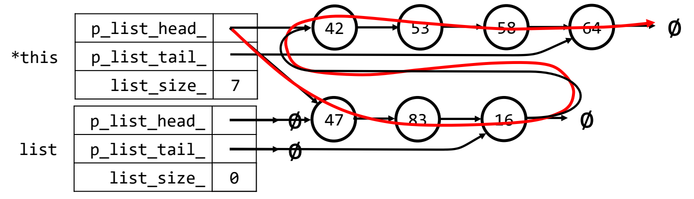
```
void push_front( Linked_list &list );
void push_back( Linked_list &list );
```

  - This will take all nodes out of the argument and push them at the appropriate location in this linked list
    - It will not copy the nodes

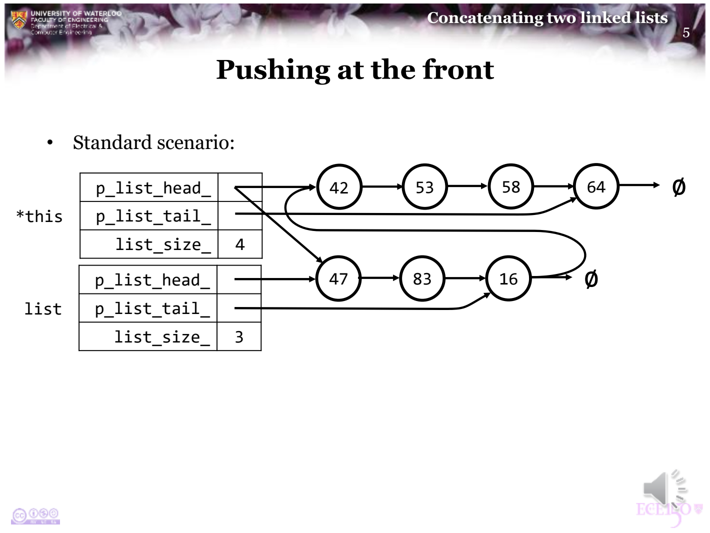
3

Pushing at the front

- General scenario:

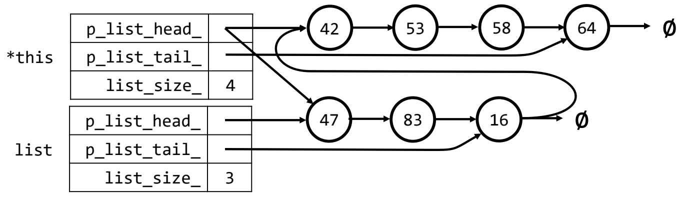


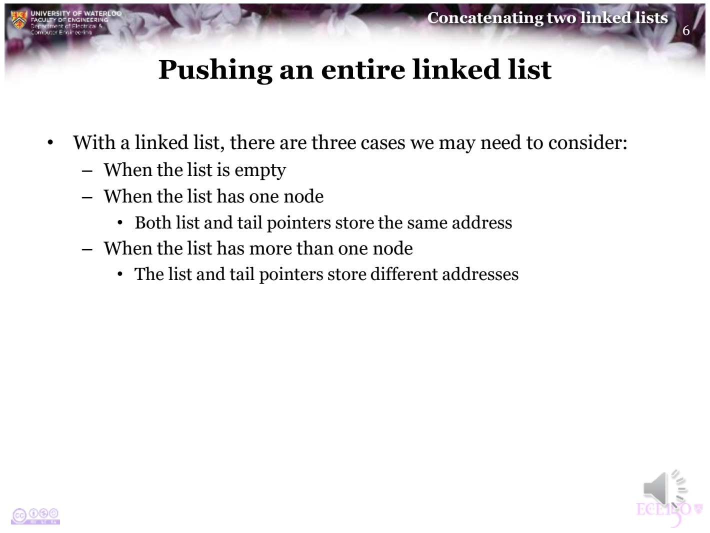
4



Pushing at the front

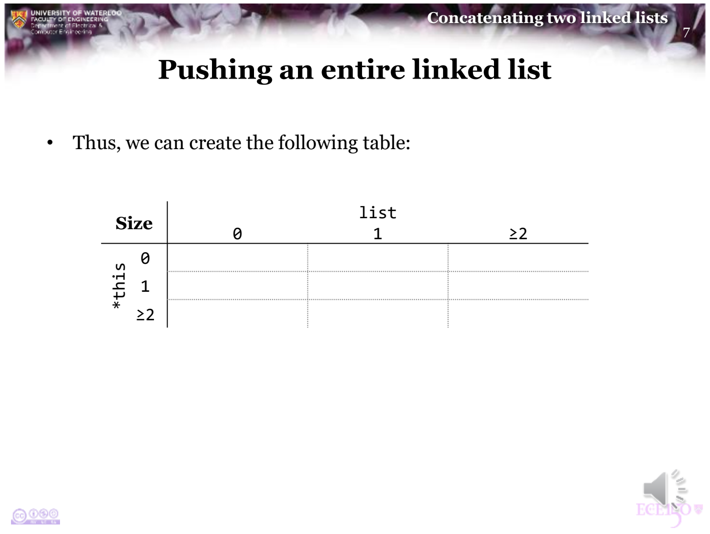
- Standard scenario:





Pushing an entire linked list

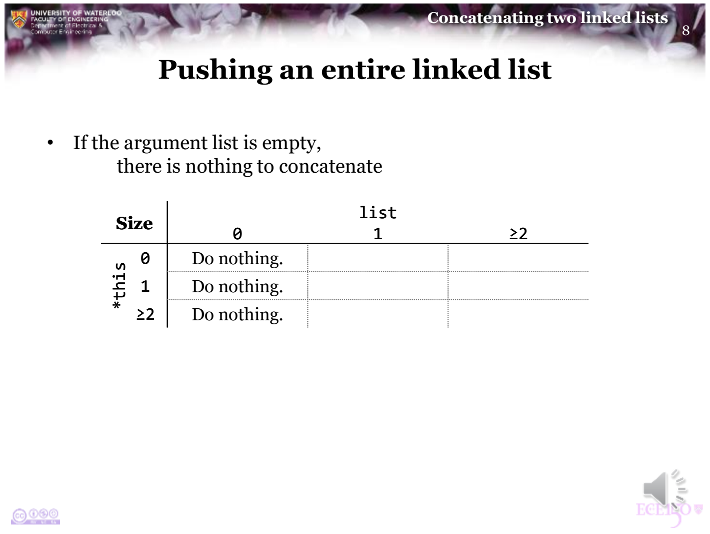
- With a linked list, there are three cases we may need to consider:
  - When the list is empty
  - When the list has one node
    - Both list and tail pointers store the same address
  - When the list has more than one node
    - The list and tail pointers store different addresses



Pushing an entire linked list

- Thus, we can create the following table:

Size		list 0	list 1	list ≥2
*this	0			
*this	1			
*this	≥2			



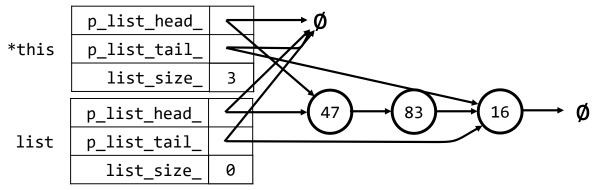
Pushing an entire linked list

- If the argument list is empty,
  there is nothing to concatenate

Size		list 0	list 1	list ≥2
*this	0	Do nothing.		
*this	1	Do nothing.		
*this	≥2	Do nothing.		

Pushing an entire linked list

- What if this list is empty?
  - The concatenated linked list is just the other list…
  - How about swapping all the member variables?



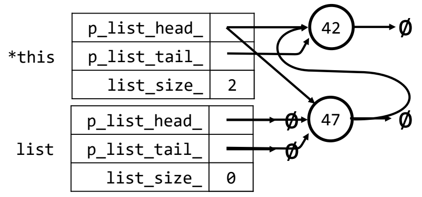
Pushing an entire linked list

- If this list is empty,
  swap the member variables of the two

Size		list 0	1	≥2
*this	0	Do nothing.	Swap	Swap
	1	Do nothing.		
	≥2	Do nothing.		

Pushing at the front

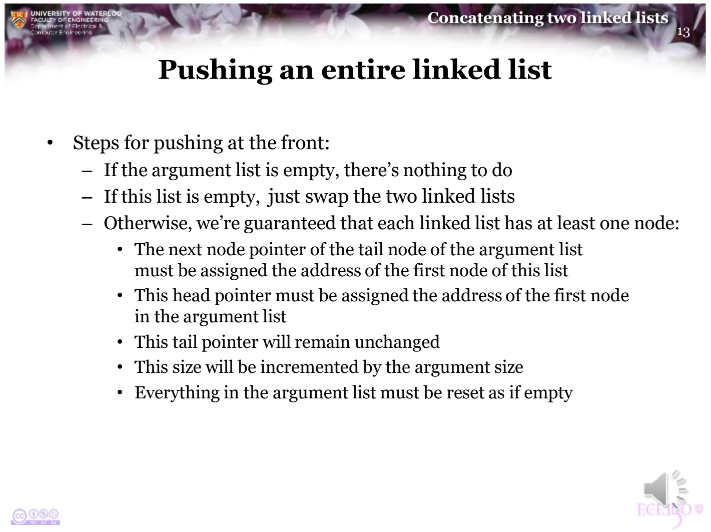
- One node each



Pushing an entire linked list

- If both lists are non-empty:
  assign member variables as described

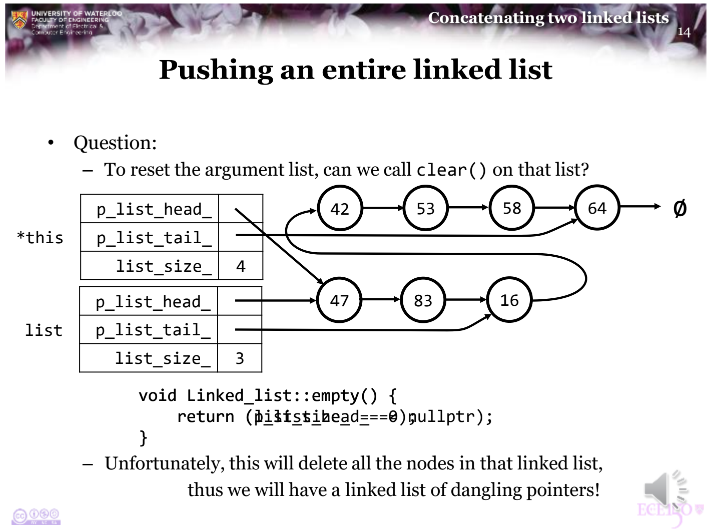
Size		list 0	1	≥2
*this	0	Do nothing.	Swap	Swap
	1	Do nothing.	General case.	General case.
	≥2	Do nothing.	General case.	General case.



Pushing an entire linked list

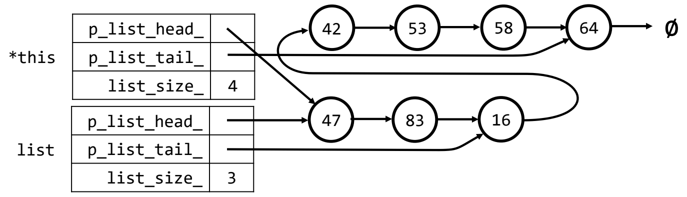
- Steps for pushing at the front:
  - If the argument list is empty, there's nothing to do
  - If this list is empty, just swap the two linked lists
  - Otherwise, we're guaranteed that each linked list has at least one node:
    - The next node pointer of the tail node of the argument list must be assigned the address of the first node of this list
    - This head pointer must be assigned the address of the first node in the argument list
    - This tail pointer will remain unchanged
    - This size will be incremented by the argument size
    - Everything in the argument list must be reset as if empty

13



Pushing an entire linked list

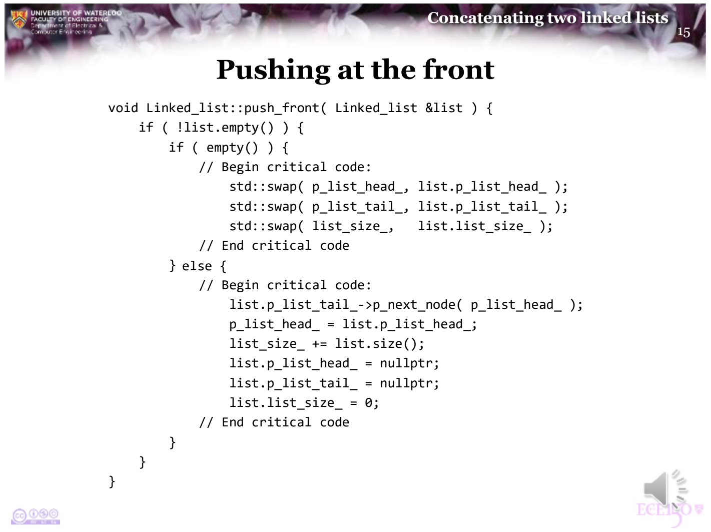
- Question:
  - To reset the argument list, can we call `clear()` on that list?



```
void Linked_list::empty() {
    return (p_list_head_ == nullptr);
    return (list_size_ == 0);
}
```

  - Unfortunately, this will delete all the nodes in that linked list, thus we will have a linked list of dangling pointers!

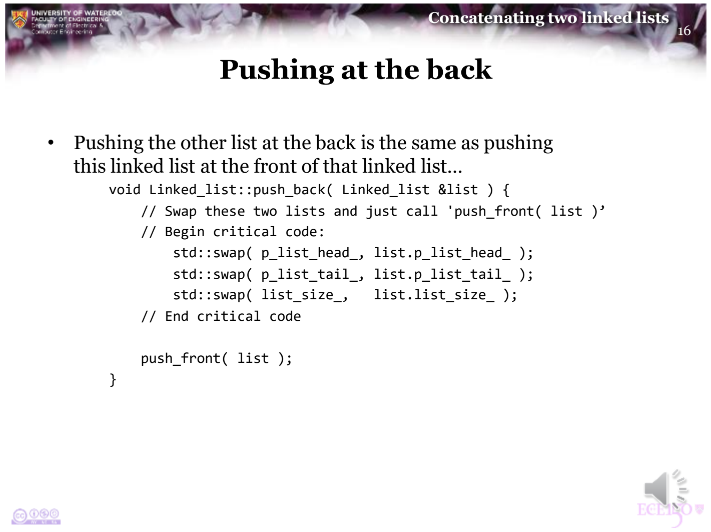
14



Pushing at the front

```
void Linked_list::push_front( Linked_list &list ) {
    if ( !list.empty() ) {
        if ( empty() ) {
            // Begin critical code:
                std::swap( p_list_head_, list.p_list_head_ );
                std::swap( p_list_tail_, list.p_list_tail_ );
                std::swap( list_size_,   list.list_size_ );
            // End critical code
        } else {
            // Begin critical code:
                list.p_list_tail_->p_next_node( p_list_head_ );
                p_list_head_ = list.p_list_head_;
                list_size_ += list.size();
                list.p_list_head_ = nullptr;
                list.p_list_tail_ = nullptr;
                list.list_size_ = 0;
            // End critical code
        }
    }
}
```

15



Pushing at the back

- Pushing the other list at the back is the same as pushing this linked list at the front of that linked list…

```
void Linked_list::push_back( Linked_list &list ) {
    // Swap these two lists and just call 'push_front( list )'
    // Begin critical code:
        std::swap( p_list_head_, list.p_list_head_ );
        std::swap( p_list_tail_, list.p_list_tail_ );
        std::swap( list_size_,   list.list_size_ );
    // End critical code

    push_front( list );
}
```

16

Swapping two linked lists

- You may have immediately noted that we just did the same operation twice in two different functions:

```
        // Begin critical code:
            std::swap( p_list_head_, list.p_list_head_ );
            std::swap( p_list_tail_, list.p_list_tail_ );
            std::swap( list_size_,   list.list_size_ );
        // End critical code
```

  – Should this not be a separate member function?

```
    void Linked_list::swap( Linked_list &list ) {
        // Begin critical code:
            std::swap( p_list_head_, list.p_list_head_ );
            std::swap( p_list_tail_, list.p_list_tail_ );
            std::swap( list_size_,   list.list_size_ );
        // End critical code
    }
```

  – Should it be private or public?

17

Swapping two linked lists

```
void Linked_list::push_front( Linked_list &list ) {
    if ( !list.empty() ) {
        if ( empty() ) {
            swap( list );
        } else {
            // Begin critical code:
                list.p_list_tail_->p_next_node( p_list_head_ );
                p_list_head_ = list.p_list_head_;
                list_size_ += list.size();
                list.p_list_head_ = nullptr;
                list.p_list_tail_ = nullptr;
                list.list_size_ = 0;
            // End critical code
        }
    }
}

void Linked_list::push_back( Linked_list &list ) {
    swap( list );
    push_front( list );
}
```

18

Did we miss something?

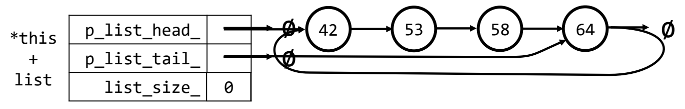
- Did we miss something critical?
  – Consider the following program:

```
    int main() {
        Linked_list data{};

        data.push_front( 72 );
        data.push_front( 98 );

        data.push_front( data );

        return 0;
    }
```

19

Did we miss something?

- This is going to be painful…



20

Did we miss something?

- What do we do if we try to push this linked list onto the front or back of this linked list?
  - Do we duplicate it?
    - Probably not, as this is supposed to be a fast function
  - Do we do nothing?
  - Do we through an invalid argument exception?

- In this case, I'd suggest an exception, because doing nothing is almost certainly not what the user wanted
  - The outcome of silently doing nothing gives an unexpected result
  - If one linked list is of size five, and the argument is also of size five, you expect the one to end up with size ten, and the other to be empty
  - In all likelihood, this was a programming error;
    either that, or the programmer should make an explicit copy first

Did we miss something?

- We can check for this as follows:

```
void Linked_list::push_front( Linked_list &list ) {
    if ( this == &list ) {
       throw std::invalid_argument{
            "You cannot push a list onto the front of itself" };
    } else if ( !list.empty() ) {
        // ...
    }
}

void Linked_list::push_back( Linked_list &list ) {
    swap( list );

    try {
        push_front( list );
    } catch ( std::invalid_argument &e ) {
        throw std::invalid_argument{
            "You cannot push a list onto the back of itself" };
    }
}
```

Summary

- Following this lesson, you now
  - Know how to concatenate two linked lists
    - We have pushed one linked list onto the front or back of this linked list
  - Have seen how to determine the appropriate steps
  - Have observed that sometimes there are
    interesting ways of reusing code
  - Understand that in some cases, you may have to be careful about performing an operation between this object and itself
  - Know how to avoid the issue, as well

References

[1] https://en.wikipedia.org/wiki/Linked_list
[2] https://en.wikipedia.org/wiki/Node_(computer_science)

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.

25

Disclaimer

26