1

Outline

- In this lesson, we will:
  - Describe a linked list class with no member variables
  - Look at how to implement a linked list class derived from this class
  - Consider the use of reference variables

2

Deep dive…

- This topic will now do a deep dive on combining so many different topics we have already discussed
  - We are going to create a linked list class that has no member variables
  - The linked list class we created will derive from this linked list class

3

Deep dive…

- In an operating system design, there is often already memory allocated for pointers to the heads of various linked lists
  - That user may want the functionality we've already implemented but may want to specify their own pointer to the list head
    - We want to give the option of the user specifying their own `p_list_head_` variable
  - This has horrible potential consequences for encapsulation, but if the user is willing to take the risk
- In other cases, we want the linked list to have its own `p_list_head_`

4

Deep dive...

- Let us define a `Base_linked_list` class as follows:
  - Everything is copied from `Linked_list`, with only the following changes:

```
class Base_linked_list {
    public:
        Base_linked_list( Node *&p_new_list_head );
        // All else identical except for name changes...

    private:
        Node *&ref_p_list_head_;

    // Friends are all similar except for name changes...
};
```

5

Deep dive...

- As for the constructor:

```
Base_linked_list::Base_linked_list( Node *&new_p_list_head ):
ref_p_list_head_{ p_new_list_head } {
    // Empty constructor
}
```

- In all other member functions, destructors, etc., change any use of `p_list_head_` to `ref_p_list_head_`

6

Deep dive...

- Now you can pass your own list head pointer:

```
int main() {
    Node my_head_ptr{};
    Base_linked_list my_list{ my_head_ptr };

    my_list.push_front(  4.2 );
    my_list.push_front( 10.1 );

    std::cout << my_list << std::endl;
    std::cout << my_head_ptr->value() << std::endl;
    std::cout << my_head_ptr->next_node()->value() << std::endl;

    return 0;
}
```

Output:

```
head -> 10.1 -> 4.2 -> nullptr
10.1
4.2
```

7

Deep dive...

- The linked list class may now be implemented as follow:

```
class Linked_list : public Base_linked_list {
    public:
        Linked_list();

    private:
        Node *p_list_head_;
};

Linked_list::Linked_list():
Base_linked_list{ p_list_head_ } {
    // Empty constructor
}
```

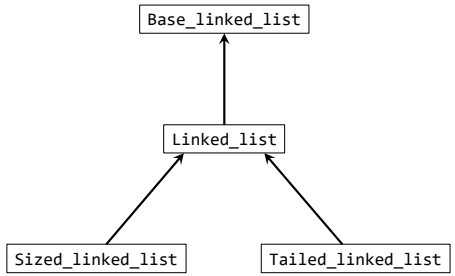
8

Overhead?

- What is the overhead of all this?
  - Fortunately, C++ compilers will usually optimize in such a way so as to make it no different from having this complex web of classes to having each class declared individually
- Consequently, it is almost always in your benefit to use inheritance

Showing the relationship

- To show the relationship between base classes and derived classes, we use the following format:



No memory?

- Actually, there is some memory involved:
  - Each object must occupy memory and information is needed about the class inheritance

```
int main() {
    Node *my_head{};

    std::cout << sizeof( Base_linked_list ) << std::endl;
    std::cout << sizeof( Linked_list ) << std::endl;

    return 0;
}
```

Output:
16
24

Summary

- Following this lesson, you now
  - Know of another example of inheritance
  - Have seen another use of reference variables
  - Understand that class inheritances can be extended in both directions

References

[1] https://en.wikipedia.org/wiki/Linked_list

[2] https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)#Subclasses_and_superclasses

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.



Disclaimer