

Operator Overloading, Linked Lists

CSC 112 Fall 2009

Overloading [] Operator

- Possible to overload [] (subscripting from arrays) operator
- Very useful for container style classes
- Usage examples:

```
Vector a; // assume Vector  
class is a class we are writing  
a[1] = 3; // write into Vector  
double coord2 = a[2]; //  
read from Vector
```

Overloading [] Operator

- Effectively could replace some of our get set methods
 - Vector – get/setValueForDimension methods

Overloading [] Operator

- [] is a member function, which takes an integer as a parameter and whose semantics is usually to return one of the types that compose the class type
- Have to be able use return value from [] operator as l-value for writes, so return the specified value by reference
 - Similar to assignment

Overloading [] Operator

- Standard implementation:

- header

```
returnType& operator[](int  
index);
```

- source

```
returnType&  
className::operator[](int index)  
{ return appropriate value; }
```

Overloading [] Operator

- What if indices given to [] are invalid?
 - What does that error mean for your type?
 - Do you want to handle it explicitly, and if so, how?
 - Since C++ doesn't handle it for the array container, should you?

Overloading [] Operator

■ const version:

- The subscript operator is often written a 2nd time as follows: `const returnType& operator[](int index) const;`
- First `const` says return data shouldn't be changed, second is a statement that subscript function doesn't change type it is being applied on
- Provides `const` semantics to be used if called on a `const` variable
- Example:

```
void f(const MyContainer& a)
{   a[3] = y;           a[3].mutate();   }
```
- The compiler, since `MyContainer` is `const`, will use the `const []` operator. This will raise a compiler error when it sees the attempt to modify `a`'s data through the `[]` operation (via assignment or use of another non-`const` function).

OOP: First Class Classes

- Really useful classes (for usage and programming) should have at least these five methods implemented:
 - Multi argument constructor
 - No argument constructor
 - Copy constructor
 - Assignment
 - Insertion (<<) operator

LinkedList Container

- Container:

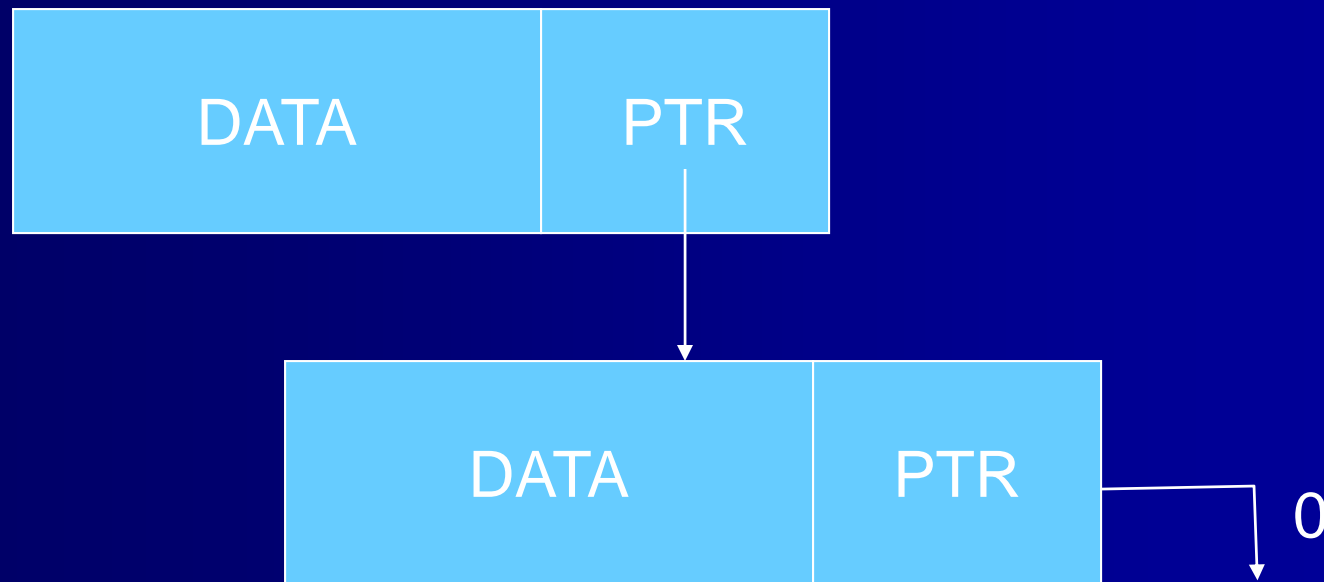
- holder of multiple pieces of data
- `array` is really our only example so far

- There are some problems with array based containers:

- Fixed in size at declaration
- Require large chunks of contiguous memory
- May over/under allocate, which requires re-allocating, copying, and deleting when discovered

Pointer-Based Implementations

- Imagine a type of data that:
 - Can hold data
 - Point to another piece of data



Pointer-Based Lists

- This structure
 - Can locate data in any place in memory, not required to be sequential
 - Has no requirements on computing size of list beforehand
 - Requires no more space than needed
 - Has no bounds on space besides physical memory
 - No requirements to resize
 - Still fairly trivial to manage and understand
 - Implementing interface similar to arrays
 - Tradeoff: direct access to spots is harder to come by