Chapter 18: Vector and Arrays

Plan for today

- We will talk about:
  - doubly linked list nodes (the in-class practice from previous meeting) (17.9.3)
  - initialization of vector objects
  - copy constructors (recall HW 7 assignment)
  - copy assignments
  - copy terminology
  - moving

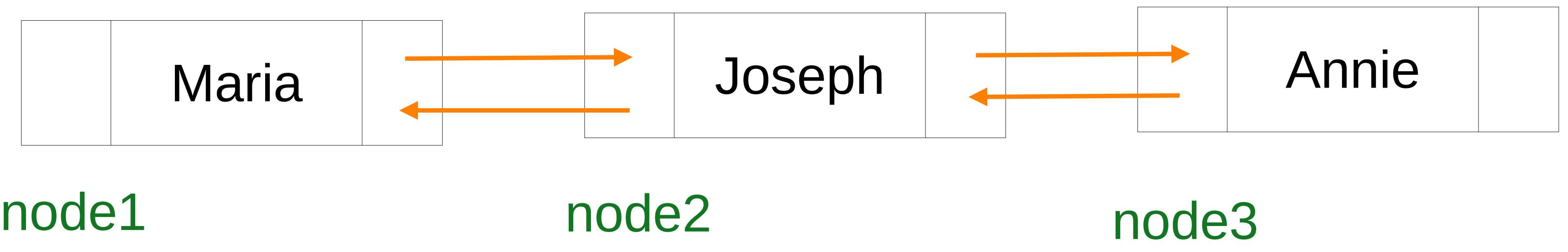
In-class practice from previous class

- Consider the following struct:

```
struct Link{
    string value;
    Link* prev;
    Link* succ;
    Link(const string& str, Link* p = nullptr,
            Link* s = nullptr):
        value{str}, prev{p}, succ{s}
    {}
};
```

In-class practice from previous class

- Let’s create the following connected list of those Links:



Vector class – what we have so far

```
class vector {
    int sz; // the size
    double* elem;// a pointer to the elements

public:
    vector(int s);  // constructor
    ~vector();      // destructor
    double get(int n) const;     // access:read
    void set(int n, double v);   // access:write
    int size() const;           // the current size

    // a member function that would display the values of the vector object
    void display() const;
    void resize(int newSz); // resizes to new size, copies the existing elements

    vector& operator=(const vector& other); // overloading the assignment operator, with chaining a = b = c
    void copy(const vector* other); // HW 7 assignment
};
std::ostream& operator<<(std::ostream& out, const vector& v); // overload opeartor<<
```

Initialization

- So far, when declaring an object of type vector, we supply the size:
  - `vector a(10); // create a vector of 10 elements`

Initialization

- So far, when declaring an object of type vector, we supply the size:
  - `vector a(10); // create a vector of 10 elements`
- What if we want to declare and initialize a vector object?
  - `vector a1 = {1,5,2,3,6,7};`
  - it is much better than initializing to default values, and then assigning the new values like `a[2]=5`

Initialization

- So far, when declaring an object of type vector, we supply the size:
  - `vector a(10); // create a vector of 10 elements`
- What if we want to declare and initialize a vector object?
  - `vector a1 = {1,5,2,3,6,7};`
  - it is much better than initializing to default values, and then assigning the new values like `a[2]=5`
- A { }-delimited list of elements of type `T` is presented to the programmer as an object of the standard library type `initializer_list<T>`, a list of `T`s

Initialization

- So far, when declaring an object of type vector, we supply the size:
  - `vector a(10); // create a vector of 10 elements`
- What if we want to declare and initialize a vector object?
  - `vector a1 = {1,5,2,3,6,7};`
  - it is much better than initializing to default values, and then assigning the new values like `a[2]=5`
- A { }-delimited list of elements of type `T` is presented to the programmer as an object of the standard library type `initializer_list<T>`, a list of `T`s, so we can write:

```
vector(std::initializer_list<double> lst)
           : sz(lst.size()), elem{ new double[sz]} {
           std::copy(lst.begin(), lst.end(), elem); }
```

Initialization: lists and sizes

- If we initialize a vector by 17 is it
  - 17 elements (with value 0)?
  - 1 element with value 17?
- By convention use
  - () for number of elements
  - {} for elements
- For example
  - `vector v1(17);    // 17 elements, each with the value 0`
  - `vector v2 {17};   // 1 element with value 17`

Copying

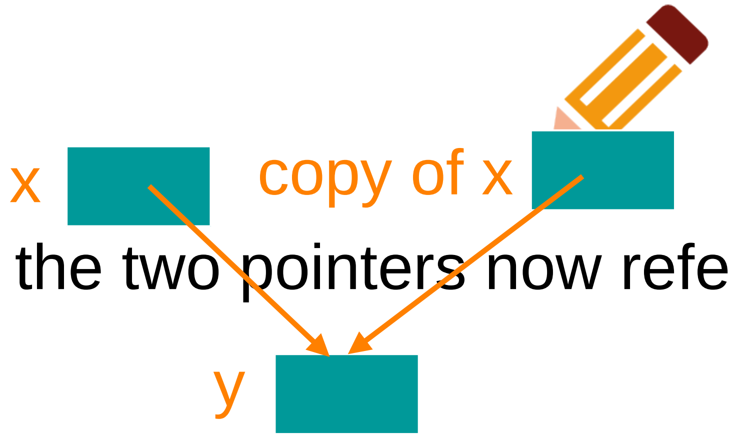
- Copy constructor
  - `vector(const vector& other);`
  - Examples:
    - `vector c{a1};`
    - `vector b = a1;`
  - The vector object is being created, so it's a "fresh start"
- Copy assignment
  - `vector& operator=(const vector& other);`
  - The vector object already exists, so we need to deal with the old elements

See vector.h and vectorTesting.cpp

Copy terminology

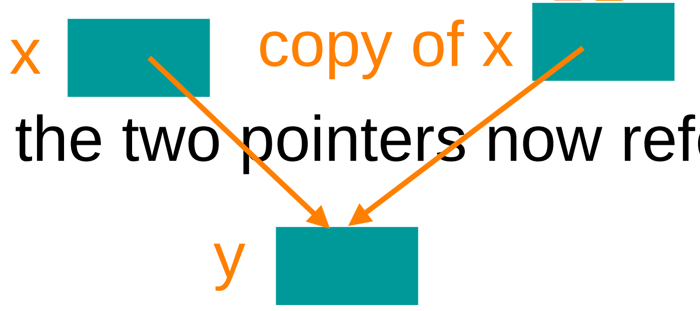
- Shallow copy: copy only a pointer so that the two pointers now refer to the same object
  - What pointers and references do

Copy terminology



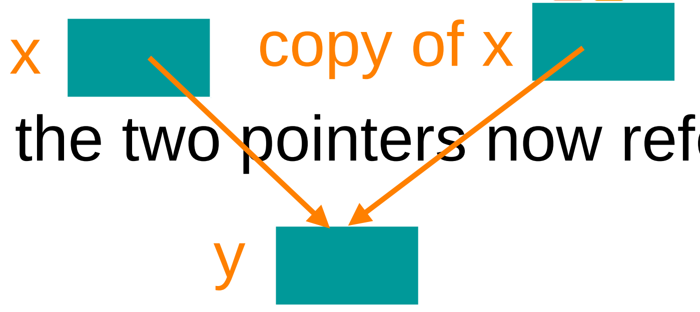
- Shallow copy: copy only a pointer so that the two pointers now refer to the same object
  - What pointers and references do

Copy terminology

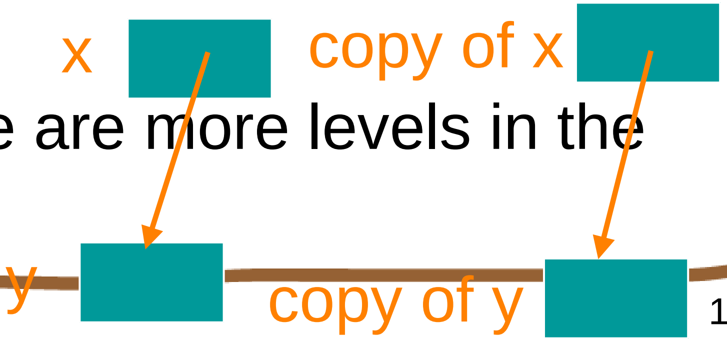


- Shallow copy: copy only a pointer so that the two pointers now refer to the same object
  - What pointers and references do
- Deep copy: copy what the pointer points to so that the two pointers now each refer to a distinct object
  - What vector, string, etc. do
  - Requires copy constructors and copy assignments for container classes
  - Must copy "all the way down" if there are more levels in the object

Copy terminology



- Shallow copy: copy only a pointer so that the two pointers now refer to the same object
  - What pointers and references do
- Deep copy: copy what the pointer points to so that the two pointers now each refer to a distinct object
  - What vector, string, etc. do
  - Requires copy constructors and copy assignments for container classes
  - Must copy “all the way down” if there are more levels in the object



Moving: move constructor and assignment

- If a vector has a lot of elements, it can be expensive to copy
- We can "move" (steal) information from one vector to another by defining move operations to complement copy operations:
  - `vector(vector&& a); // move constructor`
  - `vector& operator=(vector&& a);`
    `// move assignment`
  - && is called an "rvalue reference"
  - Note that we do not take **`const`** arguments, because our goal is to modify the source, to make it empty

Moving: move constructor

```
vector(vector&& a)  // move constructor
   :sz{a.sz}, elem{a.elem} // copy a’s elem and sz
{
   a.sz = 0;    // make a the empty vector
   a.elem = nullptr;
}
```

Moving: move assignment

```
vector& operator=(vector&& a)
{
    delete[] elem;      // deallocate old space
    elem = a.elem;      // copy a’s elem and sz
    sz = a.sz;
    a.elem = nullptr;// make a the empty vector
    a.sz = 0;
    return *this;     // return a self-reference
}
```

See vector.h and vectorTesting.cpp

Moving

- Using move constructor explicitly:

```
    vector x = std::move(a1);
```

- Using move assignment explicitly:

```
    b = x;
```

- We can use “moving” to implement keyboard input of vector elements (it’s not working yet, just an idea)

Resources used for these slides

- slides provided by B. Stroustrup at https://www.stroustrup.com/PPP2slides.html

- Class textbook