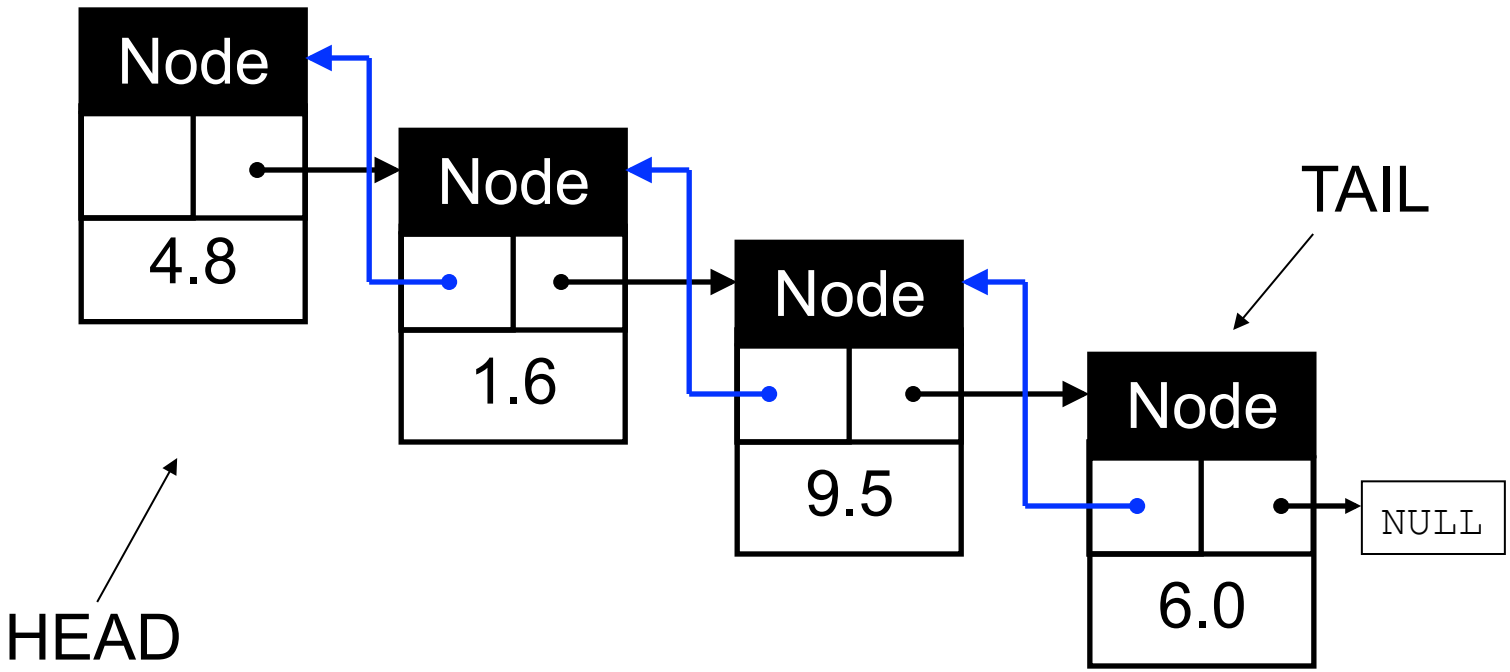
TEMPLATES AND VECTORS

CMPS 148

Linked Lists



Key Takeaways

- We can allocate "just enough" space
  - However, there is some overhead…
- We can insert at front or back quickly
  - **<u>Given a node</u>**, we can insert before or after it quickly too…
- Random access suffers though… why?

Now what?

- This seems like a useful class… but it's a lot of work
  - It would be helpful if we didn't have to rewrite the entire thing to hold lists of doubles, characters, strings, circles…..
  - Templates.

Templates

- We can make templates for functions and classes
- Templates use a place-holder as the data type until runtime, when the actual data type is injected into the code.

Template Functions – Max Value Example

```
int maxValue(int v1, int v2) {
  if (v1 > v2 ) return v1;
  else return v2;
}
```

```
float maxValue(float v1, float v2) {
  if (v1 > v2 ) return v1;
  else return v2;
}
```

```
char maxValue(char v1, char v2) {
  if (v1 > v2 ) return v1;
  else return v2;
}
```

```
template<typename T>
T maxValue(T v1, T v2) {
  if (v1 > v2 ) return v1;
  else return v2;
}
```

Template Classes

- Classes that have member variables can be made with template types as well.

```
class IntContainer {
  public:
     int getValue() {
        return v;
     }
  private:
     int v;
};
```

```
class FloatContainer {
  public:
     float getValue() {
        return v;
     }
  private:
     float v;
};
```

```
template <typename T>
class Container {
  public:
     T getValue() {
        return v;
     }
  private:
     T v;
};
```

Syntax "Issues"

Class and Function template code are best placed in **header** files

This goes against the normal convention of discouraging implementation code in header files

```
int main() {
  Container<int> ic;
  Container<float> fc;
  ic.set(5);
  fc.set(4.5);
  cout << ic.getValue() << endl;
  cout << fc.getValue() << endl;
}
```

```
template <typename T>
class Container {
  public:
     T getValue() {
       return v;
     }
     void setValue(T val) {
       v = val;
     }
  private:
     T v;
};
```

Example Problem

- Lets now adapt our LinkedList class to accommodate **any** data type

Lists

- Just as c-strings can be replaced with the string class, collections like ours can be replaced with the **vector** class

- A **vector** works very similar to our dynamic array class (not sorted though)
  - Grows automatically
  - Can accommodate **any** data type by utilizing templates

The `vector` class

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<int> v1;
  vector<int> v2(3);
  v1.push_back(10);
  v1.push_back(11);
  v1.push_back(12);
  for (int i = 0; i < 3; i++) {
    cout << v1.at(i) << " " << v2.at(i) << endl;
  }
}
```

Creates vector with 3 default values

Adds to end

```
10 0
11 0
12 0
```

Equivalent to v1[i]

The `vector` class

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<int> v1;
  v1.push_back(10);
  v1.push_back(11);
  v1.push_back(12);
  cout << v1.size() << endl;
  v1.pop_back();
  cout << v1.size() << endl;
  v1.clear();
  cout << v1.size() << endl;
}
```

Returns number of element in vector

Removes the last element

Removes all elements

3
2
0

Missing Features?

- Notice that there are no functions that allow you to replace/insert/remove <u>*at a specific index*</u>.
- This is because STL containers use an alternative to integers to represent "positions" in the list
  - STL uses a concept called ***iterators***

Iterator?

- Example: int n[5];
  - You represent positions within an array with a simple integer: n[1], n[i]…
  - You can move to the next position by incrementing or decrementing (i++, i--)
  - The first position is always 0 and the last position is always size -1 .
- An iterator is an abstraction of "position"
  - You can move forwards and backwards
  - You can get the first "iterator" or last "iterator"

Vectors and Iterators

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<int> v1;
  vector<int> :: iterator it;
  v1.push_back(10);
  v1.push_back(11);
  v1.push_back(12);
  it = v1.begin();
  v1.insert(it+1, 9);

  it = v1.begin();
  for (it = v1.begin(); it < v1.end(); it++) {
    cout << *it << endl;
  }
}
```

```
10
9
11
12
```

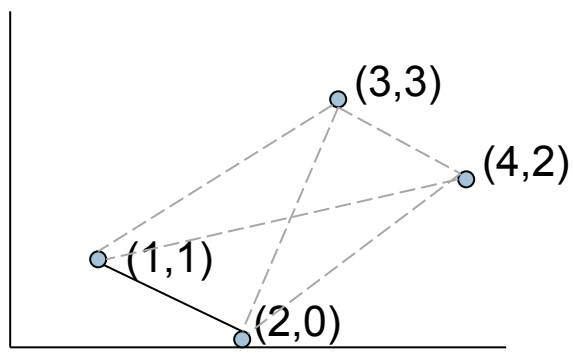
Iterator-based Functions

- vector:: insert(iterator, value)
- vector:: erase(iterator)

- *iterator *dereferences the iterator*
  - Think of an iterator as *pointing* to a value in the vector…

Exercise

- Allow the user to enter a series of 2D points (integer pairs).
- Write your program such that finds the two points that are closest together and prints out their positions and the distance between them.



Standard Template Library

- The vector class is part of a C++ extension called the *Standard Template Library (STL)*.
- The library primarily contains classes that hold *collection* of objects
- All of the collections are templates, and they all have similar functions

```
bitset
deque
list
map
multimap
multiset
priority_queue
queue
set
stack
vector
```

Sequence Containers

- Sequence Containers store **ordered** lists
  - Vector: fast insertion at end, fast random access.
    - Array – based
  - Deque (deck): fast insertion at front **and** end, fast random access.
    - Array-based
    - More overhead than vector
  - List: supports fast insertion anywhere, slow random access
    - Linked-List Based

Associative Containers

- Associative containers are made for **lookup**.
- Set: No duplicates
- MultiSet: Set, with duplicates
- Map: Key/Value pairs – no duplicate keys
- Multi-Map: Maps, with duplicate keys

CMPS 148

REVIEW FOR FINAL

Final Exam

- December 17th: 6:30-9:30pm
- Exam is comprehensive, similar to other exams
- Open book, notes, computer, etc.

Topics

- Functions:
  - Pass By Value vs. Pass By Reference
  - Passing Arrays
- Arrays:
  - Partially filled arrays
  - Inserting / Deleting elements (front, middle, end) - supporting sorted lists
- C-Strings
- Header and cpp Files

Topics

- Classes
  - Member variables and functions
  - Access Protection / Encapsulation
    - Public, private, protected
  - Constructors and Destructors
  - Operator Overrides
  - Using const
    - Parameters, “this”

Topics

- Standard C++ Classes
  - String
  - String Stream
  - Files (ofstream, ifstream)
- Pointers
  - Dynamic Memory Allocation
    - New / delete
    - Dynamic Arrays
    - Pointers and Classes (. vs, ->)

Topics

- Polymorphism
  - Inheritance
  - Polymorphic Pointers & virtual keyword
  - Abstract Classes

Topics

- Linked Lists
  - Know the difference between an ADT and an implementation
  - Know how linked lists work (diagram) and be able to program simple manipulation of nodes
- Templates
  - Template Functions
  - Template Classes
- STL Vectors
  - No iterators