Admin

- Assign 1 due next Wed
  - Web announcements for late-breaking news
- MLK, Jr Day on Monday, no lecture
- Today’s topics
  - CS106 class library: Vector, Grid, Stack, Queue
- Reading
  - Handout 14 (today & next)
- A note about arrays/pointers
  - Covered in Ch. 2, but we wait to introduce until we have a good use for them, so don’t worry for now
- Terman cafe today after lecture

Client use of templates

- Client includes interface file as usual
  - #include "vector.h"
- Client must specialize to fill in the placeholder
  - Cannot use Vector withoutqualification, must be Vector<char>, Vector<locationT>, ...
  - Applies to declarations (variables, parameters, return types) and calling constructor
- Vector is specialized for its element type
  - Attempt to add locationT into Vector<char> will not compile!

Vector class

- Indexed, linear homogenous collection
  - Knows its size
  - Access is bounds-checked
  - Storage automatically handled (grow & shrink)
  - Convenient insert/remove
  - Deep-copy on assignment, pass/return-by-value
- Usage
  - Constructor creates empty vector
  - Add/insert adds new element
  - Access elements using setAt, getAt or operator []
- Useful for:
  - every kind of list you can imagine!

Vector interface

```cpp
template<typename ElemType>
class Vector {
public:
    Vector();
    ~Vector();
    int size();
    bool isEmpty();
    ElemType getAt(int index);
    void setAt(int index, ElemType value);
    void add(ElemType value);
    void insertAt(int pos, ElemType value);
    void removeAt(int pos);
};
```
**Template specialization**

```cpp
class Vector<double> {
public:
  Vector<double>();
  ~Vector<double>();
  int size();
  bool isEmpty();
  double getAt(int index);
  void setAt(int index, double value);
  void add(double value);
  void insertAt(int pos, double value);
  void removeAt(int pos);
};
```

**Client use of Vector**

```cpp
#include "vector.h"
Vector<int> MakeRandomVector(int sz) {
  Vector<int> numbers;
  for (int i = 0; i < sz; i++)
    numbers.add(RandomInteger(1, 100));
  return numbers;
}
void PrintVector(Vector<int> &v) {
  for (int i = 0; i < v.size(); i++)
    cout << v[i] << " ";
}
int main() {
  Vector<int> nums = MakeRandomVector(10);
  PrintVector(nums);
  ...
}
```

**Templates are type-safe!**

```cpp
#include "vector.h"
void TestVector() {
  Vector<int> nums;
  nums.add(7);
  Vector<string> words;
  words.add("apple");
  nums.add("banana"); // COMPILE ERROR!
  char c = words.getAt(0); // COMPILE ERROR!
  Vector<double> s = nums; // COMPILE ERROR!
}
```

**Grid class**

- 2-D homogenous collection indexed by row & col
- Access to elements is bounds-checked
- Deep-copy on assignment, pass/return by value

**Usage**
- Set dimensions in constructor (can later resize)
- Elements have default value for type before explicitly assigned
- Access elements using getAt/setAt or operator ()

**Useful for:**
- Game board
- Images
- Matrices
- Tables
**Grid interface**

```
template <typename ElemType>
  class Grid {
    public:
      Grid();
      Grid(int numRows, int numCols); // overloaded constructor
      ~Grid();
      int numRows();
      int numCols();
      ElemType getAt(int row, int col);
      void setAt(int row, int col, ElemType value);
      void resize(int numRows, int numCols);
  };
```

**Client use of Grid**

```
// Returns a new 3x3 grid of chars, where each elem is initialized to space character
Grid<char> CreateEmptyBoard()
{
  Grid<char> board(3, 3); // create 3x3 board of chars
  for (int row = 0; row < board.numRows(); row++)
    for (int col = 0; col < board.numCols(); col++)
      board(row, col) = ' '; // board.setAt(row, col, ' ')
  return board; // btw, it's ok to return object
}
```

**Stack class**

- Linear collection, last-in-first-out
- Limited-access vector
- Can only add/remove from top of stack
- Deep-copy on assignment, pass/return by value

**Usage**

- Constructor creates empty stack
- push to add objects, pop to remove

**Useful for:**

- Reversing a sequence
- Managing a series of undoable actions
- Tracking history when web browsing

**Stack interface**

```
template <typename ElemType>
  class Stack {
    public:
      Stack();
      ~Stack();
      int size();
      bool isEmpty();
      void push(ElemType element);
      ElemType pop();
      ElemType peek();
  };
```
### Client use of Stack

```cpp
void ReverseResponse()
{
    cout << "What say you? ";
    string response = GetLine();
    Stack<char> stack;
    for (int i = 0; i < response.length(); i++)
        stack.push(response[i]);
    cout << "That backwards is :";
    while (!stack.isEmpty())
        cout << stack.pop();
}
```

### Queue class

- **Linear collection, first-in-first-out**
- **Limited-access vector**
- Can only add to back, remove from front
- Deep-copy on assignment, pass/return by value

#### Usage
- Constructor creates empty queue
- enqueue to add objects, dequeue to remove

#### Useful for:
- Modeling a waiting line
- Storing user keystrokes
- Ordering jobs for a printer
- Implementing breadth-first search

---

### Queue interface

```cpp
template<typename ElemType>
class Queue
{
    public:
    Queue();
    ~Queue();
    int size();
    bool isEmpty();
    void enqueue(ElemType element);
    ElemType dequeue();
    ElemType peek();
};
```

### Client use of Queue

```cpp
void ManageQueue()
{
    Queue<string> queue;
    while (true) {
        cout << "? ";
        string response = GetLine();
        if (response == "next") {
            if (queue.isEmpty())
                cout << "No one waiting!" << endl;
            else
                cout << "Handle" << queue.dequeue() << endl;
        } else {
            queue.enqueue(response);
            cout << "Add" << response << endl;
        }
    }
}
```
Nested templates

- Queue can hold stacks or vector of vector, etc
  ```c++
  Vector<Queue<string>> checkoutLines;
  Grid<Stack<string>> game;
  ```
- Need space between `>>` closers
  - Otherwise compiler see stream extraction
- Can use typedef to make shorthand name
  ```c++
  typedef Vector<Vector<int>> calendarT;
  ```