Data Structures

end of C++
Simple lists
Update:

- HW due tomorrow
- Availability today/tomorrow in 12-12:30 or 1:30-2

  tutoring - afternoon

- Friday: some code, if you want a laptop!
- Git: working now!
  - Remove the csc2100 course (old) one
    - `rm -R csc2100`
  - Make sure your repo is not inside course one
  - Put yours in the cs2100
    I had you create in lab1, so as to avoid name conflicts w/future CS courses

- Next HW - up on Friday, due in a week
The stack ADT:

- push(e) : adds e to top
- pop() : removes top

Also:

- size()
- empty()
- top()
Example:

```cpp
int main() {  
    stack<int> mystack;
    for (int i = 10; i < 20; i+=2)  
        mystack.push(i);
    mystack.pop();
    mystack.push(100);
    cout << mystack.top() << endl;
}
```

See cplusplus.com for lab tomorrow on stacks.

This week, we'll code our own!

At yesterday's lab, due Friday via git.
Meanwhile:

A few more C++ odds and ends

Enum:

```cpp
enum Color { RED, BLUE, GREEN; all caps

Color sky = BLUE;
Color grass = GREEN;
if (sky == BLUE)
    cout << "It's a nice day!";
```

Reason: more readable code
structs : useful for simple collections of data

eenum MealPref {NORMAL, VEG, KOSHER};

struct Passenger {
    string name;
    MealPref foodpref;
    bool isFrequentFlyer;
    int freqFlyerNum;
}

int main() {
    Passenger pass;
    pass.name = "Erin Chambers";
    Passenger pass2 = {"John Smith", VEG, true, 12345};
    
}
Templates

If we want a function to work for multiple data types, like ints and floats, use templates.

Ex:

```cpp
#include <type_traits>

namespace IO {  //Namespaces...}

using namespace std;

namespace IO {

constexpr int min(int a, int b) { return a < b ? a : b; }

int main() {  //Then, in main...
    cout << min(5, 6) << endl;
    cout << min("name", "other") << endl;
    cout << min(pass1, pass2) << endl;
}

} // End of IO namespace
```
Templates in classes

These are important in data structures.

Why?

Need each data struct. to work for many types of data.

Actually, you'll use these in the stack lab:

```cpp
stack <int> mystack;
stack <string> names;
```
Error Handling

In C++, we handle errors by throwing exceptions. (Exceptions are actually their own classes also.)

Recall: What were the ones in Python?

- Type Error
- Value Error
- Name Error

I'll base mine of C++'s default ones:

```cpp
#include <stdexcept>
```

→ see cplusplus for details
Some examples

In Python:

```python
def sqrt(number):
    if number < 0:
        raise ValueError('number is negative')
```

In C++:

```cpp
double sqrt(double number) {
    if (number < 0)
        throw domain_error("number is negative");
}
```

Example:

```cpp
#include <stdexcept>
float & operator[](int index) {
    if ((index >= size) || (index < 0))
        throw out_of_range("index out of range");
    return a[index];
}
```
Then, in your main, need to handle errors:

```cpp
try {
    // any sequence of commands, possibly nested
} catch (domain_error& e) {
    // what should be done in case of this error
} catch (out_of_range& e) {
    // what should be done in case of this error
} catch (exception& e) {
    // catch other types of errors derived from exception class
} catch (...) {
    // catch any other objects that are thrown
}
```

Ex. (in main)
```
MyFloatVec v1(3);
// code to add data
```
```
try {
    cout << v1[5] << endl;
} catch (out_of_range & e) {
    cout << e.what() << endl;
    // index out of range
}
```
void openFileReadRobust(ifstream& source) {
    source.close(); // disregard any previous usage of the stream
    while (!source.is_open()) {
        string filename;
        cout << "What is the filename? ";
        getline(cin, filename);
        source.open(filename.c_str());
        if (!source.is_open())
            cout << "Sorry. Unable to open file " << filename << endl;
    }
}
Now: A first data structure
Singly linked lists:
A collection of nodes that have a linear ordering

Example:

But in memory:

```
head
LAX -> STL
MSP
null
```

```
head
LAX
101
102
MSP
296
297
```

```
101
356
```
Why this structure?

Note: Not the same as C++'s list class (or Python's, for that matter)

However, this linked structure is useful in a number of data structures.

Why not use an array?

- fixed size

Downside:

- getting i-th element is slow
Implementation: Nodes

Huh?

We'll need a node struct (or class).

Contents:
- data
- pointer to another node

Then, in the class, have:
- size
- head
- (sometimes also tail)

Functions:
- constructor
- housekeeping
- add front
- get front
- size or empty
template<typename Object>

class SLinkedList {

private:

    struct SNode {
        Object data;
        SNode *next;
    };

    int s;
    SNode *head;

public:

    SLinkedList();
    ~SLinkedList();

    bool empty();
    int size();
    void addFront(Object val);
    void removeFront();
    Object getFront();