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 csci2100 course (old) one
    - → rm -R csci2100
  - 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() → returns top w/out removing

see cplusplus.com

Example:

```
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!

(yesterday's lab due Friday via git)

Meanwhile:

A few more C++ odds + ends

Enum:

```
enum Color {RED, BLUE, GREEN};
```

RED = "0", BLUE = "1", GREEN = "2"

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

```
enum 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 & floats, use templates.

Ex:

```
template <typename T>
T min(T a, T b) {
    if (a < b)
        return a;
    else
        return b;
}
```

"variable type"

Then: in main

```
cout << min(5, 6);
cout << min("name", "other");
```

```
min(5, "hi");
```
↳ error: input to fxn doesn't match

```
min(pass1, pass2);
```
↳ error: no < for this struct

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:

```
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?

- TypeError
- ValueError
- NameError

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

```
#include <stdexcept>
```

↳ see cplusplus for details

Some examples

In Python:

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

In C++:

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

Ex: My FloatVec

```
#include <stdexcept>
using ...
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:

```
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"

Reading input example:

```
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;
  }
}
```

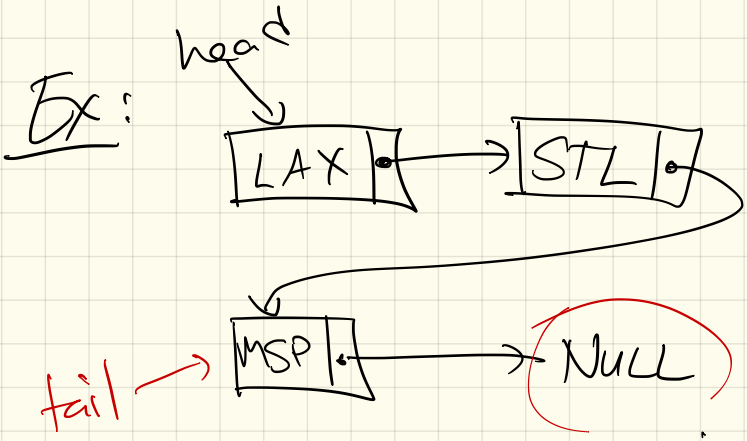
Not covering cin or fstream errors.

Now: A first data structure

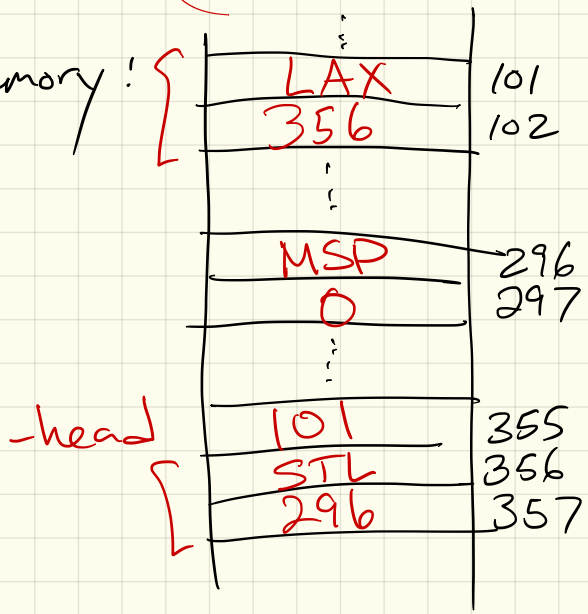
Singly linked lists:

A collection of nodes that have a linear ordering

Ex:



But in memory:



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.

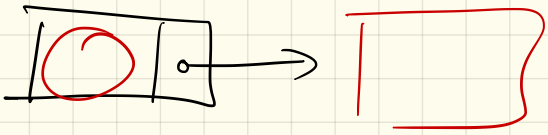
Why not use an array?

- 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
- addFront
- getFront
- size & empty

Code .h file

```
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();
};
```