Data Structures

Housekeeping functions
Stacks (intro)
Last C++ odds and ends
Today:
- HW due Thursday
- Lab tomorrow
- Check repo for grade
Pointers in a class

Pointers are especially useful in classes.

Often, we don't know the details of private variables at time of object creation.

Example: using an array

At time of declaration, need:

- type
- var name
- size
An example: A simple vector class

vector in $\mathbb{R}^2$: $\langle 2, 5 \rangle$

vector in $\mathbb{R}^4$: $\langle 0, 1, 0, 5 \rangle$

So size is not fixed!

How to make a class?

class My Float Vec {
    private:
        int size;
        float * a; // pointer to an array
    public:
        My Float Vec (int s = 10) {
            size = s;
            a = new float [size];
        }

    3. in main
        My Float Vec v(4); v
        size = 4
        a = 1068

    1068
}

// file
Accessing an array:

Pointers to arrays are special because any array in fact is just a pointer to the 1st spot in the array (no * or \rightarrow needed)

Ex: Write a function to allow \[ \] notation, so \[ x[i] \] gives the \( i \)th element in the vector:

```c
public:
    //constructor
    
    float& operator[](int i) {
        if (i < size) return a[i];
        else error = 3;
    }
```
**Garbage Collection:**

In Python, data that is no longer in use are automatically destroyed.

**Ex:**

\[ \begin{align*}
\text{X} & \rightarrow \text{5} \\
\text{5} & \rightarrow \text{Int+} \\
\text{Int+} & \rightarrow \text{Y} \\
\text{Y} & \rightarrow \text{10} \\
\text{10} & \rightarrow \text{Int+} \\
\text{Int+} & \rightarrow \text{X} \\
\text{X} & \rightarrow \text{0} \\
\end{align*} \]

\[ x = 5 \]
\[ x = 10 \]

**Pros:**
- easy
- no user overhead

**Cons:**
- slows language down
C++:

- Value & reference variables are destroyed at the end of their scope

Standard variables are just a label attached to data

=> data is deallocated, so those spaces are now free again.

**Problem: Pointers**

The pointer is destroyed

=> not underlying data

```cpp
int main() {
    int * x = new int (5);
    // memory leak
    // x is deallocated
}
```

Rule: If you use `new`, you must explicitly destroy that data.
So: Housekeeping functions

Basically, need to deal w/these pointer issues.

(1) Copy Constructor

Say I call:

```cpp
MyFloatVec c;
//add data to c
MyFloatVec b(c);
```

Default result?

Copy constructor:

- Takes each private variable
- Sets them

b.size = c.size;

b.a = c.a;
So - overriding this:

```cpp
class MyFloatVec &
{
  // other things...

  public:
    MyFloatVec(const MyFloatVec &other) {
      size = other.size;
      a = new float[size];
      for (int i = 0; i < size; i++)
        a[i] = other.a[i];
    }

  // local "other" is gone
}
```

→ result: deep copies
The `=` operator

Same issue:

```cpp
MyFloatVec x, b;
//put data in

x = b;
```

by default, does shallow copy.

If private data includes a pointer:

```
y = x; x = x.j
```

don't do this
So:

in the class

myFloatVec& operator=(const myFloatVec& other)

if (this! = & other) {
    delete [] a;
    size = other.size;
    a = new float[size];
    for (int i = 0; i < size; i++)
        a[i] = other.a[i];
}

return *this;

(this is like self in python)
Finally: when you create an object

```c
int main() {
    myFloat Vec x;
}
```

3 `x` is destroyed = what happens

Need to deallocate whatever free space is going to use

Memory

```
X
size = 962
```

```c
10.0
3.2
962
```
So:

in class:

```cpp
~MyFloatVec() { delete[] a; }
```

Housekeeping fans:

- Any class in a new
- "opposite" of new:
- tells compiler to follow a pointer
- deallocate what
- variable points to
- tells compiler to follow a pointer
- deallocate what
- variable points to
- tells compiler to follow a pointer
- deallocate what
- variable points to
A note on our first data structure

Stacks: a way to store a list

Ex: Previously visited web pages

Ex: Previous changes to a word document

undo

pop

push

LIFO

Stack
The stack ADT:

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

Also:

- `size()`
- `empty()`
- `top()`

If returns top w/out removing

see cplusplus.com
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;
}
```

Seecplusplus.com for lab tomorrow on stacks.
This week, we'll code our own!
Meanwhile:
A few more C++ odds + ends

Enum:
enum Color { RED, BLUE, GREEN };

Color sky = BLUE;
Color grass = GREEN;

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

Reason:
Struts: 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, 125453};
    
    
}
Templates

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

Ex:

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

Then:
Templates in classes

These are important in data structures.

Why?

Actually, you'll use these in the stock lab.
Error Handling

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

Recall: What were the ones in Python?

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

```cpp
#include <stdexcept>

// seecplusplus 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");
}
```

In general, to avoid crashing:

```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
}
```
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;
    }
}