CS2100: C++ overview

Variable models
+ classes
Housekeeping
C++ odds + ends

Recap

- HW due Friday
- Don't forget free tutoring!
- Lab tomorrow
  speed calculations

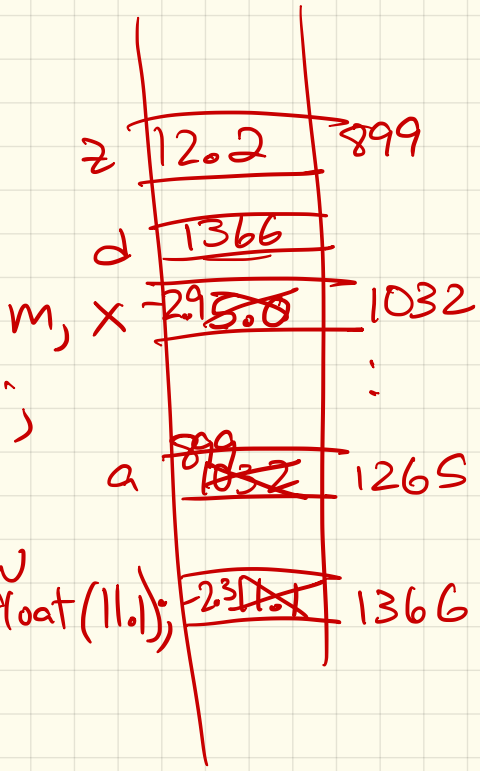
Last time:

Pointer, Reference & Value Variables

(see P1 of HW)

Mess of syntax / models:

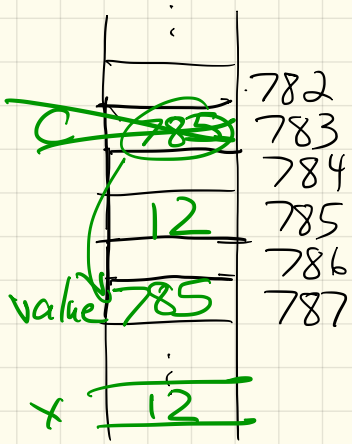
```
float x = 5.0;
float * a;
a = &x;
float & m(x);
float z = 12.2;
a = &z;
float * d = new float(11.1);
(*d) = -2.3;
x = -2.9;
```



The new command

in some function; myfunc

```
int *c;
c = new int(12);
return c;
```



Why: The data persists even after the pointer is gone!

in funct 2:

```
int x = 12;
return x;
```

x → 12

Main use:

in main:

```
int* value = myfun();
```

Passing pointers

Can be useful, since allows NULL option.

Ex:
```
bool isOrigin(Point *pt = NULL){
    return pt->getX()==0 &&
           pt->getY()==0;
}
```

Similar to pass by reference, but can also pass a NULL this way.

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:

- name
- type
- size

But- what if size might change, or is unknown?

Ex: Vector (coordinates in Euclidean space) $\mathbb{R}^n$

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 MyFloatVec {
private:
    int size;
    float * a; //pointer to an array

public:
    MyFloatVec (int s=10) {
        size = s;
        a = new float [size];
    }
    ; write more fcns
```

Accessing an array:

Pointers to arrays are special

↳ any array in fact *is* just a pointer to the 1st spot in the array

(no * or → needed)

Ex: Write a function to allow [] notation, so x[i] gives i^{th} element in the vector:

```
float operator[](int i) {
    return a[i];
}
```

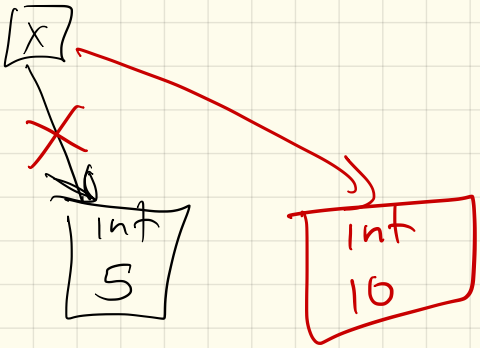
Another: Write a function to scale vector by scalar:

```
void scale(float value) {
    for (int i=0; i<size; i++)
    {
        a[i] = a[i] * value;
    }
}
```

Garbage Collection:

In python, data that is longer in use are automatically destroyed.

Ex:



x=5

x=10

Pros: Easy

Cons: Slow / space

C++:

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

Standard variables are just a label attached to data

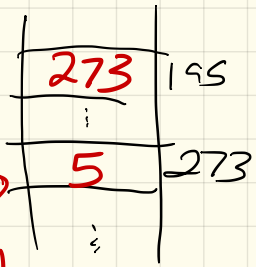
↳ data is deallocated, so those spaces are now free again.

Problem: Pointers

The pointer is destroyed

↳ not underlying data

```
int main() {
  int * x = new int(5);
}
```



// x is destroyed

memory leak! is not

Rule: Must deallocate data yourself - delete

So: Housekeeping functions

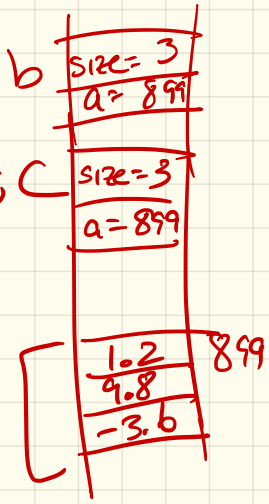
Basically, need to deal w/ these pointer issues.

① Copy Constructor

Say I call:

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

Default result?



↳ copies private var:

b's size = c's size

b's a = c's a

Shallow copy

So - overriding this:

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

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

② The = operator

Same issue:

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

x = b;
```

write operator= to fix this (deep copy)

So:

in the class

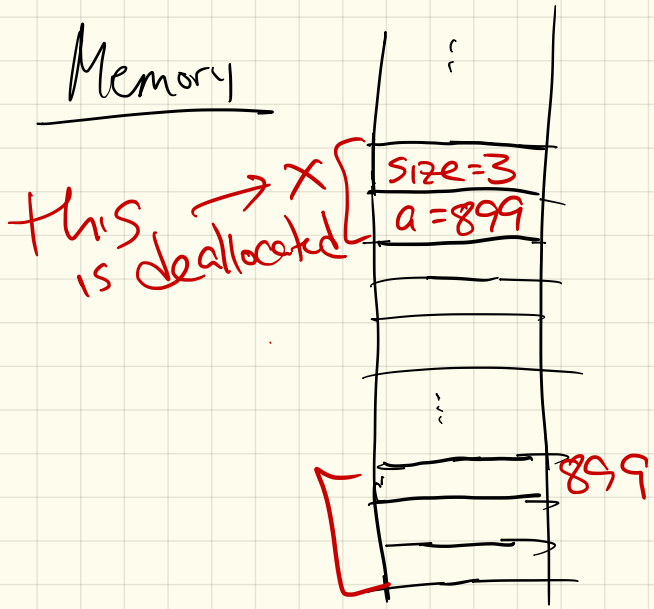
```
myFloatVec& operator=(const myFloatVec& other)
{
    if (this != other) {

    }
    return *this;
}
```

(3) The destructor

Finally: when you create an object

```
int main() {
    myFloatVec x(3);
    :
```

} // x is destroyed ← what happens?

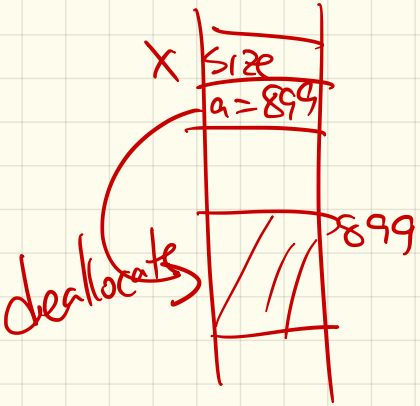


So:

in class:

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

opposite of new



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:

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;
}
```

Then:

Templates in classes

These are important in data structures.

Why?

Actually, you'll use these in the stack 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:

```
#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");
```

In general, to avoid crashing:

```
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
}
```

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;
  }
}
```