Announcements

HW is due today

HW 2 will be posted today due in 1 week

CS2105 - More on C++
Type a size:

Example:

```c
    arr[] = {'a', 'b', 'c'};
```

What do we need when we use an array?

Often, we don't know all the details of the private variables to put in a class, especially useful in classes.

Pointers in a class
```java
public class MyFloatVec {
  private MyFloatVec (int s = 10) : slice(s) { }  
  A = new float [slice]
}
```

class MyFloatVec {  
   public:  
    float *slice = 0;  
    int slice {  
      private:  
    }  
}
```
Declaring the vector: `typedef char Vector[10];`

Accessing the array: `Accessing the array:

Accessing the array:

```
#define MAX_LEN 10
int main()
{
    char Vector[MAX_LEN];
    Vector[0] = 'A';
    Vector[1] = 'B';
    Vector[2] = 'C';
    Vector[3] = 'D';
    Vector[4] = 'E';
    Vector[5] = 'F';
    Vector[6] = 'G';
    Vector[7] = 'H';
    Vector[8] = 'I';
    Vector[9] = 'J';
    return 0;
}
```
```c
for (int i = 0; i <= 32; i++)
    read scale (F卅t value)

Function to Scale by int (in class);
```
Pros: easy, no memory leaks
Cons: slow
In Python, variables that are no longer
destroyed in use are automatically
Garbage Collection
This holds for any type of variable.

End of its scope. Automatically destroyed at the
Basically, any standard variable is
In C++, things are sometimes handled

C++
Rule: If you have a delete, you must have
3 // a is destroyed
delete a;

Need:
Put A in a

int x = a = new int(5);

int main()
{

"new" spot is not.
"A" spot is next created with
while the pointer variable is deleted.

Problem: Points
Large Projects

In C++, we often separate a class into multiple files.
- Easier version control.
- Allows division of files.
- Easy reference for later use.
Example:

Don't actually define a class or function.

In a function header, this is used to declare the
Functions.
```cpp
#include <iostream>

class Point {  //Freestanding operator definitions, outside the formal Point class definition
public:
    double x, y;  // Inline function body
    double operator+(const Point other) const;
    double operator-(const Point other) const;
    double operator*(const double scalar) const;
    double operator/(const double scalar) const;
    double distance(const Point other) const;
    void scale(const double scalar) {
        x *= scalar;
        y *= scalar;
    }  // Inline function body
    void normalize() {
        double length = sqrt(x*x + y*y);
        x /= length;
        y /= length;
    }  // Inline function body
private:
    class Point {  // Need ostream definition for operator<< signature
        std::ostream &operator<<(std::ostream &out, const Point p);
        Point operator+(double scalar) const;
    }  // Inline function body
};

Point operator+(const Point &p, const Point &q) {  // Freestanding operator definitions, outside the formal Point class definition
    return Point(p.x + q.x, p.y + q.y);  // Inline function body
}

Point operator-(const Point &p, const Point &q) {  // Freestanding operator definitions, outside the formal Point class definition
    return Point(p.x - q.x, p.y - q.y);  // Inline function body
}

double operator*(const Point &p, const double scalar) {  // Freestanding operator definitions, outside the formal Point class definition
    return Point(p.x * scalar, p.y * scalar);  // Inline function body
}

double operator/(const Point &p, const double scalar) {  // Freestanding operator definitions, outside the formal Point class definition
    return Point(p.x / scalar, p.y / scalar);  // Inline function body
}

double distance(const Point &p, const Point &q) {  // Freestanding operator definitions, outside the formal Point class definition
    return sqrt((p.x - q.x) * (p.x - q.x) + (p.y - q.y) * (p.y - q.y));  // Inline function body
}

void scale(Point &p, const double scalar) {  // Freestanding operator definitions, outside the formal Point class definition
    p.x *= scalar;
    p.y *= scalar;
}

void normalize(Point &p) {  // Freestanding operator definitions, outside the formal Point class definition
    double length = sqrt(p.x * p.x + p.y * p.y);
    p.x /= length;
    p.y /= length;
}

int main() {  // Freestanding operator definitions, outside the formal Point class definition
    Point p1(1, 2), p2(3, 4);
    Point p3 = p1 + p2;  // Freestanding operator definitions, outside the formal Point class definition
    Point p4 = p1 - p2;  // Freestanding operator definitions, outside the formal Point class definition
    Point p5 = p1 * 2;  // Freestanding operator definitions, outside the formal Point class definition
    Point p6 = p1 / 2;  // Freestanding operator definitions, outside the formal Point class definition
    double dist = distance(p1, p2);  // Freestanding operator definitions, outside the formal Point class definition
    scale(p3, 2);  // Freestanding operator definitions, outside the formal Point class definition
    normalize(p3);  // Freestanding operator definitions, outside the formal Point class definition
    return 0;
}
```

by testPoint.cpp

The usual function (a container)
- One to test problem (A container)
    (might see this)
    (Point.cpp)
    One to declare functions. Class

We then have a kinds of cpp files.
```cpp
double normalization()
{
    if (mag > 0)
    {
        scale(1/mag);
    }

    // measure distance to the origin
    double mag = distance(point);
}

void normalize()
{
    normalization();
}

double point::distance(point other)
{
    double dx = x - other.x;
    double dy = y - other.y;
    double sqrt(dx * dx + dy * dy);
    return sqrt;
}

void point::scale(double factor)
{
    x *= factor;
    y *= factor;
}

Point::Point(double x, double y)
{
    this->x = x;
    this->y = y;
}
```

To Class Scope

using namespace std; // allows us to avoid qualified std: :iostream
#include <cmath>
#include <iostream>

Point::Point(double x, double y) : x(initialX), y(initialY)
```
3

\[
\begin{align*}
\text{Point } (x, y) & \in \text{ menu } (C) \\
\text{using menu specifier std.}
\end{align*}
\]
Got Point

So:

Need to compile in correct order.

Complications: main can't run without compiling.
I'll post a template of how these work...

It can just type "make" I suggest.

If you use the names I suggested.

Generally provide this.

Machines are used to automate this.

Alternatively:
What exceptions were raised in Python?

There are really just classes themselves.

Throwing exceptions by hand in C++ is done even by Experienced

Even Handlings
# More on C++plus.com

```cpp
#include <stdio.h>

int main()
{
    // Included exception classes.
    // Most of mine will be based on C++3.

    // The book uses its own error classes.
    // C++ Exceptions
```
throw DomainError("number is negative");

if (number > 0) {
    double sqrt(double number)
}

raise ValueError("number is negative");

if number > 0:
    def sqrt(number):

Python:
3 return A[index];
// code to pad deck
MyIlootVec &T(E)

To use:

3
3
catch (out-of-range e) 
Cout << e.what() << endl;

if (count <= T.size())

println "Index out of range."
```java
{ catch any other objects that are thrown // }
{ catch other types of errors derived from exception class } (exception e) { try { catch (out-of-range e) { try { catch (domain-error e) { any sequence of commands, possibly nested // } } } } } Catching exceptions
```
Can get a bit long...

Use:
  head(), fail(), etc.
  to detect

Instead, if sets fail:

When something goes wrong. By default, can doesn't raise errors

Other errors
```c
{
    
    cout << "Your number must be from 1 to 10." << endl;
    } (number > 1) || number < 10) {
    
    number = 7
    }
    
    cout << "We will choose for you." << endl;
    }
    else if (cin.fail()) {
    
    number = 7
    }
    
    cout << "We will choose for you." << endl;
    } (cin.eof()) {
    
    number = 7
    } else if (cin.fail()) {
    
    cout << "That is not a valid integer." << endl;
    } else {
    
    cin.clear();
    }
    
    cout << "Enter a number from 1 to 10." << endl;
    }
    while (number > 1 || number < 10) {
    
    number = 0;
    }
```
```cpp
{
    cout << "Sorry. Unable to open file " << filename << endl;
    if (!source.is_open())
        source.open(filename.c_str());
    else
        cout << "Which is the filename?";
    string filename;
    while (source.is_open())
    {
        source.close();
    }
    void openIfReadable(istream& source)
    {
        Simlar to cin.
    }
    The streams are enps
```
Sorcery (someone?)

So what do we need (someone?)

- In My Fucking class, we need to
  - More on arrays as private variables.

Problem: first array was incorrect with a new value in an array.

Delete? (someone?)

\[ p[3] = p[2] + \vec{v} = \begin{cases} 4 \cdot \vec{e}_3 \\
2 \cdot \vec{e}_2 \\
\end{cases} \]

My feet vec p[1] (3).

In Main:

Example Main:
3

delete[]

My Function is

in Class

Destructor:
Copy Constructor

Shallow Copy (Best)

by default, sets private variables =

\[
\begin{align*}
\frac{s_1}{a} &= \frac{10}{10} \\
\frac{s_2}{a} &= \frac{10}{5}
\end{align*}
\]

Consider the MyFloatVec class.

What if we have \( a = b \) or MyFloatVec \( b(a) \)?
\[ a \in [0, 1] \quad \text{and} \quad c \in [1, \infty) \]

For \( c \geq 0 \), \( c \leq \frac{1}{2} \) \( \Rightarrow \)

\[ a = \text{new float (outside of \([0, 1]\))} \]

\[ \text{size} = \text{other size} \]

\[ \text{MyFloatVec} \text{ (const MyFloatVec & other)} \]

\( \text{clone a copy constructor function} \)

To avoid shallow copies, we need to
Another issue:

```
MyFloat Vec::
C = a;
```

What does this do?

\[
\frac{a}{10}\frac{512}{20}
\]

\[
\text{size}
\]

10
3. return this;

1t (this "= 80flor")

If Footloc Operator = (const MyFootloc& other)

Solution: rewrite the "=" operator.
Récit: Transcending Functions