Thursday, August 31, 2006
11:07 AM

Point class contd.

Thursday, August 31, 2006
11:07 AM

What do we want point class to handle:

- easily take the average — math
- sort points
- remove duplicates

} access to point data

done by STL

Thursday, August 31, 2006
11:07 AM

sorting:

```
#include <algorithm>
vector<Point* >   pts;
sort(pts.begin(), pts.end(), pointAxisCompare0());
```

what is this?

semantics:

```
if (p1[0] < p2[0])
    return true;
```

a fancy function that compares two points

if

true & false are C++ keywords

Thursday, August 31, 2006
11:07 AM

Our "fancy" C++ function is actually a class with just one member method, namely operator()

function object or functor.

```
class PointAxisCompare ∅
{
    public:
    bool operator()(Point p1, Point p2) {
            return (p1[0] < p2[0]);
    }
}
```

need double Point::operator[](int i) {
    return point(i);
}

vector<double> point

Thursday, August 31, 2006
11:07 AM

- in asg1:
  - implement 2D point class so that you can compute average, sort, & removal of duplicates (triop.cpp) ← driver
  - use point.h as starting place
  - you need to implement operator>>, operator<<

```
Point(vector<double> c);
Point(const Point& rhs);
point(Point* rhs);
```

} constructors, operator=

Thursday, August 31, 2006
11:07 AM

```
pts.erase(unique(pts.begin(), pts.end(), pointDuplicate(),
                 pts.end()),
```

another functor
that compares two points —
semantically, it's like operator==

```
class pointDuplicate
{
public:
bool operator()(Point p1, Point p2)
    return(p1.duplicate(p2));
}
```

you need to write this member function

```
{ dx = p1(0) - p2(0)
  dy = p1(1) - p2(1)
  dist = sqrt(dx*dx + dy*dy);
  if(dist < 0.01)
      return true
  else return false
```

Thursday, August 31, 2006
11:07 AM

```
dx = p1[0] - p2[0]
dy = p1[1] - p2[1]
```

p1.duplicate(p2)

this

```
bool Point::duplicate(const Point & rhs)
{
    double dx = (*this)[0] - rhs[0];
    double dy = (*this)[1] - rhs[1];
    ⋮
```

Thursday, August 31, 2006
11:07 AM

```
#include <iostream>
# include <iomanip>
# include <vector>
# include <cmath>

using namespace std;  // bad habit

int main(---)
{   Point*   ptp;
    vector<Point* >   pts;
    vector<Point* >::iterator   pp;
    while(std::cin >> (ptp = new Point(0.0, 0.0)))
        pts.push_back(ptp);
```

operator>>(istream s, Point *p) {
  s >> *p;
}

operator>>(s, Point p)
{
  s>>p(0); s>>p(1)

Dr. D's

constructor Point(double, double)
allocates mem
just like malloc

Point *

operator>>

Thursday, August 31, 2006
11:07 AM

Need to calculate average:

Several ways to do this:

1. use iterators:

```
vector<Point*> pts;

Point mean(0.0, 0.0);

for (pp = pts.begin(); pp != pts.end(); ++pp)
    mean += *pp;
mean /= (float) pts.size();
```

```
Point.operator+= (Point *rhs)
{
    (*this)[0] += (*rhs)[0];
    (*this)[1] += (*rhs)[1];
}
```

or

```
for (int i=0; i < point.size(); ++i)
    (*this)[i] += (*rhs)[i];
```

Basic approach,

lets say given array of floats,

```
float val[10];
float sum = 0.0;
for (int i=0; i<10; i++)
    sum += val[i];
sum = sum/(float)10;
```

Thursday, August 31, 2006
11:07 AM

2. use random access on vector<Point* > pts;

```
for (int k=0, k< (int) pts.size(); ++k)
        mean += pts[k],
mean /= (float) pts.size();
```

```
Point::operator/=(float f)
{
    (*this)[0] /= f;
    (*this)[1] /= f;
}
```

Thursday, August 31, 2006
11:07 AM

Calculating average "on the fly" (no final division)

$$m_0 = pt_0$$
$$m_1 = \frac{1}{2} m_{\cancel{0}1} + \frac{1}{2} pt_1$$
$$m_2 = \frac{2}{3} m_{\cancel{1}2} + \frac{1}{3} pt_2$$
$$m_3 = \frac{3}{4} m_{\cancel{2}3} + \frac{1}{4} pt_3$$
$$\vdots$$
$$m_k = \frac{k}{k+1} m_{\cancel{k-1}} + \frac{1}{k+1} pt_k$$

red ink = programming instruction

k → oop! what does "previous" mean here?

$$m_k = \frac{k}{k+1} m_k + \frac{1}{k+1} pt_k$$

does it work for base case?

$$m_0 = \frac{0}{0+1} m_0 + \frac{1}{1} pt_0 = pt_0$$

Thursday, August 31, 2006
11:07 AM

- Sort of asg1 — most of the C++ stuff is in text:
  - Chp. 1.4–1.5, 1.6.4 (C++ classes, C++ details)
  - chp 3.3 (vector & list in the STL)

Chp. 1.4:
- C++ class
- ✓ members
- ✓ member functions
- ✓ constructor
- accessor
- mutator — operator()
- interface + implementation → Makefiles
- scoping op ::
- vector & string (STL)

Chp. 1.5:
- pointers
- $\boxed{\text{parameter passing}}$
- the hy: 3

Chp. 1.6.4 functors

Chp. 3.3 vector:
- size(), push_back()

Thursday, August 31, 2006
11:07 AM

separation of interface + implementation

point.h point.cpp

Makefile

step1: compile (.cpp → .o)

step2: link (assembly of all .o files)

Makefile.thisp:

step1:

.cpp.o:

g++ -c $*.c -o $*.o

step2:

g++ -o $@ thisp.o $(OBJS) $(LDLIBS)