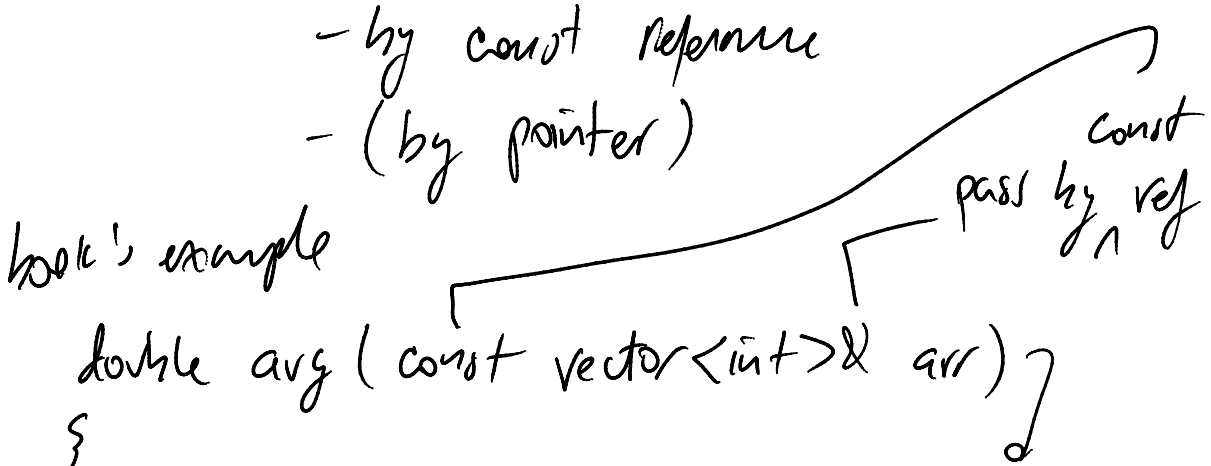
Thursday, August 27, 2009
3:46 PM

Today:

- parameter passing
  - by value
  - by reference
  - by const reference
  - (by pointer)

book's example




```
double avg(const vector<int>& arr)
{
    double sum=0.0;
    double av=0.0;

    for(int i=0; i<arr.size(); i++)
        sum += arr[i];

    av = sum/(double)arr.size();

    return av;
}
```

(int)

Thursday, August 27, 2009
3:51 PM

pass by value

```
double avgv(vector<int> arr)
{
    :
}
```

this works, but...

performance hit in copy operation

how big is myarr?
Unknown. size 0 at present

```
main()
{
    vector<int> myarr;
    for(int i=0; i<10000; i++)
        myarr.push_back(i);
    std::cout << avg(myarr);
    std::cout << avgv(myarr);
}
```

3rd way: pass-by-pointer (old style)

```
double avgp(vector<int> *arr)
```

```
{
    for(int i=0;i < (*arr).size(); i++)
        sum += (*arr)[i]
    :
}
```

call: `avg(&myarr);`

Thursday, August 27, 2009
4:02 PM

Incidentally,

```
double avg(const vector<int>& arr)
{
    double av = 0.0;
    for(int i=0; i<arr.size(); i++)
        av = (float)i/(float)(i+1) * arr[i] +
             1.0/(float)(i+1) * arr[i];
    return av;
}
```

"running mean" instead of familiar two-step approach (sum, sum/n)

$$m_i = \frac{i}{i+1} m_i + \frac{1}{i+1} a_i \qquad \equiv \frac{1}{n}\sum a_i$$

$$m_0 = \frac{0}{1} m_0 + \frac{1}{1} a_0 = a_0$$

$$m_1 = \frac{1}{2} m_1 + \frac{1}{2} a_1 = \frac{1}{2}(a_0) + \frac{1}{2} a_1$$

$$m_2 = \frac{2}{3}\left(\frac{1}{2}a_0 + \frac{1}{2}a_1\right) + \frac{1}{3} a_2$$

$$= \frac{1}{3} a_0 + \frac{1}{3} a_1 + \frac{1}{3} a_2$$

$$\vdots$$

$$m_n = \frac{n}{n+1}\left(\frac{1}{n}\sum_i^{n-1} a_i\right) + \frac{1}{n+1} a_n$$

$$m_n = \frac{n}{n+1}\left(\frac{1}{n}\sum_i a_i\right) + \frac{1}{n+1}a_n$$

$$\frac{1}{n+1}\sum_{(n-1)} a_i + \frac{1}{n+1}a_n$$

$$\frac{1}{n+1}\left(\sum_{(n)} a_i\right) \equiv \frac{1}{n}\sum a_i \quad \checkmark$$

QED

Thursday, August 27, 2009
4:12 PM

what's this < > notation?

indicates template

templating a class is like giving it flexibility to handle different types (generic class)
int, float, double

specializations

- our Array class currently only handles (provides) an int array
- I want the class to provide int's & float's
- just like vector's

vector of int's

usage: main()
{
vector<int> iv;
vector<double> dv;
}

- to make a class templated. doubles.

① put in forward declaration (.h)

② add in typename T "syntactic sugar" to generalize the class

③ add in specialization (.cpp) bottom of