more on operator overloading

Monday, October 18, 2010
9:00 AM

- using operator+= to define operator+

$\downarrow$

```
vec_t v4,v5;
double d=6.0;
```

v4 += v5

$$\underset{v4}{\begin{pmatrix}1\\2\\3\end{pmatrix}} + \underset{v5}{\begin{pmatrix}6\\7\\8\end{pmatrix}} \rightarrow \underset{v4}{\begin{pmatrix}7\\9\\11\end{pmatrix}}$$

v4 += d;

$$\underset{v4}{\begin{pmatrix}1\\2\\3\end{pmatrix}} + \begin{pmatrix}d\\d\\d\end{pmatrix}$$

same for
v4 *= d;
$\uparrow$
same as old vec_scale(a,b,c)

```
// compound assignment operator +=, from:
// http://www.cs.caltech.edu/courses/cs11/material/cpp/donnie/cpp-ops.html

Vec_t& operator+=(const vec_t& rhs)
{
  ~~if (this != &rhs)~~
      for(int i=0;i<3; i++) vec[i] += rhs[i]
  return *this;
}
```

not sure if this is needed - prob. not

will this work for a += a ? should work...

? (a+b) = c

allows why? I don't know

const vec_t operator+(--- rhs)

allows ? why ! I don't know

```
const vec_t operator+(--- rhs)
{
    vec_t result(*this);  ≡ vec_t result = *this,
    result += rhs;
    return(result)
}
```

can reduce further

```
const vec_t operator+(const vec_t& rhs)
{
    return vec_t(*this) += rhs;
}
```

Monday, October 18, 2010
9:21 AM

material_t

```
class material_t
{
        :
    private:
    int cokie;
    std::string name;
    drgb_t ambient;
      "     diffuse
      "     specular;
};
```

- up at ~~the~~ top:

```
    // constructor
    material_t () : \
cookie(MAT_COOKIE), \
ambient(0.0, 0.0, 0.0), \
        ⋮
{ };
```

// copy constructor

// destructor (default ok)

// assignment operator

// friends
-- istream --
-- ostream ---

class
↑ interface

Monday, October 18, 2010
9:24 AM

```
{ #include <iostream>        "pixel.h"
{ #include <string>          "material.h"
  using namespace std;
```

material_t class implementation!

```
~~std~~::ostream& operator<<(~~std~~::ostream& s, const material_t & rhs)
                              ~~std~~::string
{
    s << "material␣" << rhs.name.c_str() << ~~std~~::endl;
                                 gives c-style
                                 char *

    s << "{" << ~~std~~::endl;
  ( s << "{\n"; )
    if (! rhs.ambient.iszero()) s << "␣ambient␣" << rhs.ambient << std::endl;
          "
        same for diffuse, specular
    s << "}" << std::endl << std~~::endl~~;

    return s;
}

std::istream& operator>>(std::istream& s, material_t& rhs)
{
        char c;
        std::string attrname;
    // have read "material" token.

    s >> rhs.name;

    // consume all chars until '{'
    while (s.good() && s.get(c) && (c != '{'));
    // loop until we hit '}'
    while ((c = s.peek()) != '}') {

            // read attr name
            s >> attrname;
        //read in attribute & consume whitespace at EOL
        if( attrname == "ambient") s >> rhs.ambient >> std::ws ;

              " same for diffuse, specular

    } // end of while

    // eat '}' character
    while ( s.good() && s.get(c) && (c!='}'));

    return s;
}
```

Monday, October 18, 2010
9:37 AM

in main.cpp

material name f'n looks for

```
material_t * material_getbyname(list_t& mats, std::string name)
{
        material_t   *mat;

    mats.reset();
    while (mats.not_end()) {
        mat = (material_t *) mats.get_data();
        assert (mat->getcookie() == MAT_COOKIE);
```

have to provide this accessor f'n otherwise compiler says: mat->cookie is private

```
        if (mat->getname() == name) return mat;
        mats.nextlink();
    }
    return NULL;
}
```

in 212, we use

C++ built-in lists ("containers")

& list iterators

```
std::list<material_t *>    mats;

std::list<material_t *>::iterator    it;
while (it= mats.begin(); it < mats.end(); it++)
{
        if ((*it)->getname() == name)
                return (*it)

        return null;
}
```