Subject: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Fri, 10 Jun 2011 14:00:09 GMT
View Forum Message <> Reply to Message

hi guys..

upp has AFAIK no complex datatype. why not add one.?
does not need to be fancy, in fact best to be compatible in alignment for later use with fftw or the like.

up for discussion:

```
struct complex
{
 double re;
 double im;

public:
 static const complex i;
 static const complex j;
 static const complex zero;

 complex() : re(0.), im(0.) {}
 complex(double re) : re(re), im(0.) {}
 complex(double re, double im): re(re), im(im) {}

 complex& operator=(const complex& c) { re = c.re; im = c.im; return *this; }
 complex& operator=(const double& val) { re = val; im = 0.; return *this; }

 complex conj() const { return complex(re, -im); }
 double norm() const { return re * re + im * im; }

 complex& operator++ () { ++re; return *this; }
 complex operator++ (int) { complex temp(*this); ++re; return temp; }
 complex& operator-- () { --re; return *this; }
 complex operator-- (int) { complex temp(*this); --re; return temp; }

 complex operator+(const complex& c) const { return complex(re + c.re, im + c.im); }
 complex operator-(const complex& c) const { return complex(re - c.re, im - c.im); }
 complex operator*(const complex& c) const { return complex(re * c.re - im * c.im, re * c.im + im *
c.re); }
 complex operator/(const complex& c) const { double den = c.re * c.re + c.im * c.im; return
complex((re * c.re + im * c.im) / den, (im * c.re - re * c.im) / den); }
 complex& operator+= (const complex& c) { re += c.re; im += c.im; return *this; }
 complex& operator-= (const complex& c) { re -= c.re; im -= c.im; return *this; }
 complex& operator*= (const complex& c) { const double temp = re; re = re * c.re - im * c.im; im =
im * c.re + temp * c.im; return *this; }
```

```
 complex& operator/= (const complex& c) { const double den = c.re * c.re + c.im * c.im; const
double temp = re; re = (re * c.re + im * c.im) / den; im = (im * c.re - temp * c.im) / den; return *this;
}

 complex operator+ (const double& val) const { return complex(re + val, im); }
 complex operator- (const double& val) const { return complex(re - val, im); }
 complex operator* (const double& val) const { return complex(re * val, im * val); }
 complex operator/ (const double& val) const { return complex(re / val, im / val); }
 complex& operator+= (const double& val) { re += val; return *this; }
 complex& operator-= (const double& val) { re -= val; return *this; }
 complex& operator*= (const double& val) { re *= val; im *= val; return *this; }
 complex& operator/= (const double& val) { re /= val; im /= val; return *this; }

 friend complex operator+ (const double& l, const complex& r) { return complex(l + r.re, r.im); }
 friend complex operator- (const double& l, const complex& r) { return complex(l - r.re, -r.im); }
 friend complex operator* (const double& l, const complex& r) { return complex(l * r.re, l * r.im); }
 friend complex operator/ (const double& l, const complex& r) { const double den = r.re * r.re +
r.im * r.im; return complex(l * r.re / den, -l * r.im / den); }

 bool operator==(const complex &c) const { return re == c.re && im == c.im; }
 bool operator!=(const complex &c) const { return re != c.re || im != c.im; }
 bool operator==(const double& val) const { return re == val && im == 0.; }
 bool operator!=(const double& val) const { return re != val || im != 0.; }

 friend bool operator==(const double& l, const complex& r) { return l == r.re && r.im == 0.; }
 friend bool operator!=(const double& l, const complex& r) { return l != r.re || r.im != 0.; }
};

//.cpp
#include "complex.h"

const complex complex::i(0., 1.);
const complex complex::j(0., 1.);
const complex complex::zero(0., 0.);
```

note that it does not know anything about Value !! to keep it as 'native' as possible.

thus is tricky to make it Value aware from 'outside'. a way would be what i described in
 http://www.ultimatepp.org/forum/index.php?t=msg&goto=325 14&#msg_32514
see my last patch.

this would make possible to use types for Value, that are not intrinsic but not 'editable' (so as to specify AssignTypeNo etc. in derive list).

NAMESPACE_UPP

```
template<> inline bool IsNull(const complex& r) { return r.re < DOUBLE_NULL_LIM || r.im < DOUBLE_NULL_LIM; }
template<> inline void SetNull(complex& x) { x.re = x.im = DOUBLE_NULL; }
inline const complex& Nvl(const complex& a, const complex& b)  { return IsNull(a) ? b : a; }

const dword COMPLEX_V   = 20;
template<> inline dword ValueTypeNo(const complex*)   { return COMPLEX_V; }

VALUE_COMPARE(complex)

template<> inline unsigned GetHashValue(const complex& x) { return CombineHash(GetHashValue(x.re), GetHashValue(x.im)); }
template<> inline String AsString(const complex& x) { return String().Cat() << "C(" << x.re << "," << x.im << ")"; }
template<> inline Stream& operator%(Stream& s, complex& x) { s % x.re % x.im; return s; }

END_UPP_NAMESPACE
```

ideas and critics welcome

background: i'm on way of wrapping fftw or providing some native fft support for Upp..we don't habve anything such here yet.
