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.

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by dolik.rce on Fri, 10 Jun 2011 20:52:45 GMT
View Forum Message <> Reply to Message

Hi!

You are right about U++ missing the basic mathematical tools. As a physicist, I have needed the complex numbers in my programs many times. The same goes for many other mathematical tools.

Complex type doesn't really fit into Core, but what about creating a Math package, that would include such tools? It could also use the Eigen package (btw: I think there is already a complex numbers type in eigen, what about just wrapping that one?) and the fftw package you talked about. And possibly others, as they emerge.

About your proposed implementation: It looks quite good, from the quick look it has most of the basic functions I ever needed... I don't think I would really use a Value compatibility, but if you write it, I won't mind I have only one objection: The "complex::i" is not very useful. It is too long to use in equations I would prefer something like global constant/macro named just "I". In uppercase to avoid problems with "int i", which is on thousands of places in any program I know

Best regards,

Honza

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Sun, 12 Jun 2011 07:30:17 GMT
View Forum Message <> Reply to Message

here comes a first package. complex extended a bit. got some ideas here and there..

it also has an fft implementation, which i havent credited to author yet. but i think this will be refactored anyway. it's just example..

proper complex implementation should go out of Math anyway..
since Nuller needs to have it as 'operator complex()'..

i'd really like to keep complex as netaive as possible.

se for your own. i'll be away for a week

File Attachments
1) Math.rar, downloaded 307 times

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by Tom1 on Sun, 12 Jun 2011 16:04:52 GMT
View Forum Message <> Reply to Message

Hi,

As you are referring to FFT etc.. Please check out Ooura FFT packages at http://www.kurims.kyoto-u.ac.jp/~ooura/fft.html as they have decent license terms (IMO) in contrast to e.g. fftw. I have used those for several years now and including them in Upp would be optimal.

Best regards,

Tom

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by koldo on Mon, 13 Jun 2011 06:47:25 GMT
View Forum Message <> Reply to Message

Hello kohait

There is also a FFT package in Eigen.

See here: http://eigen.tuxfamily.org/index.php?title=EigenFFT

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by Tom1 on Mon, 13 Jun 2011 09:20:56 GMT
View Forum Message <> Reply to Message

Hi Koldo,

Please note that EigenFFT license terms (GPL/LGPL) are not especially favorable from the Upp point of view. Of course the FFTW (one fft backend for EigenFFT) offers a commercially usable license option, but it comes with a price tag on it.

Best regards,

Tom

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by dolik.rce on Mon, 13 Jun 2011 10:15:18 GMT
View Forum Message <> Reply to Message

Tom1 wrote on Mon, 13 June 2011 11:20Hi Koldo,

Please note that EigenFFT license terms (GPL/LGPL) are not especially favorable from the Upp point of view. Of course the FFTW (one fft backend for EigenFFT) offers a commercially usable license option, but it comes with a price tag on it.

Best regards,

Tom

Hi Tom,

Since Eigen consist of headers only, it can be used in comercial projects without worrying about the (L)GPL Also AFAIK at least on of the backends (kissfft) is licensed under the same terms as Eigen to assure compatibility.

Eigen authors say this about LGPL:Quote:When you distribute software that uses [unmodified] Eigen, the LGPL requires you to:
Say somewhere that that software uses Eigen, and that Eigen is LGPL-licensed.
Give a link to the text of the LGPL license.

See Section 3 of the LGPL.
That's pretty much the same as the 2-clause BSD license.
Please read the Licensing FAQ on Eigen website for further details and explanations

Best regards,
Honza

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by Tom1 on Mon, 13 Jun 2011 17:24:57 GMT
View Forum Message <> Reply to Message

Hi all,

Please do not get me wrong, but I'm really and severely allergic to both GPL and LGPL... If I pick up anything with even LGPL license, I feel like digging a hole for myself.

Generally: I like to link statically to keep everything compatible and hassle free on my clients' computers. So even LGPL is not an option.

OK, I looked at http://sourceforge.net/projects/kissfft/ and it says it's BSD licensed, so kissfft is OK in that sense.

Best regards,

Tom

[EDIT] Oh yes, forgot to mention: I understand they have a finely tuned licensing solution with the header structure to make it almost like BSD, but still LGPL without the hassle.... I wonder why not BSD if this is what they want to compare it to?

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by mirek on Mon, 13 Jun 2011 18:12:48 GMT
View Forum Message <> Reply to Message

W.r.t. complex, maybe in this particular, would not it be a good ideas to use std::complex this time?

Mirek

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by dolik.rce on Tue, 14 Jun 2011 05:44:16 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 13 June 2011 20:12W.r.t. complex, maybe in this particular, would not it be a

good ideas to use std::complex this time?

Mirek
Hi Mirek,
Std::complex is not a bad implementation. It is versatile, optimized and also has a good support in gdb On the other hand, I'd like to have sommething more "U++-like". What about wrapper class that would have std::complex as a base and just add functions like ToString, Xmlize etc.?

Honza

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by mirek on Tue, 14 Jun 2011 06:15:20 GMT
View Forum Message <> Reply to Message

dolik.rce wrote on Tue, 14 June 2011 01:44mirek wrote on Mon, 13 June 2011 20:12W.r.t. complex, maybe in this particular, would not it be a good ideas to use std::complex this time?

Mirek
Hi Mirek,
Std::complex is not a bad implementation. It is versatile, optimized and also has a good support in gdb On the other hand, I'd like to have sommething more "U++-like". What about wrapper class that would have std::complex as a base and just add functions like ToString, Xmlize etc.?

Honza

Some of them could be done as external function, just like we did it for int/double...

Mirek

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by dolik.rce on Tue, 14 Jun 2011 06:45:26 GMT
View Forum Message <> Reply to Message

mirek wrote on Tue, 14 June 2011 08:15dolik.rce wrote on Tue, 14 June 2011 01:44mirek wrote on Mon, 13 June 2011 20:12W.r.t. complex, maybe in this particular, would not it be a good ideas to use std::complex this time?

Mirek
Hi Mirek,
Std::complex is not a bad implementation. It is versatile, optimized and also has a good support in gdb On the other hand, I'd like to have sommething more "U++-like". What about wrapper class that would have std::complex as a base and just add functions like ToString, Xmlize etc.?

Honza

Some of them could be done as external function, just like we did it for int/double...

Mirek
I know it is possible in most cases... But the wrapper has more pros that I forgot to mention. E.g. it is a clean way to "move" complex to Upp namespace... I am too lazy person to type "using std::complex;" every time I want to use it (And hardcoding the using clause into U++ doesn't feel right...)

Honza

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Mon, 20 Jun 2011 07:36:36 GMT
View Forum Message <> Reply to Message

Quote:
I know it is possible in most cases... But the wrapper has more pros that I forgot to mention. E.g. it is a clean way to "move" complex to Upp namespace... I am too lazy person to type "using std::complex;" every time I want to use it (And hardcoding the using clause into U++ doesn't feel right...)

this is again a base principles decision. why not thinking of complex as 'just another logical type like int or double'. this would relief the namespace boundaries..

i'm not quite familiar with the std::complex implementation, which states to be optimized (through template specialization) for double, int and float..

but as mirek said, i dont mind to use std::complex at all.

again, to make std::complex usable in Value, we'd need to have those (or some related) changes in SetNull behaviour or extend Nuller (which probably is a lot esier.

as of fft: doubtlessly fftw is the fastest implementation and the most versatile. but the api sucks definitely. so it'd be a must to have it as plugin to be able to expose it's c api as C++, thus avoiding LGPL issues. ofcorse static linkage isnt possible then.

EDIT: in case std::complex<>: to write complex<double> each time is cumbersome. a \typedef complex<double> cdouble' and respectively for the others would be good..
typedef

EDIT: no need to derive from complex to pull it into upp namespace. just place the 'typedef std::complex<double> cdouble' in the upp namespace..cdouble is then Upp::cdouble. just tested it.

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Mon, 20 Jun 2011 11:44:27 GMT

as of EigenFFT:
Quote:
The work is currently in the unsupported section.
Help is welcome.

seems as if they are still in progress with it.

as of FFTW, which is GPL, there will be a problem adding it as plugin isn't it? would a c++ wrapper be considered derivative work? (the old hassle question..)

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Mon, 20 Jun 2011 12:24:06 GMT

as a starting point:

```
#include <complex>

NAMESPACE_UPP

typedef std::complex<double> cdouble;

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

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

//VALUE_COMPARE(cdouble), doesnt work since Value has no native cdouble conversion support, TODO
inline bool operator==(const Value& v, cdouble x)   { return RichValue<cdouble>::Extract(v) == x; }
inline bool operator==(cdouble x, const Value& v)   { return RichValue<cdouble>::Extract(v) == x; }
inline bool operator!=(const Value& v, cdouble x)   { return RichValue<cdouble>::Extract(v) != x; }
inline bool operator!=(cdouble x, const Value& v)   { return RichValue<cdouble>::Extract(v) != x; }

template<> inline unsigned GetHashValue(const cdouble& x) { return CombineHash(x.real(), x.imag()); }
template<> inline String AsString(const cdouble& x) { return String().Cat() << "C(" << x.real() << "," << x.imag() << ")"; }
template<> inline Stream& operator%(Stream& s, cdouble& x) {
 double r,i;
 if(s.IsStoring()) { r = x.real(); i = x.imag(); }
```

```
 s % r % i;
 if(s.IsLoading()) { x = cdouble(r,i); }
 return s;
}

END_UPP_NAMESPACE
```

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Tue, 21 Jun 2011 17:00:46 GMT
View Forum Message <> Reply to Message

i'm done with a 'plugin/kissfft', but need the cdouble thing for it.

Core.h

```
#include <string>
+ #include <complex>
```

Defs.h:247

```
+ typedef std::complex<double> cdouble;
```

String.h:783

```
+ template<> inline String AsString(const cdouble& x)        { return String().Cat() << "C(" << x.real() << "," << x.imag() << ")"; }
```

with theese changes it should be possible to use the plugin in attachment, move the kissfft to plugin folder.

for Value interaction i need some help, since the simple = Null assignment wont work due to ctor and operator= ambiguities in std::complex. will try to outline it in next post.

File Attachments

1) kissfft.rar, downloaded 278 times

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Wed, 22 Jun 2011 11:12:16 GMT
View Forum Message <> Reply to Message

here comes a patch file for including complex as an additional basic datatype. thats why i have placed it in Defs.h, though it's 'math' related, but the other data types are math as well, right
it contains the integration of it in Value as far as possible. but due to it beeing a class/struct and having ctor and operator=, it is not possible to use =Null / from-Value-conversion semantics. or at least i havent succeeded to do so. see my ComplexTest.

quick view, maybe anyone has an idea how to make this possible anyway..or leave it as issue.

NOTE: for successfull usage of std::complex (cdouble) in RichValue, the SetNull template usage change is needed. see my thread
http://www.ultimatepp.org/forum/index.php?t=msg&goto=325 14&#msg_32514

attached is a complete patch for that.. (patch8)

```
 //cdouble to value conversion test
 Value v = RichToValue(y);
 v = y;

 //Value to cdouble conversion test
// x = v; //ambiguity in std::complex::operator=
 x = v.operator cdouble(); //needs explicit call
#if flagMSC
 //works for MSC, GCC has constructor amiguity
 x = cdouble(v);
 x = (cdouble)v;
#endif

// x = Null; //ambiguity in std::complex::operator=
 x = Null.operator cdouble(); //needs explicit call
#if flagMSC
 //works for MSC, GCC has constructor amiguity
 x = cdouble(Null);
 x = (cdouble)Null;
#endif
```

EDIT:

Topt.h:227

+NTL_MOVEABLE(cdouble);

File Attachments

1) patchcomplex.patch, downloaded 550 times

2) ComplexTest.rar, downloaded 255 times
3) patch8.patch, downloaded 279 times

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by mirek on Sat, 25 Jun 2011 17:48:47 GMT
View Forum Message <> Reply to Message

Well, I am afraid that Null issue is actually unsolvable without encapsuplating std::complex... IMO, it is good enough reason to do so..

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by mirek on Sun, 26 Jun 2011 07:30:14 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Wed, 22 June 2011 07:12

```
// x = Null; //ambiguity in std::complex::operator=
 x = Null.operator cdouble(); //needs explicit call
```

Well, thought that through and it now seems to me that we can easily do this after all. See this (U++ without complex support):

```
#include <Core/Core.h>
#include <complex>

using namespace Upp;

typedef std::complex<double> complex;

CONSOLE_APP_MAIN
{
 complex x = Null;
 DDUMP(x.real());
}
```

-> we do not need to add complex to Nuller, it is enough to exploit Nuller::operator double() here

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Sun, 26 Jun 2011 08:47:19 GMT
View Forum Message <> Reply to Message

thats nice idea...

one remains though..

the assignment from Value

// x = v; //ambiguity in std::complex::operator=

EDIT: tried the version with derive..

```
struct cdouble : std::complex<double>
{
 typedef std::complex<double> C;
 cdouble() {}
 cdouble(double r) : C(r) {}
 cdouble(double r, double i) : C(r,i) {}
 cdouble(const Nuller&) { operator=(DOUBLE_NULL); }
};
```

it works, but i've got problems with those COMPARE_VALUE() things..again ambiguity, due to some template operator specifications as it seems..

any idea?

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Sun, 26 Jun 2011 14:05:13 GMT
View Forum Message <> Reply to Message

ok this all doesnt help..we got a semantic problem here.

complex can be constructed both from double and anther complex.
same applies for operator=, it can assign a double or a complex.

now, for Value support (and Nuller) we have both operator double() and operator cdouble() implicit conversion, that applies here.

so the compiler could construct a cdouble from a Value (or Nuller) in 2 paths, converting it to double or cdouble. and it does not know which one to prefer.

cdouble x = val.operator double();
cdouble x = val.operator cdouble();

both work, same for Null.

thats due to the implicit handling of double as a sole real part in complex context in std::complex. so there is no way around that. not even deriving from std::complex, unless we spare out the complex(double) ctor and operator=(double).

and it only touches Value conversion stuff. one could leave out the Value support, but it's bad too. so at least to have the Value support like that (dealing with explicit conversion access in this special context) is an advantage, at least the user could know what to call if the compiler doesnt.

RESULT: though it's 'ugly' to explicitly call Null.operator cdouble() and v.operator cdouble(), it's a reasonable price to pay for a common std::complex usage which we dont need to code oursselves anymore. Nuller.operator cdouble() even could be left out, so it only would apply to Value.operator cdouble() to call. but i'd keep it the same semantic, Null.operator cdouble() as well.

is it worth it, that's the final question? everything else works..

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by mirek on Mon, 27 Jun 2011 16:04:30 GMT
View Forum Message <> Reply to Message

I think it is possible that constructor from const Value& might improve the situation.

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Tue, 28 Jun 2011 07:04:43 GMT
View Forum Message <> Reply to Message

do you mean explicit Value(const cdouble&) ?

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by mirek on Tue, 28 Jun 2011 07:16:34 GMT
View Forum Message <> Reply to Message

No, cdouble(const Value&).

See e.g. Color...

Not sure if that helps, but I guess it is worth of try...

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Tue, 28 Jun 2011 07:22:09 GMT
View Forum Message <> Reply to Message

then the go is for deriving? i'll try.

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Tue, 05 Jul 2011 11:28:07 GMT
View Forum Message <> Reply to Message

got sth workin.. i think it would do it. thanks for the hints..

attached are Complex.h and Complex.cpp for Core.
the attached patch is to include them properly...

cdouble is derived from std::complex. i wanted cdouble to have as little code as possible and use native external means, except for where not otherwise possible like Value conversion.

if this could go to Core, the posted plugin/kissfft (see above) is also possible..

test

```
void Test(const cdouble& c) { RLOG(c); }

CONSOLE_APP_MAIN
{
 cdouble y(1.,5.);
 cdouble x = 12;

 //cdouble handling
 double d = y.real();
 y+=3.;

 //cdouble to value conversion test
 Value v = RichToValue(y);
 v = y;
 int type = v.GetType();

 Vector<cdouble> vc;
 vc.Add(12);

 //Value to cdouble conversion test
 x = v;

 //Null handling
 Test(Null);
```

```
 x = Null;

 bool b;
 b = (v == x);
 b = (x == v);
 b = (v != x);
 b = (x != v);
}
```

File Attachments

1) comp.svn.patch, downloaded 324 times
2) Complex.h, downloaded 352 times
3) Complex.cpp, downloaded 575 times

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by mirek on Wed, 06 Jul 2011 11:17:52 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Tue, 05 July 2011 07:28got sth workin.. i think it would do it. thanks for the hints..

attached are Complex.h and Complex.cpp for Core.
the attached patch is to include them properly...

cdouble is derived from std::complex. i wanted cdouble to have as little code as possible and use native external means, except for where not otherwise possible like Value conversion.

if this could go to Core, the posted plugin/kissfft (see above) is also possible..

test

```
void Test(const cdouble& c) { RLOG(c); }

CONSOLE_APP_MAIN
{
 cdouble y(1.,5.);
 cdouble x = 12;

 //cdouble handling
 double d = y.real();
 y+=3.;

 //cdouble to value conversion test
 Value v = RichToValue(y);
 v = y;
```

```
 int type = v.GetType();

 Vector<cdouble> vc;
 vc.Add(12);

 //Value to cdouble conversion test
 x = v;

 //Null handling
 Test(Null);
 x = Null;

 bool b;
 b = (v == x);
 b = (x == v);
 b = (v != x);
 b = (x != v);
}
```

Would you mind if instead of "cdouble" we use either complex or Complex?

I think integer complex numbers are unlikely and is we later decide to support 'float', we can still use something like complex32 or so...

Mirek

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Wed, 06 Jul 2011 12:48:44 GMT
View Forum Message <> Reply to Message

i dont mind at all.. vote for Complex..complex would probably have issues with std::complex, where users do using namespace std

---

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by mirek on Wed, 06 Jul 2011 15:25:23 GMT
View Forum Message <> Reply to Message

Thanks, with some minor changes applied.

One question: Should we consider polyequal for Complex vs double? (I mean, number+0i == number)

Or perhaps even type conversions where possible? As I see it, complex could be constructed from any number based Value as well, correct?

Mirek

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Wed, 06 Jul 2011 15:50:47 GMT
View Forum Message <> Reply to Message

ofcorse...

to complex is probably semantically right.
i'd say: int/bool/double -> complex
what to do with complex -> *?
in case of i0 it's ok, but to raise exception or assert here could prove fatal

cheers and thanks.

PS: what to do with kissfft plugin? will you upload it?

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by mirek on Wed, 06 Jul 2011 17:02:08 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Wed, 06 July 2011 11:50ofcorse...

to complex is probably semantically right.
i'd say: int/bool/double -> complex
what to do with complex -> *?
in case of i0 it's ok, but to raise exception or assert here could prove fatal

cheers and thanks.

PS: what to do with kissfft plugin? will you upload it?

I think right now the right place is bazaar. I know it is potentially very usefull package, but by including it to uppsrc I would have to take certain level of responsibility about its maintainace, which I am right now not willing to do...

Mirek

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by mirek on Wed, 06 Jul 2011 17:03:51 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Wed, 06 July 2011 11:50ofcorse...

to complex is probably semantically right.
i'd say: int/bool/double -> complex
what to do with complex -> *?
in case of i0 it's ok, but to raise exception or assert here could prove fatal

I think number->complex is fine and easy to do.

I think complex->number is both harder to implement and badly defined, so let us forget about this.

Mirek

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by mirek on Wed, 06 Jul 2011 17:31:31 GMT
View Forum Message <> Reply to Message

Implemented and commited...

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Wed, 06 Jul 2011 19:48:17 GMT
View Forum Message <> Reply to Message

thank you very much..
should i repost you the kissfft current state?

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by Zbych on Wed, 06 Jul 2011 20:19:00 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Wed, 06 July 2011 17:50
what to do with complex -> *?

I would suggest modulus (sqrt(r^2+i^2)) for doubles and ints.

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by unknown user on Wed, 06 Jul 2011 21:23:37 GMT
View Forum Message <> Reply to Message

Hi,

Revision r3624 does not compile under g++, because Complex.h@14:
Complex(const Value& v) : C(IsNumber(v) ? C((double)v) : RichValue<Complex>::Extract(v)) {}
Error: no match for ternary 'operator?:'

Maybe RichValue<Complex>::Extract(v) should be cast to Complex?

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Wed, 06 Jul 2011 21:38:34 GMT
View Forum Message <> Reply to Message

nope, Extract already returns Complex, but the ternary expects same type in first part as well..

change C to Complex:
Complex.h:14

Complex(const Value& v) : C(IsNumber(v) ? Complex((double)v) :
RichValue<Complex>::Extract(v)) {}

BTW: nice idea with modulus. is it mathematically semantically well placed?

EDIT: i adjusted kissfft, here comes the package for plugin. if it's ok for you...

File Attachments
1) kissfft.rar, downloaded 301 times

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by mirek on Thu, 07 Jul 2011 06:19:22 GMT
View Forum Message <> Reply to Message

Zbych wrote on Wed, 06 July 2011 16:19kohait00 wrote on Wed, 06 July 2011 17:50
what to do with complex -> *?

I would suggest modulus (sqrt(r^2+i^2)) for doubles and ints.

I do not think it is a good idea. While there are situations where this is required, it should not happen behind the scene.

It is alike performing "abs" for int -> unsigned conversion...

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Thu, 07 Jul 2011 06:23:25 GMT
View Forum Message <> Reply to Message

i see in this case one-way conversion as the better strategy, just as in real world, the transition to complex world is one-way..

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by mirek on Thu, 07 Jul 2011 15:40:27 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Thu, 07 July 2011 02:23i see in this case one-way conversion as the better strategy, just as in real world, the transition to complex world is one-way..

Then we are finished now with complex, right?

Good job, kohait

Mirek

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Mon, 25 Jul 2011 08:41:06 GMT
View Forum Message <> Reply to Message

is there any interest adding kissfft as plugin?

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by mirek on Sat, 30 Jul 2011 08:11:44 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Mon, 25 July 2011 04:41is there any interest adding kissfft as plugin?

For now, I would like to see it in Bazaar (you can even start bazaar plugin). The, perhaps, if there is a demand, we can review and move to uppsrc.

Mirek

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by kohait00 on Sun, 31 Jul 2011 13:41:32 GMT
View Forum Message <> Reply to Message

done bazaar/plugin was already there..

Subject: Re: [DISCUSSION] Add 'complex' datatype, to Value too
Posted by koldo on Mon, 01 Aug 2011 07:50:11 GMT
View Forum Message <> Reply to Message

Ah!

Bazaar/plugin, seems a good idea.

I will move some things there.