vsip_vputoffset_f, vsip_cvputoffset_f

Vector Put Offset

Put (Set) the offset attribute of a vector view object.

Functionality

This function puts (sets) the offset (in elements) to the first scalar element of a vector view, from the start of the block object's data array, to which it is bound.

Prototypes

```
vsip_vview_f *vsip_vputoffset_f(
   vsip_vview_f *v,
   vsip_offset offset);

vsip_vview_f *vsip_cvputoffset_f(
   vsip_cvview_f *v,
   vsip_offset offset);
```

Arguments

`*v`

Vector view object.

`offset`

Offset in elements relative to the start of the block object.

Return Value

Returns a pointer to the source vector view object as a programming convenience.

Restrictions

None.

Requirements

The arguments must conform to the following:

- The vector view object must be valid.
- The offset argument must not specify a vector view that exceeds the bounds of the data array of the associated block.

vsip_vputstride_f, vsip_cvputstride_f

Vector Put Stride

Put (Set) the stride attribute of a vector view object.

Functionality

Put (Set) the stride attribute of a vector view object. Stride is the distance in elements of the block's data array between successive elements of the vector view.

Prototypes

```
vsip_vview_f *vsip_vputstride_f(
   vsip_vview_f *v,
   vsip_stride stride);

vsip_vview_f *vsip_cvputstride_f(
   vsip_cvview_f *v,
   vsip_stride stride);
```

Arguments

`*v`

Vector view object.

`stride`

Stride in elements.

Return Value

Returns a pointer to the source vector view object as a programming convenience.

Restrictions

None.

Requirements

The arguments must conform to the following:

- The vector view object must be valid.
- The stride argument must not specify a vector view that exceeds the bounds of the data array of the associated block.

vsip_vrealview_f

Create Real Vector View

Create a vector view object of the real part of a complex vector from a complex vector view object.

Functionality

This function creates a real vector view object from the "real part of a complex" vector view object, or returns null if it fails.

On success, the function creates a derived block object (derived from the complex block object). The derived block object is bound to the real data part of the original complex block and then binds a real vector view object to the block. The new vector encompasses the real part of the input complex vector.

Prototypes

```
vsip_vview_f *vsip_vrealview_f(
   const vsip_cvview_f *v);
```

Arguments

`*v`

Source vector view object.

Return Value

Returns a pointer to the created "real" vector view object, or null if the memory allocation for new object fails.

Restrictions

The derived block object cannot be destroyed or released. The parent complex block object may be released (if it is bound to user data). Destroying the complex block is the only way to free the memory associated with the derived block object.

Requirements

The arguments must conform to the following:

- The complex vector view object must be valid.

Notes

It is important for the application to check the return value for a memory allocation failure.

This function should not be confused with the function `vsip_sreal_f`, which is a copy operator (copies the real data).

There are no requirements on offset or stride of a real view on its derived block. By using `vsip_vgetattrib_f`, information about the layout of the view and the block may be obtained.

Caution: Using attribute information, and the block bound to the vector, to bind new vectors outside the data space of the original vector produced by `vsip_srealview_f` will produce non-portable code.

Portable code may be produced by:

- Remaining inside the data space of the vector,
- Not assuming a set relationship of strides and offsets, and
- Using the get attributes functions to obtain necessary information within the application code to understand the layout for each implementation.

Examples

See example with `vsip_vimagview_f`.

See Also

`vsip_vcloneview_f`, `vsip_cvcloneview_f`,
`vsip_vimagview_f`,
`vsip_vsubview_f`, `vsip_cvsubview_f`

vsip_vsubview_f, vsip_cvsubview_f

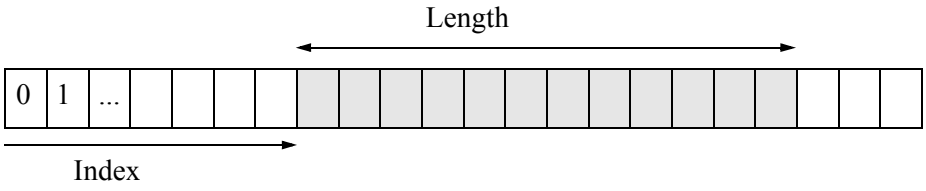
Create Subview Vector View

Create a vector view object that is a subview of a vector view object (offset, and length are relative to the source view object, not the bound block object).

Functionality

This function creates a subview vector view object from a source vector view object, and binds it to the same block object, or returns null if it fails. The zeroth element of the new subview corresponds to the index element of the source vector view.

(The subview is relative to the source view, and stride is inherited from the source view).



Prototypes

```
vsip_vview_f *vsip_vsubview_f(
    const vsip_vview_f *v,
    vsip_index index,
    vsip_length length);

vsip_cvview_f *vsip_cvsubview_f(
    const vsip_cvview_f *v,
    vsip_index index,
    vsip_length length);
```

Arguments

`*v`

Source vector view.

`index`

The subview vectors first element (`index 0`) is at vector index `index` of the source vector.

`length`

Length in elements of new vector view.

Return Value

Returns a pointer to the created subview vector view object, or null if the memory allocation for new object fails.

Restrictions

None.

Requirements

The arguments must conform to the following:

- The vector view object must be valid.
- The length must be positive.
- The subview must not extend beyond the bounds of the source view.

Notes

It is important for the application to check the return value for a memory allocation failure.

Examples

None.

See Also

`vsip_vcloneview_f`, `vsip_cvcloneview_f`,
`vsip_vimagview_f`,
`vsip_vrealview_f`

vsip_cmplx_f

Form Complex Scalar

Form a complex scalar from two real scalars.

Functionality

$$r \leftarrow a + ib, \quad a, b \in \Re$$

Prototypes

Return by argument reference:

```
void vsip_CMPLX_f(
   vsip_scalar_f a,
   vsip_scalar_f b,
   vsip_cscalar_f *r);
```

Return by return value:

```
vsip_cscalar_f vsip_cmplx_f(
   vsip_scalar_f a,
   vsip_scalar_f b);
```

Arguments

`a`

Real part: real scalar argument

`b`

Imaginary part: real scalar argument

`*r`

Pointer to output: complex scalar

Return Value

Returns complex scalar.

Restrictions

The return by argument reference form may be implemented as a macro and may have restrictions on its usage. The return by argument reference form may be implemented as a macro and may have restrictions on its usage. The return by

argument reference form may be implemented as a macro and may have restrictions on its usage.

Requirements

None.

Notes

None.

Examples

None.

See Also

vsip_imag_f, vsip_real_f

vsip_imag_f

Complex Scalar Imaginary

Extract the imaginary part of a complex scalar.

Functionality

$$r \leftarrow \mathrm{Im}\{a\}$$

Prototypes

Return by return value:

```
vsip_scalar_f vsip_imag_f(
   vsip_cscalar_f a);
```

Arguments

`a`

Complex scalar argument

Return Value

Returns a real scalar containing the imaginary part of the complex scalar input.

Restrictions

This function may be implemented as a macro and may have restrictions on its usage.

Requirements

None.

Examples

None.

See Also

`vsip_cmplx_f`, `vsip_real_f`

vsip_real_f

Complex Scalar Real

Extract the real part of a complex scalar.

Functionality

$$r \leftarrow \mathrm{Re}\{a\}$$

Prototypes

Return by return value:

```
vsip_scalar_f vsip_real_f(
   vsip_cscalar_f a);
```

Arguments

`a`

Complex scalar argument

Return Value

Returns a real scalar containing the real part of the complex scalar input.

Restrictions

This function may be implemented as a macro and may have restrictions on its usage.

Requirements

None.

Examples

None.

See Also

`vsip_cmplx_f`, `vsip_imag_f`

vsip_svadd_f

Scalar Vector Add

Computes the sum, by element, of a scalar and a vector.

Functionality

$$r_j \leftarrow a + b_j$$

for j = 0, 1, ..., N-1

Prototypes

```
void vsip_svadd_f(
   vsip_scalar_f alpha,
   const vsip_vview_f *b,
   const vsip_vview_f *r);
```

Arguments

`alpha`

Input scalar

`*b`

View of input vector

`*r`

View of output vector

Return Value

None.

Requirements

The arguments must conform to the following:

- Input and output views must all be the same size.
- All view objects must be valid.
- The input and output views must be identical views of the same block (in-place), or must not overlap.

Examples

```
/* Example of scalar vector add */
#include<stdio.h>
#include "vsip.h"
#define L 7 /* length */
int main()

{
    int i;
    vsip_scalar_f dataLeft;
    vsip_vview_f* dataRight = vsip_vcreate_f(L,0);
    vsip_vview_f* dataSum = vsip_vcreate_f(L,0);
    /* Make up some data to find the magnitude of */
    /* First set the scalar equal to 1*/
    dataLeft = 1.0;
    /* Then compute a ramp from one to minus one */
    vsip_vramp_f(1.0, -2.0/(double)(L-1), dataRight);
    /* Add the scalar and the vector */
    vsip_svadd_f(dataLeft, dataRight, dataSum);

    /* now print out the data and its sum */
    for(i=0; i<L; i++)
       printf("%7.4f = (%7.4f) + (%7.4f) \n",
          vsip_vget_f(dataSum,i),
          dataLeft, vsip_vget_f(dataRight,i));
    /* destroy the vector views and any associated blocks */
    vsip_blockdestroy_f(vsip_vdestroy_f(dataRight));
    vsip_blockdestroy_f(vsip_vdestroy_f(dataSum));
    return 0;
    }

/* output */
/* 2.0000 = ( 1.0000) + ( 1.0000)
    1.6667 = ( 1.0000) + ( 0.6667)
    1.3333 = ( 1.0000) + ( 0.3333)
    1.0000 = ( 1.0000) + ( 0.0000)
    0.6667 = ( 1.0000) + (-0.3333)
    0.3333 = ( 1.0000) + (-0.6667)
    0.0000 = ( 1.0000) + (-1.0000) */
```

See Also

vsip_vadd_f, vsip_cvadd_f

vsip_svdiv_f

Scalar Vector Divide

Computes the quotient, by element, of a scalar and a vector.

Functionality

$$r_j \leftarrow \frac{a}{b_j}$$

for j = 0, 1, ..., N-1

Prototypes

```
void vsip_svdiv_f(
   vsip_scalar_f alpha,
   const vsip_vview_f *b,
   const vsip_vview_f *r);
```

Arguments

`alpha`

Input scalar

`*b`

View of input vector

`*r`

View of output vector

Restrictions

The result of division by zero is implementation dependent.

Requirements

The arguments must conform to the following:

- Input and output views must all be the same size.
- All view objects must be valid.

- The input and output views must be identical views of the same block (in-place), or must not overlap.

Examples

```
/* example of scalar-vector divide */
#include<stdio.h>
#include "vsip.h"
#define L 5
int main()
{

    int i;
    /* define some data space */
    vsip_cvview_f* dataComplex = vsip_cvcreate_f(L,0);
    vsip_cscalar_f scalarComplex;
    vsip_cvview_f* dataComplexQuotient = vsip_cvcreate_f(L,0);
    /* put some complex data in dataComplex */
    for(i = 0; i < L; i++)
    vsip_cvput_f(dataComplex,i,vsip_cmplx_f((double)(i * i), (double)(i
        + 1)));
    /* define a complex scalar */
    scalarComplex = vsip_cmplx_f(3,4);
    /*divide scalarComplex by dataComplex and print the input and output */
    vsip_csvdiv_f(scalarComplex, dataComplex, dataComplexQuotient);

    for(i=0; i<L; i++)
        printf("(%7.4f + %7.4fi) / (%7.4f + %7.4fi) = (%7.4f + %7.4fi)\n",
            vsip_real_f(scalarComplex),
            vsip_imag_f(scalarComplex),
            vsip_real_f(vsip_cvget_f(dataComplex,i)),
            vsip_imag_f(vsip_cvget_f(dataComplex,i)),
            vsip_real_f(vsip_cvget_f(dataComplexQuotient,i)),
            vsip_imag_f(vsip_cvget_f(dataComplexQuotient,i)));
    vsip_cblockdestroy_f(vsip_cvdestroy_f(dataComplex));
    vsip_cblockdestroy_f(vsip_cvdestroy_f(dataComplexQuotient));
    return 0;
}

/* output */
/* ( 3.0000 + 4.0000i) / ( 0.0000 + 1.0000i) = ( 4.0000 + -3.0000i)
    ( 3.0000 + 4.0000i) / ( 1.0000 + 2.0000i) = ( 2.2000 + -0.4000i)
    ( 3.0000 + 4.0000i) / ( 4.0000 + 3.0000i) = ( 0.9600 + 0.2800i)
    ( 3.0000 + 4.0000i) / ( 9.0000 + 4.0000i) = ( 0.4433 + 0.2474i)
    ( 3.0000 + 4.0000i) / (16.0000 + 5.0000i) = ( 0.2420 + 0.1744i) */
```

See Also

vsip_svdiv_f, vsip_vdiv_f

vsip_svmul_f, vsip_csvmul_f, vsip_rscvmul_f

Scalar Vector Multiply

Computes the product, by element, of a scalar and a vector.

Functionality

$$r_j \leftarrow a \cdot b_j$$

for j = 0, 1, L, N-1

Prototypes

```
void vsip_svmul_f(
   vsip_scalar_f alpha,
   const vsip_vview_f *b,
   const vsip_vview_f *r);

void vsip_csvmul_f(
   vsip_cscalar_f alpha,
   const vsip_cvview_f *b,
   const vsip_cvview_f *r);

void vsip_rscvmul_f(
   vsip_scalar_f alpha,
   const vsip_cvview_f *b,
   const vsip_cvview_f *r);
```

Arguments

`alpha`

Input scalar

`*b`

View of input vector

`*r`

View of output vector

Requirements

The arguments must conform to the following:

- Input and output views must all be the same size.
- All view objects must be valid.
- The input and output views must be identical views of the same block (in-place), or must not overlap.

Examples

```
/* example of scalar-vector multiply */
#include<stdio.h>
#include "vsip.h"
#define L 5
int main()

{
    int i;
    vsip_cvview_f* dataComplex = vsip_cvcreate_f(L,0);
    vsip_cscalar_f scalarComplex;
    vsip_cvview_f* dataComplexProduct = vsip_cvcreate_f(L,0);
    /* put some complex data in dataComplex */
    for(i = 0; i < L; i++)
        vsip_cvput_f(dataComplex,i,vsip_cmplx_f((double)(i * i),
        (double)(i + 1))
        );
    /* define a complex scalar */
    scalarComplex = vsip_cmplx_f(3,4);

    /* Multiply scalarComplex by dataComplex, print the input and output: */
    vsip_csvmul_f(scalarComplex,dataComplex,dataComplexProduct);
    for(i=0; i<L; i++)
        printf("(%7.4f + %7.4fi) * (%7.4f + %7.4fi) = (%7.4f + %7.4fi)\n",
            vsip_real_f(scalarComplex),
            vsip_imag_f(scalarComplex),
            vsip_real_f(vsip_cvget_f(dataComplex,i)),
            vsip_imag_f(vsip_cvget_f(dataComplex,i)),
            vsip_real_f(vsip_cvget_f(dataComplexProduct,i)),
            vsip_imag_f(vsip_cvget_f(dataComplexProduct,i)));

    vsip_cblockdestroy_f(vsip_cvdestroy_f(dataComplex));
    vsip_cblockdestroy_f(vsip_cvdestroy_f(dataComplexProduct));
    return 0;
    }
```

```
/* output */
/* ( 3.0000 + 4.0000i) * ( 0.0000 + 1.0000i) = (-4.0000 + 3.0000i)
    ( 3.0000 + 4.0000i) * ( 1.0000 + 2.0000i) = (-5.0000 + 10.0000i)
    ( 3.0000 + 4.0000i) * ( 4.0000 + 3.0000i) = ( 0.0000 + 25.0000i)
    ( 3.0000 + 4.0000i) * ( 9.0000 + 4.0000i) = (11.0000 + 48.0000i)
    ( 3.0000 + 4.0000i) * (16.0000 + 5.0000i) = (28.0000 + 79.0000i) */
```

See Also

vsip_vmul_f, vsip_cvmul_f, vsip_rcvmul_f