Blitz++ User's Guide

A C++ class library for scienti c computing
for version 0.10, 11 May 2012

The Blitz++ library is licensed under both the GPL and the more permissive \Blitz++ Artistic License". Take your pick. They are detailed in GPL and LICENSE, respectively. The artistic license is more appropriate for commercial use, since it lacks the \viral" properties of the GPL.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Puo-33333(Pu39pt,Av21(spt,ASee)-331(for)-331(more)-33d

Table of Contents

Short Contents

.
.

Then go into the `blitz-VERSION

```
    $(CXX) $(CXXFLAGS) -c $*.cpp

$(TARGETS):
    $(CXX) $(LDFLAGS) $@.o -o $@ $(LIBS)

all:
    $(TARGETS)
```

Nested homogeneous arrays using `TinyVector` and `TinyMatrix`, in which each element array is laid out in memory. By altering the contents of a `GeneralArrayStorage<N>` object,

```
Array(int extent1, GeneralArrayStorage<N_rank> storage);
```

2.3.6 Constructing an array from an expression

Arrays may be constructed from expressions, which are described in Chapter 3 [Array Expressions], page 35. The syntax is:

```
Array(...array expression...);
```

For example, this code creates an array B which contains the square roots of the elements in A:

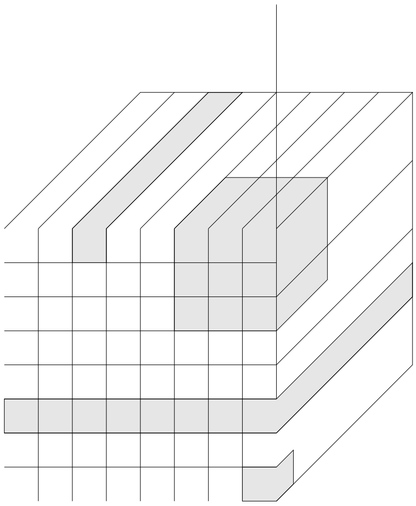
```
Array<float,2> A(N,N);   // ...
Array<float,2> B(sqrt(A));
```

2.3.7 Creating an array from pre-existing data

When creating an array using a pointer to already existing data, you have three choices for how Blitz++ will handle the data. These choices are enumerated by the enum type `preexistingMemoryPolicy`:

```
enum preexistingMemoryPolicy {
  duplicateData,
  deleteDataWhenDone,
  neverDeleteData
};
```

If you choose `duplicateData`, Blitz++ will create an array object using a copy of the data you provide. If you choose `deleteDataWhenDone`, Blitz++ will not create a copy of the,6h7 -841944 -13.15 T
Fte5(an)-283(oz)]TJ/F56 10.9091 Tf5[(20945 0 Td



```
Array<int,2> D(Range(1,5), Range(1,5));      // 1..5, 1..5
Array<int,2> E = D(Range(2,3), Range(2,3)); // 1..2, 1..2
```

An array can be used on both sides of an expression only if the subarrays don't overlap. If the arrays overlap, the result may depend on the order in which the array is traversed.

2.4.3 RectDomain and StaidedDomain

The classes RectDomain and StaidedDomain, de ned in blitz/domain.h, o er a dimension-independent notation for subarrays.

RectDomain and StaidedDomain can be thought of as a TinyVector<Range,N>. Both have a vector of lower- and upper-bounds; StaidedDomain has a staide vector. For example, the subarray:

```
Array<int,2> B = A(Range(4,7), Range(8,11));  // 4..7, 8..11
```

could be obtained using RectDomain this way:

```
TinyVector<int,2> lowerBounds(4, 8);
TinyVector<int,2> upperBounds(7, 11);
RectDomain<2> subdomain(lowerBounds, upperBounds);

Array<int,2> B = A(subdomain);
```

Here are the prototypes of RectDomain and StaidedDomain.

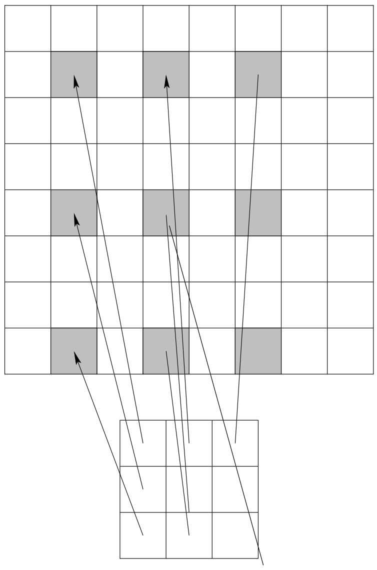
```
template<int N_rank>
class RectDomain {

public:
    RectDomain(const TinyVector<int,N_rank>& lbound,
        const TinyVector<int,N_rank>& ubound);

    const TinyVector<int,N_rank>& lbound() const;
    int lbound(int i) const;
    const TinyVector<int,N_rank>& ubound() const;
    int ubound(int i) const;
    Range operator[](int rank) const;
    void shrink(int amount);
    void shrink(int dim, int amount);
    void expand(int amount);
    void expand(int dim, int amount);
};

template<int N_rank>
class StaidedDomain {

public:
    StaidedDomain(const TinyVector<int,N_rank>& lbound,
        const TinyVector<int,N_rank>& ubound,
        const TinyVector<int,N_rank>& staide);

    const TinyVector<int,N_rank>& lbound() const;
    int lbound(int i) const;
    const TinyVector<int,N_rank>& ubound() const;
    int ubound(int i) const;
    const TinyVector<int,N_rank>& staide() const;
```

```
    int stride(int i) const;
    Range operator[](int rank) const;
    void shrink(int amount);
    void shrink(int dim, int amount);
    void expand(int amount);
    void expand(int dim, int amount);
};
```

2.4.4 Slicing

```
using namespace blitz;

int main()
{
    Array<int,2> A(8,8);
    A = 0;

    Array<int,2> B = A(Range(1,7,3), Range(1,5,2));
    B = 1;

    cout << "A = " << A << endl;
    return 0;
}
```

Here's an illustration of the B subarray:



B
'sing strides to create non-contiguous subarrays.

And the program output:

```
A = (0,7) x (0,7)
[ 0 0 0 0 0 0 0 0
  0 1 0 1 0 1 0 0
  0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0
  0 1 0 1 0 1 0 0
  0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0
  0 1 0 1 0 1 0 0 ]
```

2.4.6 A note about assignment

The assignment operator (=) always results in the expression on the right-hand side (rhs) being *copied* to the lhs (i.e. the data on the lhs is overwritten with the result from the rhs). This is di erent from some array packages in which the assignment operator makes the lhs a reference (or alias) to the rhs. To further confuse the issue, the copy constructor for arrays *does* have

Statement 1 results in a portion of B

In debugging mode, your programs will run *very slowly*. This is because Blitz++ is doing lots of precondition checking and bounds checking. When it detects something shy, it will likely halt your program and display an error message.

For example, this program attempts to access an element of a 4x4 array which doesn't exist:

```
#include <blitz/array.h>

using namespace blitz;

int main()
{
    Array<complex<float>, 2> Z(4,4);

    Z = complex<float>(0.0, 1.0);

    Z(4,4) = complex<float>(1.0, 0.0);

    return 0;
}
```

When compiled with `-DBZ_DEBUG`, the out of bounds indices are detected and an error message results:

```
[Blitz++] Precondition failure: Module ../../blitz/array-impl.h line 1339
Array index out of range: (4, 4)
Lower bounds: (0,0)
Length:       (4,4)

debug: ../../blitz/array-impl.h:1339: bool blitz::Array<P_numtype,
N_rank>::assertInRange(int, int) const [with P_numtype = std::complex<float>,
int N_rank = 2]: Assertion `0' failed.
```

Precondition failures send their error messages to the standard error stream (`cerr`). After displaying the error message, `assert(0)`

```
A.reverse(thirdDim);
```

This code is clearer: you can see that the parameter refers to a dime7sio7, and it isn't much of a leap to realize t 33(it's)-333(rev)28(e)-1(r)1(s)-1(i)1(ng)-334(t)e element ordering in the trd dimensio7.

If you nd `firstDim`, `secondDim`, ... aesthetically u7pleasi7g, there are equivalent symbols `firstRank`, `secondRank`, `thirdRank`, ..., `eleventhRank`.

```
const T_numtype* [restrict]       data() const;
      T_numtype* [restrict]       data();
const T_numtype* [restrict]       dataZero() const;
      T_numtype* [restrict]       dataZero();
const T_numtype* [restrict]       dataFirst() const;
      T_numtype* [restrict]       dataFirst();
```

These member functions all return pointers to the array data. The NCEG `restrict` quali er

Now the B array refers to the 2nd component of every element in A. Note: for complex

A new array is allocated to contain the result. To avoid copying the result array, you should use it as a constructor argument. For example:

2.8 Inputting and Outputting Arrays

2.8.1 Output formatting

The current version of Blitz++ includes rudimentary output formatting for arrays. Here's an example:

```
#include <blitz/array.h>

using namespace blitz;

int main()
{
    Array<int,2> A(4,5,FortranArray<2>());
    firstIndex i;
    secondIndex j;
    A = 10*i + j;

    cout << "A = " << A << endl;

    Array<float,1> B(20);
    B = exp(-i/100.);
```

```
#include <blitz/array.h>
#ifdef BZ_HAVE_STD
#include <fstream>
#else
#include <fstream.h>
#endif

BZ_USING_NAMESPACE(blitz)

const char* filename = "io.data";

void write_arrays()
{
    ofstream ofs(filename);
    if (ofs.bad())
    {
        cerr << "Unable to write to file: " << filename << endl;
        exit(1);
    }

    Array<float,3> A(3,4,5);
    A = 111 + tensor::i + 10 * tensor::j + 100 * tensor::k;
    ofs << A << endl;

    Array<float,18]TJ0fe<
    Afs 111A+<225(endli)]TJ0-20120.921Cd[(ray<float,18]TJ0f25(endC051;)]TJ0-1061Td[(A)-525(=)-525(100)-52
```

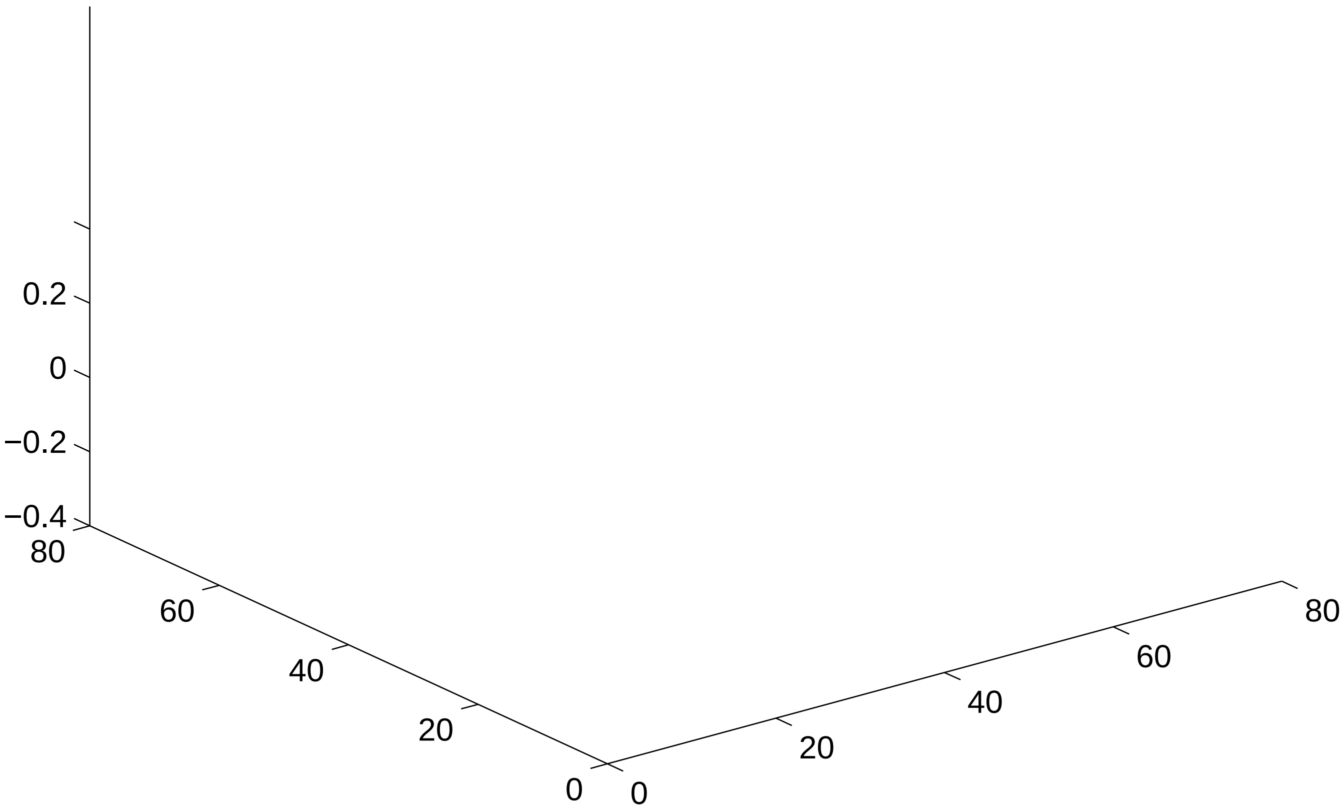
```
Array<int,2> A(3, 3, FortranArray<2>());
```

The third parameter, `FortranArray<2>()`, tells the `Array` constructor to use a 1ΩF4me format tells the [illegible] appropriate dimensional [illegible]

```
  7 8 9 ]

C = (0,2) x (0,2)
[ 1 2 3
  4 5 6
  7 8 9 ]

7 8[ 1(0,2)[ 100,2)          1[ 1[ 1[ 1x[x (0,2.461Td[([)-525(1(0,2) x (0,2)
[ 1 24 5
```

```
float tau = - 10.0 / N;

// Index placeholders
firstIndex i;
secondIndex j;

// Fill the array
F = cos(omega * sqrt(pow2(i-midpoint) + pow2(j-midpoint)))
    * exp(tau * sqrt(pow2(i-midpoint) + pow2(j-midpoint)));
```

Here's a plot of the resulting array:



Array lled using an index placeholder expression.

You can use index placeholder expressions in up to 11 dimensions. Here's a three dimensionalIndexexample:

`hypot(x,y)`
Computes so that under ow does not occur and over ow occurs only if the nal result warrants it.

`nextafter(x,y)`
Returns the next representable number after x in the direction of y.

`remainder(x,y)`

```
    return 0;
}
```

```
for (int n=0; n < 4; ++n)
  for (int m=0; m < 4; ++m)
    A(n,m) = cos(x(n)) * sin(y(m));
```

The functions `cos` and

`all()` True if the expression is true everywhere (bool)

`first()` First index at which the expression is logical true (int); if the expression is logical true nowhere, then `tiny(int())` (INT_MIN) is returned.

`last()` Last index at which the expression is logical true (int); if the expression is logical true nowhere, then `huge(int())` (INT_MAX) is returned.

The reductions `any()`, `all()`, and `first()` have short-circuit semantics: the reduction will halt as soon as the answer is known. For example, if you use

first(*0(the)35010.90kno6w1.( )]TJ-22001032-20.061Td[(T3fs)-329(kno)28(w]TJ0-13.129(as321.9

4 Stencils

Blitz++

The factor terms always consist of an integer multiplier (often 1) and a power of h. For ease of use, all of th(of)-381peratorsofelowofvidedofersionof whichofth(of)17(in)28(teger)-multiplierisof1. Thenormalizedversions haveanappendedtoofth(of)16(name,f)220(for)]TJ h mean tation.

4.4.1 Cer

These are available in multicomponent versions: for example, `central12(A,component,dimension)` gives the central12 operator for the specified component (Components are numbered 0, 1, ... N-1).

4.4.2 Forward differences

`forward11(A,dimension)`
1st derivative, 1st order accurate. Factor: h

0	1
-1	1

`forward21(A,dimension)`
2nd derivative, 1st order accurate. Factor: h^2

0	1	2
1	-2	1

`forward31(A,dimension)`
3rd derivative, 1st order accurate. Factor: h^3

0	1	2	3
-1	3	-3	1

`forward41(A,dimension)`
4th derivative, 1st order accurate. Factor: h^4

0	1	2	3	4
1	-4	6	-4	1

`forward12(A,dimension)`
1st derivative, 2nd order accurate. Factor: $2h$

0	1	2
-3	4	-1

`forward22(A,dimension)`
2nd derivative, 2nd order accurate. Factor: h^2

0	1	2	3
2	-5	4	-1

`forward32(A,dimension)`
3rd derivative, 2nd order accurate. Factor: $2h^3$

4.4.3 Backward di erences

`backward11(A,dimension)`

1st derivative, 1st order accurate. Factor: h

-1	0
-1	1

`backward21(A,dimension)`

2nd derivative, 1st order accurate. Factor: h^2

-2	-1	0
1	-2	1

`backward31(A,dimension)`

3rd derivative, 1st order accurate. Factor: h^3

-3	-2	-1	0
-1	3	-3	1

`backward41(A,dimension)`

4th derivative, 1st order accurate. Factor: h^4

-4	-3	-2	-1	0
1	-4	6	-4	1

`backward12(A,dimension)`

1st derivative, 2nd order accurate. Factor: $2h$

-2	-1	0
1	-4	3

`backward22(A,dimension)`

2nd derivative, 2nd order accurate. Factor: h^2

-3	-2	-1	0
-1	4	-5	2

`backward32(A,dimension)`

3rd derivative, 2nd order accurate. Factor: $2h^3$

-4	-3	-2	-1	0

-5	-4	-3	-2	-1	0
-2	11	-24	26	-14	3

Note that the above are available in normalized versions `backward11n`, `backward21n`, ..., `backward42n` which have factors of h, h^2, h^3, or h^4 as appropriate.

These are available in multicomponent versions: for example, `backward42(A,component,dimension)`

4.4.4 Laplacian (r^2) operators

`Laplacian2D(A)`

2nd order accurate, 2-dimensional laplacian. Factor: h^2

	-1	0	1
-1		1	
0	1	-4	1
1		1	

`Laplacian3D(A)`

2nd order accurate, 3-dimensional laplacian. Factor: h^2

`Laplacian2D4(A)`

4th order accurate, 2-dimensional laplacian. Factor: $12h^2$

	-2	-1	0	1	2

4.4.7 Grad-squared operators

There are also grad-squared operators, which return `TinyVectors` of second derivatives:

`gradSqr2D(A)`
: 2nd order, 2-dimensional grad-squared (vector of second derivatives), generated using the central22 operator. Factor: h^2

`gradSqr2D4(A)`
: 4th order, 2-dimensional grad-squared, using central24 operator. Factor: $12h^2$

`gradSqr3D(A)`
: 2nd order, 3-dimensional grad-squared, using the central22 operator. Factor: h^2

`gradSqr3D4(A)`
: 4th order, 3-dimensional grad-squared, using central24 operator. Factor: $12h^2$

Note that the above are available in normalized versions `gradSqr2Dn`, `gradSqr2D4n`, `gradSqr3Dn`, `gradSqr3D4n` which have factors h^2.

4.4.8 Curl (r) operators

These curl operators return scalar values:

`curl(Vx,Vy)`
: 2nd order curl operator using the central12 operator. Factor: $2h$

`curl4(Vx,Vy)`
: 4th order curl operator using the central14 operator. Factor: $12h$

`curl2D(V)`
: 2nd order curl operator on a 2D vector eld (e.g. `Array<TinyVector<float,2>,2>`), using the central12 operator. Factor: $2h$

`curl2D4(V)`
: 4th order curl operator on a 2D vector eld, using the central12 operator. Factor: $12h$

Available in normalized forms `curln`, `curl4n`, `curl2Dn`, `curl2D4n`.

These curl operators return three-dimensional `TinyVectors` of the app-opriate numeric type:

`curl(Vx,Vy,Vz)`
: 2nd order curl operator using the central12 operator. Factor: $2h$

`curl4(Vx,Vy,Vz)`
: 4th order curl operator using the central14 operator. Factor: $12h$

`curl(V)`
: 2nd order curl operator on a 3D vector eld (e.g. `Array<TinyVector<double,3>,3>`, using the central12 operator. Factor: $2h$

`curl4(V)`
: 4th order curl operator on a 3D vector eld, using the central14 operator. Factor: $12h$

Note that the above are available in normalized versions `curln` and `curl4n`, which factors of

```
div4(Vx,Vy)
```

5 Multicomponent, complex, and user type Arrays

5.1 Multicomponent and complex arrays

Multicomponent arrays have elements which are vectors. Examples of such arrays are vector

```
class HSV24 {
public:
    // These constants will makes the code below cleaner; we can
    // refer to the components by name, rather than number.

    static const int hue=0, saturation=1, value=2;

    HSV24() { }
    HSV24(int hue, int saturation, int value)
      : h_(hue), s_(saturation), v_(value)
    { }

    // Some other stuff here, obviously

private:
    unsigned char h_, s_, v_;
};
```

ght after the class declaration, we will invoke the macro BZ_DECLARE_MULTICOMPONENT_

Note: Blitz++ provides numerous math functions de ned over complex-valued arrays, such as `conj`, `polar`, `arg`, `abs`, `cos`, `pow`, etc. See the section on math functions (Section 3.8 [Math functions 1], page 40) for details.

5.1.3 Zipping together expressions

Blitz++ provides a function `zip()` which lets you combine two or more expressions into a single component. For example, you can combine two real expressions into a complex expression, or three integer expressions into an HSV24 expression. The function has this syntax:

```
resultexpr zip(expr1, expr2, T_element)
resultexpr zip(expr1, expr2, expr3, T_element)         ** not available yet
resultexpr zip(expr1, expr2, expr3, expr4, T_element)  ** not available yet
```

The types `resultexpr`, `expr1` and `expr2` are array expressions. The third argument is the type you want to create. For example:

```
int N = 16;
Array<complex<float>,1> A(N);
Array<float,1> theta(N);

 ...

A = zip(cos(theta), sin(theta), complex<float>());
```

The above line is equivalent to:

```
for (int i=0; i < N; ++i)
   A[i] = complex<float>(cos(theta[i]), sin(theta[i]));
```

5.2 Creating arrays of a user typ56 14.34T4encoit0,4a 10.S25(Ean)28(t)-33

```
    FixedPoint operator+(FixedPoint x)
    { return FixedPoint(mantissa_ + x.mantissa_); }

    double value() const
    { return mantissa_ / double(huge(T_mantissa())); }

private:
    T_mantissa mantissa_;
};

ostream& operator<<(ostream& os, const FixedPoint& a)
{
    os << a.value();
    return os;
}
```

The function `huge(T)` returns the largest representable value for type `T`; in the example above, it's equal to `UINT_MAX`.

The `FixedPoint`

6 Indirection

Indirection is the ability to modify or access an array at a set of selected index values. Blitz++ provides several forms of indirection:

- **Using a list of array positions**: this approach is useful if you need to modify an array at a set of scattered points.
- **Cartesian-product indirection**: as an example, for a two-dimensional array you might have a list `I` of rows and a list `J` of columns, and you want to modify the array at all (i,j) positions where i is in `I` and j is in `J`. This is a

6.2 Cartesian-product indirection

7 TinyVector

The `TinyVector` class provides a small, lightweight vector object whose size is known at compile

++

++

gcc -pt

5.Td [f 72 -2tc ger

++

)]TJ-151.597-++)]TJ/F50.94e95t-.3950Td7c.597-++)]1(usex.)-24(Ife]-271

9 Random Number Generators

9.1 Overview

These are the basic random number generators (RNGs):

`Uniform` Uniform reals on [0,1)

`Normal` Normal with specified mean and variance

`Exponential`
Exponential with specified mean

`DiscreteUniform`
Integers uniformly distributed over a specified range.

`Beta` Beta distribution

`Gamma` Gamma distribution

`F` F distribution

```
void foo()
{
    Normal<double> normalGen(0.5,0.25); // mean = 0.5, std dev = 0.25
    double x = normalGen.random();      // x is a normal random number
}
```

9.2 Note: Parallel random number generators

The generators which Blitz++
http://sprng.cs.fsu.edu

generator has a period of 2^{19937} 1 , passed several stringent statistical tests (including the
http://stat.fsu.edu/~geo/diehard.html

11 Frequently Asked Questions

11.1 Questions about installation

This problem can be xed by installing the gnu linker and binutils. Peter Nordlund found that by using `gnu-binutils-2.9.1`, this problem disappeared. You can read a detailed discussion at `http://oonumerics.org/blitz/support/blitz-support/archive/0029.html`.

I am using gcc under Solaris, and the ass.bler gives me an error that a sybol is too long.

This problem can also be xed by installing the gnu linker and binutils. See the above question.

DECcxx reports problems about \templates with C linkage"

```
void my_new_handler()
{
cerr << "Out of memory" << endl;
cerr.flush();
abort();
```

Blitz Keyword Index

_

J

L

X

Concept Index

<

=

>

[

`

A