Document Number: WG14 N392/X3J11 94-077

C9X Revision Proposal

Title: Extending VLA's to include variable rank arrays.
Author: Frank Farance
Author Affiliation: Farance Inc.
Postal Address: 555 Main Street, New York, NY, 10044-0150, USA
E-mail Address: frank@farance.com
Telephone Number: +1 212 486 4700
Fax Number: +1 212 759 1605
Sponsor: X3J11
Date: 1994-12-04
Proposal Category:
__ Editorial change/non-normative contribution
__ Correction
X_ New feature
__ Addition to obsolescent feature list
__ Addition to Future Directions
__ Other (please specify) _____
Area of Standard Affected:
__ Environment
X_ Language
__ Preprocessor
__ Library
__ Macro/typedef/tag name
__ Function
__ Header
Prior Art: Fortran 90, HPF, APL, C*, DPCE,
Target Audience: Numeric programming, data parallel applications.
Related Documents (if any): DPCE proposal, VLA proposal, XVLA docs.
Proposal Attached: X_ Yes __ No, but what's your interest?
Abstract:
The "shapeof()" operator is added to extract the dimensions of an object. This feature allows programs to operate on arrays without prior knowledge of their rank (number of dimensions). The "shapeis()" type qualifier may be used in declarations to specify the rank and dimenions when then rank is not known at compile time. Objects may be reshaped (i.e., interpreted as arrays of different shape) by using "shapeis()" in a type cast.

Proposal:
NOTE: This proposal is in the early stages of development. The intent of the proposal is to sketch the functionality, semantics, and issues. The proposal will continue to be developed.

The VLA extensions that have been proposed by MacDonald are an excellent start for providing varying length arrays. Basically, the MacDonald proposal provides two features: (1) allowing the programmer to explicitly pass ''shape'' information (i.e., length of each dimension) and the pointer to the array,

```
        /* The shape is: 10 20 */
        int A[10][20];

        /*
         * The following are passed:
         * The shape: 10 20.
         * The pointer to the array: A.
         */
        foo(10,20,A);
```

(2) run-time pasting of shape and pointer to create a type:

```
        int foo( int row, int col, int A[row][col])
        {
                /* ... */
        }
```

Thus, the programmer must explicitly pass the shape information along with every subroutine call. The Ritchie proposal, also known as the ''fat pointer'' proposal, has the compiler pass shape information along with the pointer to the array:

```
        /* The shape is: 10 20 */
        int A[10][20];

        /*
         * The following are passed:
         * The shape: 10 20.
         * The pointer to the array: A.
         */
        bar(&A);

        int bar(int (*A)[?][?])
        {
                int row;
                int col;

                /*
                 * Extract shape information.
                 * This could be implemented as
                 * just extracting theh shape info
                 * from the stack, i.e., no divide
                 * is required.
                 */
                col = (sizeof((*A)[0]))/(sizeof(int));
                row = (sizeof(*A))/(sizeof(int));
                row /= col;
        }
```

Both the MacDonald and Ritchie proposals are equivalent with repsect to the information that is passed from caller to called function: both shape and pointer are passed. The main difference is that the MacDonald proposal requires explicit passing of the information and the Ritchie proposal

requires implicit passing of the information. Each proposal has their own advantages. The MacDonald proposal doesn't require passing several copies of the shape information, thus may be more efficient for some interfaces:

```
int A[10][20],B[10][20],C[10][20];

add_matrix_1(10,20,A,B,C);

int add_matrix_1( int row, int col,
        int A[row][col],
        int B[row][col],
        int C[row][col])
{
        /* ... */
}
```

The Ritchie proposal has the advantage that it reduces programming errors by having the compiler pass information that the programmer would have to explicitly declare and pass otherwise (imagine explicitly passing the shape and pointer for each function call):

```
int A[10][20],B[10][20],C[10][20];

add_matrix_2(&A,&B,&C);

int add_matrix_2(
        int (*A)[?][?],
        int (*B)[?][?],
        int (*C)[?][?])
{
        /* ... */
}
```

In fact, some compilers may be able to optimize this so that only one copy of shape is passed as on the argument stack.

The problem is that the rank (the number of dimensions) must be known at compile time. For a function that performs an alternating sum reduction (adding all the even elements, subtracting all the odd elements) that operates on an array, a matrix, or an N-dimensional array, the rank of the operand varies. This type of function (common in numeric applications) cannot be declared within the Ritchie or MacDonald proposals.

The only solution is for the programmer to write the explicit pointer and indexing arithmetic -- something that the compiler *should* do. Aside from being error-prone (programmer must get indexing arithmetic right), there needs to be N copies (one at each array reference, possibly different because of different programmers) to calculate the offset to the indexed element. One could provide (and standardize) a function library, but that's the whole purpose of providing array indexing in C.

The solution is to provide an operator "shapeof()" that returns an array of "size_t", one for each dimension. For convenience, the "rankof()" operator has the semantics:

```
#define rankof(x) ((shapeof(shapeof(x)))[0])
```

For example:

```
/* assumes "size_t" is "int" */
int A[10][20][30];
int i;

printf("the rank is %d, shape is: ",rankof(A));
for ( i = 0 ; i < rankof(A) ; i++ )
{
        printf(" %d",shapeof(A)[i]);
}
printf("\n");
```

will produce:

```
the rank is 2, shape is: 10 20
```

Issue: The lifetime of the result of "shapeof()" has yet to be determined.

Issue: Scalars (non-arrays) have a rank of zero. Zero-length objects must be investigated.

The MacDonald VLA proposal allows automatic variables to be declared with varying length:

```
int A[N];
```

Again, there is no mechanism for declaring arrays with varying length. The "shapeis()" type qualifier is used to specify the dimensions of the type:

```
size_t dim_list[] = { 10, 20 };

/* equivalent to B[10][20] */
int shapeis(dim_list) B;
```

Another possibility is to ''clone'' shapes from other variables:

```
/* C has the same shape as B */
int shapeis(shapeof(B)) C;
```

Issue: The syntax and/or placement of "shapeis()" is yet to be refined.

Issue: Must the operand of "shapeis()" have the type ''array of "size_t"''?

The "shapeis()" type qualifier can be used as a type cast with varying rank:

```
int D[200];
/*
 * equivalent:
 * int (*E)[10][20] = (int (*)[10][20]) D;
 */
int shapeis(dim_list) (*E) =
        (int shapeis(dim_list) *) D;
```

The Ritchie proposal signals with "[?]" that the shape information be implicitly passed -- the compiler puts the shape information on the argument stack. This proposal uses "shapeis(?)" to signal that shape information is passed.

```
void xyz( int shapeis(?) G )
{
        size_t i;

        printf("the rank is %d, shape is: ",
                rankof(G));
        for ( i = 0 ; i < rankof(G) ; i++ )
        {
                printf(" %d",(int) shapeof(G)[i]);
        }
        printf("\n");
}

main()
{
        int F[10][20];

        /*
         * Passes both shape and pointer
         * because prototype has "shapeis(?)".
         */
        xyz(F);
}
```

This example prints:

```
the rank is 2, shape is: 10 20
```

Again, without some feature for handling arrays of varying rank, the programmer is left with the solution of coding these features on his/her own.

Summary:

- "shapeof(X)" returns an array of "size_t" that contains the ''dimensions'' of "X".
- "rankof(X)" returns the number of dimensions in "X". A return of zero indicates X is a scalar (not an array).
- "T shapeis(Y) Z" declares the array "Z" of shape

"Y" containing type "T" elements.
- "(shapeis(Y)) Z" recasts "Z" to the shape of "Y".
- "T shapeis(?) Z" declares a function parameter to have the shape of the argument passed along with the pointer to the array.

Development Plan:

- Determine lifetime of the result of "shapeof()".
- Investigate zero-length arrays.
- Specify "rankof()".
- Define the term "scalar".
- Develop declaration syntax for "shapeis()" type qualifier.
- Develop type cast semantics for "shapeis()".
- Determine the type of the argument to "shapeis()".
- Develop declaration syntax for "shapeis(?)".
- Demonstrate sample implementation.