Document Number: WG14 N391/X3J11 94-076

C9X Revision Proposal

Title: Arrays as first class objects.
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: C structures, APL.
Target Audience: Numeric programs, data parallel applications.
Related Documents (if any): XVLA documents, VLA proposals.
Proposal Attached: __ Yes X_ No, but what's your interest?
Abstract:

This extension allows arrays to be considered as first class objects. First class objects can be used in assignment and argument passing. Array-like objects (ALO's) are C arrays that are first class objects. Arrays can be declared as ALO's or C arrays. A sample declaration might look like:

```
        int A[4], B[4];  /* C array */
        alo int C[4], D[4]; /* first class object */

        f(A,B) /* passes two pointers as arguments */
        g(C,D) /* passes four integers for each argument */

        /*
         * Function that receives two pointers.  Adds
         * the array Y to X.  Returns the sum of all
         * elements.  NOTE: X is modified.
         */
        int f( int X[4], int Y[4] )
        {
                int i, sum = 0 ;
                for ( i = 0 ; i < 4 ; i++ )
```

045

```
                {
                        X[i] += Y[i];
                        sum += X[i];
                }
                return sum;
        }

        /*
         * Function that receives two arrays as values,
         * i.e., 8 integers are passed as arguments.
         * Adds the array Y to X.  Returns the sum of all
         * elements.  NOTE: X is NOT modified.
         */
        int g( alo int X[4], alo int Y[4] )
        {
                int i, sum = 0 ;
                for ( i = 0 ; i < 4 ; i++ )
                {
                        X[i] += Y[i];
                        sum += X[i];
                }
                return sum;
        }
```

The use of "alo" as a keyword is only a placeholder until more appropriate syntax is developed. The semantics of an ALO are:

```
        alo int E[5];
```

is equivalent to:

```
        struct { int array[5]; } E;
```

In other words, an ALO is equivalent to a C array in a structure. C arrays may be ``cast'' to ALO's:

```
        /* Note: The cast here produces an l-value. */
        int A[4], B[4];

        (alo)A = (alo)B;
```

ALO's may be cast to a C array by using the keyword "carray". The use of keyword "carray" is a placeholder until better syntax is developed. Casting an ALO to "carray" is equivalent to getting the pointer of the beginning of the array. At first glance, this looks equivalent to the address-of "&" operator. I prefer to use the cast to "carray" for the purpose of developing semantics rather than confusing the issue by overloading the "&" operator. Ultimately, the syntax *might* be "&", but this issue doesn't have to be resolved now.

```
        f( (carray)C, (carray)D );
```

Passing arrays as ALO's (pass by value) are useful because they provide a faster private copy of the array on the stack (rather than the function allocating memory for a temporary array).

ALO's are simple to implement because they are simply rewritten as an array inside a structure. Structure copying and argument passing is already part of Standard C.

Development Plan:

- Determine how this affects the type system.
- Determine how this affects type casts.
- Develop appropriate syntax.