Document Number: WG14 N393/X3J11 94-078

C9X Revision Proposal

Title: Adding support for distributed 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: Intel 80x86 compilers, C*, DPCE, HPF.
Target Audience: C programmers, distributed/MPP systems, array slicing.
Related Documents (if any): DPCE draft, HPF.
Proposal Attached: __ Yes X_ No, but what's your interest?
Abstract:
Some virtual memory systems provide discontiguous access to data, e.g., segmented memory, distributed memory. In some cases, there many be several layers in the memory hierarchy (e.g., network node, processor number, process number, local address). Pointers that access the data may be local (i.e., relative to some context, e.g., segment number, process number, etc.) or global (i.e., the context is specified). The local pointer can access contiguous data local to its context, i.e., pointers can walk local data, but not cross context boundaries. Global pointers can reference data regardless of context. Global pointers are of two varieties: local incrementing (cannot cross context boundaries) and global incrementing (can cross context boundaries automatically). In 80x86 systems, these pointers are called "near", "far", and "huge". Memory hierarchies that have more that two layers have increasing spheres of locality (e.g., local to process, local to processor, local to node, global). This proposal provides a general mechanism, including layout operators, layout type qualifiers, and a pointer hierarchy to access data on these systems.

The "layoutof()" operator returns an array of "size_t" where each element is the size of the ''extent''. For example, if A is an array of 10 integers, distributed across two processors (or segements) with 6 elements on the first processor and 4 elements on the second processor, assuming "int"s are 4 bytes:

```
layoutof(A) is { 24, 16 }
```

The "pointerof()" operator (placeholder name until syntax can be agreed) produces a list of pointers to each extent:

```
pointerof(A) is { 1000, 2000}
```

Given these two sets of values, distributed objects can now be processed as C objects since address and size are known. This is useful for writing distributed versions of "memcpy".

The type qualifier "layoutis()" is used to define layout. For example:

```
int layoutis(layoutof(A)) B[10];
```

declares B with layout similar to A. The "layoutis()" type qualifier may be used in function prototypes. The "layoutis(?)" type qualifier is used when the programmer wants the layout information passed with the argument:

```
void my_memcpy(void layoutis(?) *dst,
        void layoutis(?) *src);
```

The local pointers are still known as "void *". The global pointers with local incrementing are:

```
/* xxx is some layout specification */
void layoutis(xxx) *P;
```

The global pointers with global incrementating are:

```
/* xxx is some layout specification */
void (volatile layoutis(xxx)) *P;
```

Development Plan:

- Define result of "layoutof()".
- Determine changes to type system ("layoutis()").
- Create sample implementation.