Dimensions, Bounds, and Allocation

The shape of the array is usually specified in the declaration. For such static shape arrays, storage can be managed in the usual way:

static allocation for arrays whose lifetime is the entire program;

stack allocation for arrays whose lifetime is an invocation of a subroutine;

heap allocation for dynamically allocated arrays with more general lifetime.

Storage management is more complex for arrays whose shape is not known until elboration time, or whose shape may change during execution. For these the compiler must arrange not only to allocated space, but also to make shape information available at run time (without such information indexing would not be possible). Some dynamically typed languages allow run-time binding of both the number and bounds of dimensions. Compiled languages may allow the bounds to be dynamic, but typically require the number of dimensions to be static. A local array whose shape is known at elaboration time may still be allocated in the stack. An array whose size may change during execution must generally be allocated in the heap.

Dope Vectors

During compilation, the **symbol table** maintains dimension and bounds information for every array in the program. For every record, it maintains the offset of every field. When the number and bounds of an array dimensions are statically known, the compiler can look them up in the symbol table in order to compute the address of elements of the array. When these values are not statically known, the compiler must generate code to look them up in a dope vector at run time.

In the general case a dope vector must specify the lower bound of each dimension and the size each dimension other than the last (which is always the size of the element type, and will thus be statically known). If the language implementation performs dynamic checks for out-of-bounds subscripts in array references, then the dope vector may contain upper bounds as well.

The contents of the dope vector are initialized at elaboration time, or whenever the number or bounds of dimensions change. In a language that provides both a value model of variables and arrays of dynamic shape, we must consider the possibility that a record will contain a field whose size is not statically knwon. In this case the compiler may use dope vectors not only for dynamic shape arrays, but also for dynamic shape records. The dope vector for a record typically indicated the offset of each field from the beginning of the record.

Stack Allocation

conformant arrays: are dynamic arrays that are allowed as subroutine parameters, with shape fixed at subroutine call time.

Ada and C (though not C++) support dynamic shape for both parameters and local variables. Among other things, local arrays can be declared to match the shape of conformant array parameters, facilitating the implementaion of algorithms that require temporary space for calculations.

In many languages, including Ada and C, the shape of a local array becomes fixed at elaboration time. For such arrays it is still possible to place the space for the array in the stack frame of its subroutine, but an extra level of indirection is required. In order to ensure that ever local object can be found using a known offset from the frame pointer, we divide the stack frame into a **fixed-size part** and a **variable-size part**. An object whose size is statically known goes in the fixed part. An object whose size is not known until elaboration time goes in the variable-size part, and a pointer to it, together with a dope vector, goes in the fixed-size part. If the elaboration of the array is buried in a nested block, the compiler delays allocating space until the block is entered. It still allocates space for the pointer and the dope vector among the local variables when the subroutine itself is entered. Records of dynamic shape are handled in a similar way.

Heap Allocation

Arrays that can change shape at arbitrary times are sometimes said to be **fully dynmaic**. Because changes in size do not in general occur in FIFO order, stack allocation will not suffice; fully dynamic arrays must be allocated on the heap.

Several languages including all the major scripting languages, allow strings— arrays of characters— to change size after elaboration time. Dynamically resizeable arrays are also supported byt the `vector`, `Vector`, and `Arraylist` classes of the C++, Java, and C# libraries respectively. In many cases, increasing the size will require that the run-time system allocate a larger block, copy any data that are to be retained from the old block to the new one, and then deallocate the old.

If the number of dimensions of a fully dynamic array is statically known, the dope vector can be kept, together with a pointer to the data, in the stack frame of the subroutine in which the array was declared. If the number of dimensions can change, the dope vector must generally be placed at the beginning of the heap block instead.

In the absence of garbage collection, the compiler must arrange to reclaim the space occupied by fully dynmaic arrays when control returns from the subroutine in which they were declared. Space for stack-allocated arrays is of course reclaimed automatically by popping the stack.