Variable-length array

From Wikipedia, the free encyclopedia

In computer programming, a **variable-length array** (or **VLA**) is an array data structure of automatic storage duration whose length is determined at run time (instead of at compile time).[1]

Programming languages that support VLAs include Ada, Algol 68 (for non-flexible rows), APL, C99 (although subsequently relegated in C11 to a conditional feature which implementations are not required to support;[2][3][4] on some platforms, could be implemented previously with `alloca()` or similar functions) and C# (as unsafe-mode stack-allocated arrays), COBOL, Fortran 90, J.

Memory [edit]

Allocation [edit]

One problem that may be hidden by a language's support for VLAs is that of the underlying memory allocation: in environments where there is a clear distinction between a heap and a stack, it may not be clear which, if any, of those will store the VLA.[5]

For example, the GNU C Compiler allocates memory for VLAs on the stack.[6] VLAs, like all objects in C, are limited to SIZE_MAX bytes.[7]

Variable access [edit]

In some programming languages VLAs can be accessed via pointers, but the size can no longer be obtained when de-referenced as they are considered complete types.[8]

Examples [edit]

The following C99 function allocates a variable-length array of a specified size, fills it with floating-point values, then passes it to another function for processing. Because the array is declared as an automatic variable, its lifetime ends when the `read_and_process` function returns.

```
float read_and_process(int n)
{
    float vals[n];

    for (int i = 0; i < n; i++)
        vals[i] = read_val();
    return process(vals, n);
}
```

Following is the same example in Ada. Note that Ada arrays carry their bounds with them, there is no need to pass the length to the Process function.

```
type Vals_Type is array (Positive range <>) of Float;

function Read_And_Process (N : Integer) return Float is
   Vals : Vals_Type (1 .. N);
begin
   for I in 1 .. N loop
```

Wikipedia The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
한국어
中文
Edit links

```
      Vals (I) := Read_Val;
   end loop;
   return Process (Vals);
end Read_And_Process;
```

The equivalent Fortran 90 function is:

```
function read_and_process(n) result(o)
    integer,intent(in)::n
    real::o

    real,dimension(n)::vals
    integer::i

    do i = 1,n
       vals(i) = read_val()
    end do
    o = process(vals)
end function read_and_process
```

when utilizing the Fortran 90 feature of checking procedure interfaces at compile-time; on the other hand, if the functions use pre-Fortran 90 call interface the (external) functions must first be declared, and the array length must be explicitly passed as an argument (as in C):

```
function read_and_process(n) result(o)
    integer,intent(in)::n
    real::o

    real,dimension(n)::vals
    real::read_val, process
    integer::i

    do i = 1,n
       vals(i) = read_val()
    end do
    o = process(vals,n)
end function read_and_process
```

The following COBOL fragment declares a variable-length array of records, `DEPT-PERSON`, having a length (number of members) specified by the value of `PEOPLE-CNT`.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DEPT-PEOPLE.
    05  PEOPLE-CNT          PIC S9(4) BINARY.
    05  DEPT-PERSON         OCCURS 0 TO 20 TIMES DEPENDING ON PEOPLE-CNT.
        10  PERSON-NAME     PIC X(20).
        10  PERSON-WAGE     PIC S9(7)V99 PACKED-DECIMAL.
```

The following C# fragment declares a variable-length array of integers. The "unsafe" keyword would require an assembly containing this code to be marked as unsafe.

```
unsafe void declareStackBasedArray(int size)
{
    int *pArray = stackalloc int[size];
    pArray[0] = 123;
}
```

Dynamic vs. automatic [edit]

Languages such as Java technically do not provide variable-length arrays, because all array objects in those languages are dynamically allocated on the heap, and therefore do not have automatic storage duration for arrays.

References [edit]

1. ^ http://docs.cray.com/books/004-2179-001/html-004-2179-001/z893434830malz.html
2. ^ http://pic.dhe.ibm.com/infocenter/ratdevz/v8r0/topic/com.ibm.xlcpp111.aix.doc/language_ref/variable_length_arrays.html
3. ^ http://gcc.gnu.org/onlinedocs/gcc/Variable-Length.html
4. ^ ISO 9899:2011 Programming Languages - C 6.7.6.2 4
5. ^ https://archive.stsci.edu/fits/users_guide/node65.html
6. ^ http://gcc.gnu.org/onlinedocs/gfortran/Code-Gen-Options.html
7. ^ §6.5.3.4 and §7.20.3 of the C11 standard (n1570.pdf)
8. ^ http://msdn.microsoft.com/en-us/library/4s7x1k91.aspx

Categories: Arrays