NAME

varargs – handle variable argument list

SYNOPSIS

#include <varargs.h>

va_alist

va_dcl

va_list pvar;

void va_start(va_list pvar);

type **va_arg(va_list pvar,** *type***);**

void va_end(va_list pvar);

DESCRIPTION

This set of macros allows portable procedures that accept variable argument lists to be written. Routines that have variable argument lists (such as **printf**(3S)) but do not use **varargs** are inherently non-portable, as different machines use different argument-passing conventions.

va_alist is used as the parameter list in a function header.

va_dcl is a declaration for **va_alist**. No semicolon should follow **va_dcl**.

va_list is a type defined for the variable used to traverse the list.

va_start is called to initialize **pvar** to the beginning of the list.

va_arg will return the next argument in the list pointed to by **pvar**. *type* is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type of argument is expected, as it cannot be determined at runtime.

va_end is used to clean up.

Multiple traversals, each bracketed by **va_start** and **va_end**, are possible.

EXAMPLE

This example is a possible implementation of **execl** (see **exec**(2)).

```
#include <unistd.h>
#include <varargs.h>
#define MAXARGS     100

/*      execl is called by
                execl(file, arg1, arg2, ..., (char *)0);
*/
execl(va_alist)
va_dcl
{
        va_list ap;
        char *file;
        char *args[MAXARGS];                /* assumed big enough*/
        int argno = 0;

        va_start(ap);
```

```
                file = va_arg(ap, char *);
                while ((args[argno++] = va_arg(ap, char *)) != 0)
                        ;
                va_end(ap);
                return execv(file, args);
        }
```

SEE ALSO

exec(2), **printf**(3S), **vprintf**(3S), **stdarg**(5)

NOTES

It is up to the calling routine to specify in some manner how many arguments there are, since it is not always possible to determine the number of arguments from the stack frame. For example, **execl** is passed a zero pointer to signal the end of the list. **printf** can tell how many arguments are there by the format.

It is non-portable to specify a second argument of **char**, **short**, or **float** to **va_arg**, since arguments seen by the called function are not **char**, **short**, or **float**. C converts **char** and **short** arguments to **int** and converts **float** arguments to **double** before passing them to a function.

stdarg is the preferred interface.