NAME

stdarg − variable argument list

SYNOPSIS

#include <stdarg.h>

void va_start(va_list *ap***,** *argtypeN parmN***)**
type **va_arg(va_list** *ap***,** *type***)**
void va_end(va_list *ap***)**

DESCRIPTION

This set of macros provides a means of writing portable procedures that accept variable argument lists. Routines having variable argument lists (such as **printf**(3)) that do not use **stdarg** are inherently non-portable, since different machines use different argument passing conventions.

A function that accepts a variable argument list is declared with "..." at the end of its parameter list. It must have at least one normal argument before the "...". For example:

```
int printf(const char *format, ...) { /* code */ }
int fprintf(FILE *stream, const char *format, ...) { /* code */ }
```

va_list is a type which is used for the variable *ap* within the body of a variable argument function which is used to traverse the list.

va_start(*ap*, *parmN*) is called to initialize *ap* to the beginning of the list. The last true parameter of the function, *parmN*, must be supplied to allow **va_start** to compute the address of the first variable parameter.

va_arg(*ap*, *type*) will return the next argument in the list pointed to by *ap*. *Type* is the type to which the expected argument will be converted when passed as an argument.

Different types can be mixed, but it is up to the routine to know what type of argument is expected, since it cannot be determined at runtime.

va_end(*ap*) must be used to finish up.

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

EXAMPLE

```
#include <stdarg.h>

execl(const char *path, ...)
{
	va_list ap;
	char *args[100];
	int argno = 0;

	va_start(ap, path);
	while ((args[argno++] = va_arg(ap, char *)) != NULL) {}
	va_end(ap);
	return execv(path, args);
}
```

NOTES

It is up to the calling routine to determine how many arguments there are, since it is not possible to determine this from the stack frame. For example, **execl** passes a null pointer to signal the end of the list. **Printf** can tell how many arguments are supposed to be there by the format.

The macros **va_start** and **va_end** may be arbitrarily complex; for example, **va_start** might contain an opening brace, which is closed by a matching brace in **va_end**. Thus, they should only be used where they could be placed within a single complex statement.

BUGS

It is impossible to properly show the macros as C declarations as is done in the synopsis. They can never be coded as C functions, because all three macros use their arguments by address, and the *type* field is certainly impossible. Just look at them as being part of the C language, like **sizeof**.