

<stdarg.h>

Include the standard header <stdarg.h> to access the unnamed additional arguments in a function that accepts a varying number of arguments.

To access the additional arguments, the program must first execute the macro `va_start` within the body of the function to initialize a data object with context information. Subsequent execution of the macro `va_arg`, designating the same context information, yields the values of the additional arguments in order, beginning with the first unnamed argument. You can execute the macro `va_arg` from any function that can access the context information saved by the macro `va_start`. If you have executed the macro `va_start` in a function, you must execute the macro `va_end` in the same function, designating the same context information, before the function returns. You can repeat this sequence (as needed) to access the arguments as often as you want.

You declare a data object of type `va_list` to store context information. `va_list` can be an array type, which affects how the program shares context information with functions that it calls. (The address of the first element of an array is passed, rather than the contents of the data object itself.)

For example, to concatenate an arbitrary number of strings onto the end of an existing string (assuming that the existing string is stored in a data object large enough to hold the resulting string):

```
#include <stdarg.h>
void va_cat(char *s, ...)
{
    char *t;
    va_list ap;

    va_start(ap, s);
    while (t = va_arg(ap, char *)) NULL terminates list
    {
        s += strlen(s); skip to end
        strcpy(s, t); and copy a string
    }
    va_end(ap);
}
```

`va_arg`

```
#define va_arg(va_list ap, T) <rvalue of type T>
```

The macro yields the value of the next argument in order, specified by the context information designated by `ap`. The additional argument must

Standard C

be of data object type *T* after applying the rules for promoting arguments in the absence of a function prototype.

va_end

```
#define va_end(va_list ap) <void expression>
```

The macro performs any cleanup necessary so that the function can return.

va_list

```
typedef do-type va_list;
```

The type is the data object type *do-type* that you declare to hold the context information initialized by `va_start` and used by `va_arg` to access additional unnamed arguments.

va_start

```
#define va_start(va_list ap, last-arg) <void expression>
```

The macro stores initial context information in the data object designated by `ap`. *last-arg* is the name of the last argument you declare. For example, *last-arg* is `b` for the function declared as `int f(int a, int b, ...)`. The last argument must not have `register` storage class, and it must have a type that is not changed by the translator. (It cannot have an array type, a function type, type *float*, or any integer type that changes when promoted.)