va_arg, va_copy, va_end, va_start

11/04/2016

Accesses variable-argument lists.

Syntax

```
type va_arg(
   va_list arg_ptr,
   type
);
void va_copy(
   va_list dest,
   va_list src
); // (ISO C99 and later)
void va_end(
   va_list arg_ptr
);
void va_start(
   va_list arg_ptr,
   prev_param
); // (ANSI C89 and later)
void va_start(
   arg_ptr
);  // (deprecated Pre-ANSI C89 standardization version)
```

Parameters

type
Type of argument to be retrieved.

arg_ptr
Pointer to the list of arguments.

dest
Pointer to the list of arguments to be initialized from ***src***

src
Pointer to the initialized list of arguments to copy to ***dest***.

prev_param
Parameter that precedes the first optional argument.

Return Value

va_arg returns the current argument. **va_copy**, **va_start** and **va_end** do not return values.

Remarks

The **va_arg**, **va_copy**, **va_end**, and **va_start** macros provide a portable way to access the arguments to a function when the function takes a variable number of arguments. There are two versions of the macros: The macros defined in **STDARG.H** conform to the ISO C99 standard; the macros defined in **VARARGS.H** are deprecated but are retained for backward compatibility with code that was written before the ANSI C89 standard.

These macros assume that the function takes a fixed number of required arguments, followed by a variable number of optional arguments. The required arguments are declared as ordinary parameters to the function and can be accessed through the parameter names. The optional arguments are accessed through the macros in **STDARG.H** (or **VARARGS.H** for code that was written before the ANSI C89 standard), which sets a pointer to the first optional argument in the argument list, retrieves arguments from the list, and resets the pointer when argument processing is completed.

The C standard macros, defined in **STDARG.H**, are used as follows:

- **va_start** sets ***arg_ptr*** to the first optional argument in the list of arguments that's passed to the function. The argument ***arg_ptr*** must have the **va_list** type. The argument ***prev_param*** is the name of the required parameter that immediately precedes the first optional argument in the argument list. If ***prev_param*** is declared with the register storage class, the macro's behavior is undefined. **va_start** must be used before **va_arg** is used for the first time.
- **va_arg** retrieves a value of ***type*** from the location that's given by ***arg_ptr***, and increments ***arg_ptr*** to point to the next argument in the list by using the

size of ***type*** to determine where the next argument starts. **va_arg** can be used any number of times in the function to retrieve arguments from the list.

- **va_copy** makes a copy of a list of arguments in its current state. The ***src*** parameter must already be initialized with **va_start**; it may have been updated with **va_arg** calls, but must not have been reset with **va_end**. The next argument that's retrieved by **va_arg** from ***dest*** is the same as the next argument that's retrieved from ***src***.
- After all arguments have been retrieved, **va_end** resets the pointer to **NULL**. **va_end** must be called on each argument list that's initialized with **va_start** or **va_copy** before the function returns.

Note

The macros in VARARGS.H are deprecated and are retained only for backwards compatibility with code that was written before the ANSI C89 standard. In all other cases, use the macros in STDARGS.H.

When they are compiled by using **/clr** (Common Language Runtime Compilation), programs that use these macros may generate unexpected results because of differences between native and common language runtime (CLR) type systems. Consider this program:

```
#include <stdio.h>
#include <stdarg.h>

void testit (int i, ...)
{
    va_list argptr;
    va_start(argptr, i);

    if (i == 0)
    {
        int n = va_arg(argptr, int);
        printf("%d\n", n);
    }
    else
    {
        char *s = va_arg(argptr, char*);
        printf("%s\n", s);
    }

    va_end(argptr);
}

int main()
{
    testit(0, 0xFFFFFFFF); // 1st problem: 0xffffffff is
not an int
    testit(1, NULL);       // 2nd problem: NULL is not a
char*
}
```

Notice that **testit** expects its second parameter to be either an **int** or a **char***. The arguments being passed are 0xffffffff (an **unsigned int**, not an **int**) and **NULL** (actually an **int**, not a **char***). When the program is compiled for native code, it produces this output:

```
-1

(null)
```

Requirements

Header:<stdio.h> and **<stdarg.h>**

Deprecated Header:<varargs.h>

Libraries

All versions of the C run-time libraries.

Example

```
// crt_va.c
// Compile with: cl /W3 /Tc crt_va.c
// The program below illustrates passing a variable
// number of arguments using the following macros:
```

```
//      va_start            va_arg              va_copy
//      va_end              va_list

#include <stdio.h>
#include <stdarg.h>
#include <math.h>

double deviation(int first, ...);

int main( void )
{
    /* Call with 3 integers (-1 is used as terminator). */
    printf("Deviation is: %f\n", deviation(2, 3, 4, -1 ));

    /* Call with 4 integers. */
    printf("Deviation is: %f\n", deviation(5, 7, 9, 11,
-1));

    /* Call with just -1 terminator. */
    printf("Deviation is: %f\n", deviation(-1));
}

/* Returns the standard deviation of a variable list of
integers. */
double deviation(int first, ...)
{
    int count = 0, i = first;
    double mean = 0.0, sum = 0.0;
    va_list marker;
    va_list copy;

    va_start(marker, first);     /* Initialize variable
arguments. */
    va_copy(copy, marker);       /* Copy list for the
second pass */
    while (i != -1)
    {
        sum += i;
        count++;
        i = va_arg(marker, int);
    }
    va_end(marker);              /* Reset variable argument
list. */
    mean = sum ? (sum / count) : 0.0;
    i = first;                  /* reset to calculate
deviation */
    sum = 0.0;
    while (i != -1)
    {
        sum += (i - mean)*(i - mean);
        i = va_arg(copy, int);
    }
    va_end(copy);               /* Reset copy of argument
list. */
    return count ? sqrt(sum / count) : 0.0;
}
```

```
Deviation is: 0.816497
Deviation is: 2.236068
Deviation is: 0.000000
```

See also

Argument Access
vfprintf, **_vfprintf_l**, **vfwprintf**, **_vfwprintf_l**