va_arg

Defined in header <cstdarg>

```
T va_arg( va_list ap, T );
```

The va_arg macro expands to an expression of type T that corresponds to the next parameter from the va_list ap.

Prior to calling va_arg, ap must be initialized by a call to either va_start or va_copy, with no intervening call to va_end. Each invocation of the va_arg macro modifies ap to point to the next variable argument.

If va_arg is called when there are no more arguments in ap, or if the type of the next argument in ap (after promotions) is not compatible with T, the behavior is undefined, unless:

- one type is a signed integer type, the other type is the corresponding unsigned integer type, and the value is representable in both types; or
- one type is pointer to `void` and the other is a pointer to a character type (`char`, `signed char`, or `unsigned char`).

Parameters

ap - an instance of the va_list type
T - the type of the next parameter in ap

Expanded value

the next variable parameter in ap

Example

Run this code

```
#include <iostream>
#include <cstdarg>
#include <cmath>

double stddev(int count, ...)
{
    double sum = 0;
    double sum_sq = 0;
    va_list args;
    va_start(args, count);
    for (int i = 0; i < count; ++i) {
        double num = va_arg(args, double);
        sum += num;
        sum_sq += num*num;
    }
    va_end(args);
    return std::sqrt(sum_sq/count - (sum/count)*(sum/count));
}

int main()
{
    std::cout << stddev(4, 25.0, 27.3, 26.9, 25.7) << '\n';
}
```

Output:

```
0.920258
```

See also

va_start

Defined in header <cstdarg>

```
void va_start( va_list ap, parm_n );
```

The va_start macro enables access to the variable arguments following the named argument parm_n.

va_start should be invoked with an instance to a valid va_list object ap before any calls to va_arg.

If parm_n is declared with reference type or with a type not compatible with the type that results from default argument promotions, the behavior is undefined.

Parameters

ap	-	an instance of the va_list type
parm_n	-	the named parameter preceding the first variable parameter

Expanded value

(none)

Notes

va_start is required to support parm_n with overloaded operator&.

Example

Run this code

```
#include <iostream>
#include <cstdarg>

int add_nums(int count, ...)
{
    int result = 0;
    va_list args;
    va_start(args, count);
    for (int i = 0; i < count; ++i) {
        result += va_arg(args, int);
    }
    va_end(args);
    return result;
}

int main()
{
    std::cout << add_nums(4, 25, 25, 50, 50) << '\n';
}
```

Output:

```
150
```

See also

va_arg	accesses the next variadic function argument (function macro)
va_end	ends traversal of the variadic function arguments (function macro)
C documentation for **va_start**	

va_copy

Defined in header <cstdarg>

```
void va_copy( va_list dest, va_list src );    (since C++11)
```

The va_copy macro copies `src` to `dest`.

va_end should be called on `dest` before the function returns or any subsequent re-initialization of `dest` (via calls to `va_start` or **`va_copy`**).

Parameters

`dest` - an instance of the `va_list` type to initialize

`src` - the source `va_list` that will be used to initialize `dest`

Expanded value

(none)

Example

Run this code

```
#include <iostream>
#include <cstdarg>
#include <cmath>

double sample_stddev(int count, ...)
{
    double sum = 0;
    va_list args1;
    va_start(args1, count);
    va_list args2;
    va_copy(args2, args1);
    for (int i = 0; i < count; ++i) {
        double num = va_arg(args1, double);
        sum += num;
    }
    va_end(args1);
    double mean = sum / count;

    double sum_sq_diff = 0;
    for (int i = 0; i < count; ++i) {
        double num = va_arg(args2, double);
        sum_sq_diff += (num-mean) * (num-mean);
    }
    va_end(args2);
    return std::sqrt(sum_sq_diff / count);
}

int main()
{
    std::cout << sample_stddev(4, 25.0, 27.3, 26.9, 25.7) << '\n';
}
```

Output:

```
0.920258
```

See also

`va_start`	enables access to variadic function arguments (function macro)