std::memmove

Defined in header <cstring>

```
void* memmove( void* dest, const void* src, std::size_t count );
```

Copies count characters from the object pointed to by src to the object pointed to by dest. Both objects are reinterpreted as arrays of `unsigned char`.

The objects may overlap: copying takes place as if the characters were copied to a temporary character array and then the characters were copied from the array to dest.

If either dest or src is an invalid or null pointer, the behavior is undefined, even if count is zero.

If the objects are potentially-overlapping or not *TriviallyCopyable*, the behavior of memmove is not specified and may be undefined (http://stackoverflow.com/questions/29777492) .

Parameters

dest	–	pointer to the memory location to copy to
src	–	pointer to the memory location to copy from
count	–	number of bytes to copy

Return value

dest

Notes

std::memmove may be used to implicitly create objects in the destination buffer.

Despite being specified "as if" a temporary buffer is used, actual implementations of this function do not incur the overhead of double copying or extra memory. For small count, it may load up and write out registers; for larger blocks, a common approach (glibc and bsd libc) is to copy bytes forwards from the beginning of the buffer if the destination starts before the source, and backwards from the end otherwise, with a fall back to std::memcpy when there is no overlap at all.

Where strict aliasing prohibits examining the same memory as values of two different types, std::memmove may be used to convert the values.

Example

Run this code

```
#include <iostream>
#include <cstring>

int main()
{
    char str[] = "1234567890";
    std::cout << str << '\n';
    std::memmove(str + 4, str + 3, 3); // copies from [4, 5, 6] to [5, 6, 7]
    std::cout << str << '\n';
}
```

Output:

```
1234567890
1234456890
```

See also

memcpy	copies one buffer to another (function)
memset	fills a buffer with a character

	(function)
wmemmove	copies a certain amount of wide characters between two, possibly overlapping, arrays (function)
copy **copy_if** (C++11)	copies a range of elements to a new location (function template)
copy_backward	copies a range of elements in backwards order (function template)
is_trivially_copyable (C++11)	checks if a type is trivially copyable (class template)
C documentation for **memmove**	