

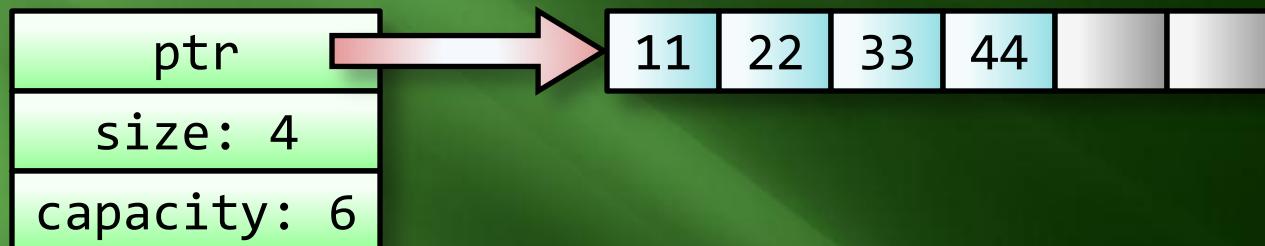
Data Structure Visualizers in Visual Studio 2010

Or: Why "DO NOT MODIFY"
Really Means "HAVE FUN"

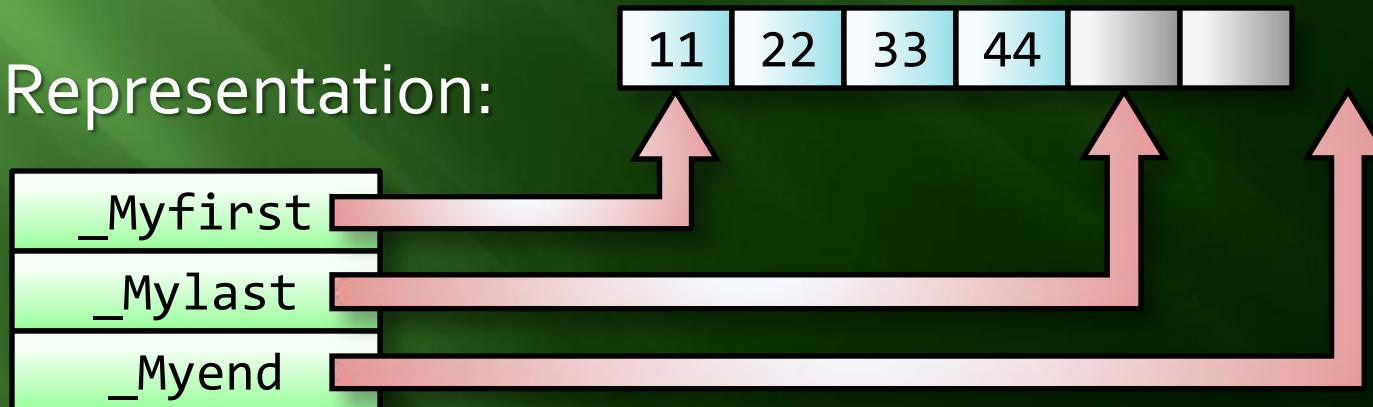
Stephan T. Lavavej
("Steh-fin Lah-wah-wade")
Visual C++ Libraries Developer
stl@microsoft.com

vector: More Than Meets The Eye

- Appearance:



- Representation:



Data Structure Representations Are Ugly

The screenshot shows the Microsoft Visual Studio interface during debugging. The title bar reads "meow (Debugging) - Microsoft Visual Studio". The menu bar includes File, Edit, View, Project, Build, Debug, Team, Data, Tools, Architecture, Test, Analyze, Window, and Help. The toolbar has icons for Stop, Start, Break, and Step. The code editor window is titled "meow.cpp" and contains the following C++ code:

```
#include <vector>
using namespace std;

int main() {
    vector<int> v;

    v.push_back(11);
    v.push_back(22);
    v.push_back(33);
    v.push_back(44);

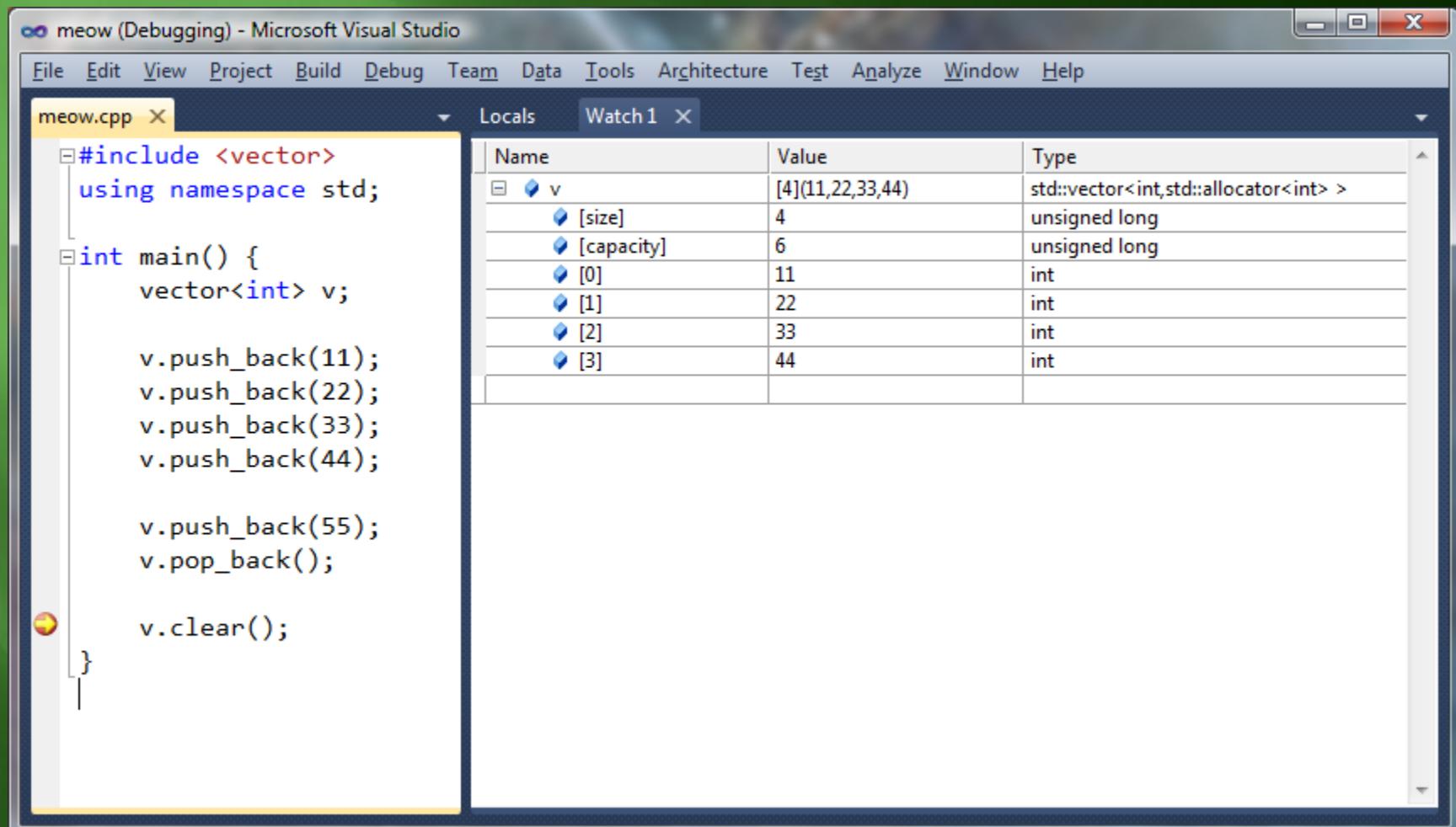
    v.push_back(55);
    v.pop_back();

    v.clear();
}
```

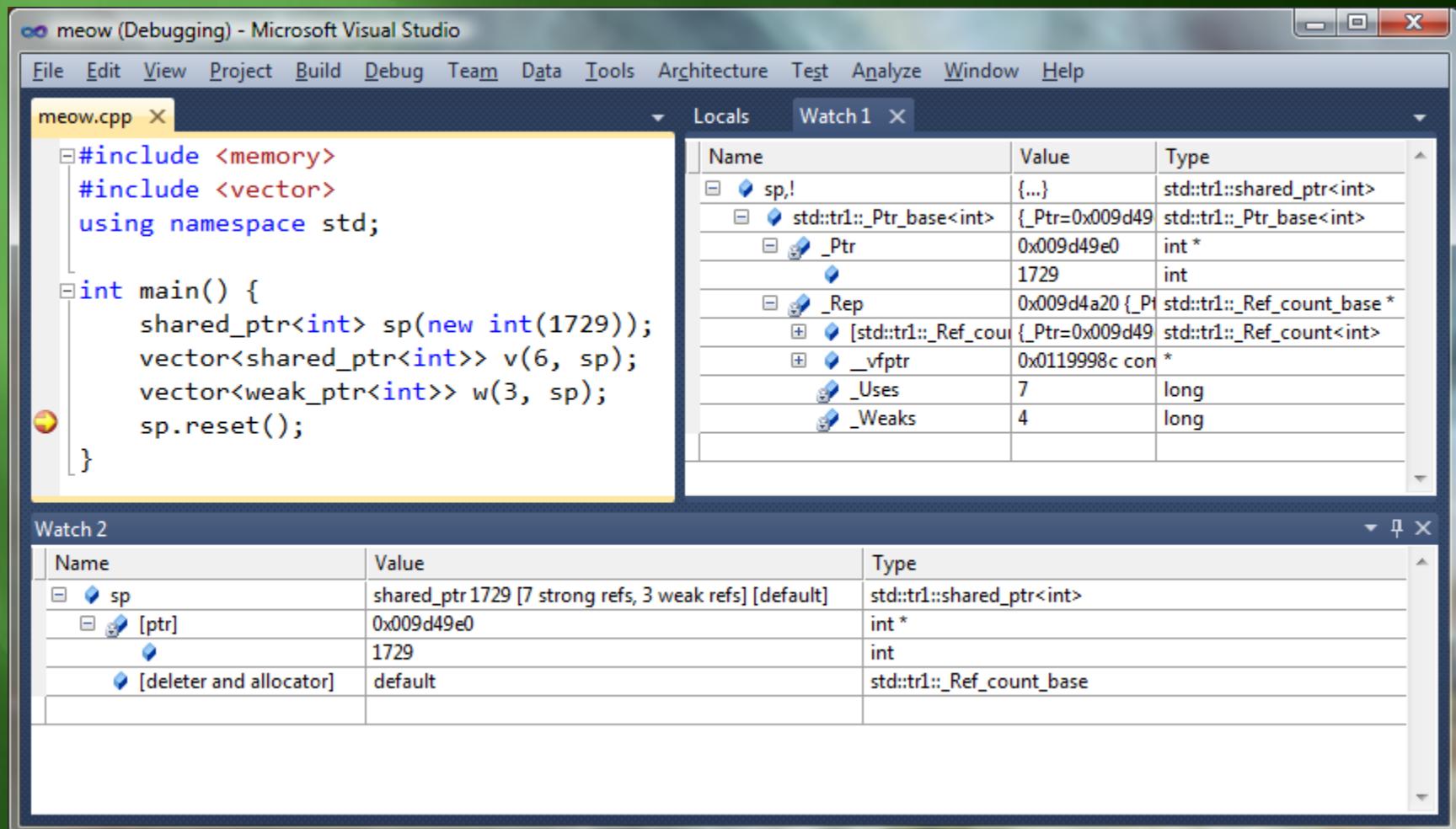
The "Locals" window is open, showing the state of variables in memory. The table lists the following variables and their values:

Name	Value	Type
v,! (expandable)	{...}	std::vector<int, std::allocator<int>>
std::Vector_val<int, std::allocator<int>> _Myfirst (expandable)	{_Myproxy=0x00b149e0}	std::Vector_val<int, std::allocator<int>>
std::Container_base12 _Myproxy (expandable)	0x00b14a78	std::Container_base12
_Myfirst (expandable)	11	int *
_Mylast (expandable)	55	int *
_Myend (expandable)	-33686019	int *
_Alval (expandable)	{...}	std::allocator<int>
v._Mylast - v._Myfirst	4	unsigned long
v._Myend - v._Myfirst	6	unsigned long
v._Myfirst[0]	11	int
v._Myfirst[1]	22	int
v._Myfirst[2]	33	int
v._Myfirst[3]	44	int

Data Structure Visualizers Are Pretty



Ugly Versus Pretty



Getting Started

- ❑ Visualizers are controlled by:
C:\Program Files (x86)\Microsoft Visual Studio 10.0\
Common7\Packages\Debugger\autoexp.dat
- ❑ Undocumented and unsupported
 - You'll find bugs, especially when trying new things
- ❑ Traditional Method: Hack this file
 - ; DO NOT MODIFY
 - Preserve the original file before hacking it!
 - autoexp.dat can be edited with VS itself
 - Bypass UAC: Right Click VS > Run As Administrator
- ❑ Super Secret Method: Provide another file
 - Put your file's location in the environment variable _vcee_autoexp
 - Begin your file with [Visualizer]
- ❑ Start Debugging (F5) picks up changes
- ❑ var,! in the Watch Window disables visualization

Previews And Children

Name	Value	Preview	Type
v	[4](11,22,33,44)		std::vector<int, std::allocator<int> >
[size]	4		unsigned long
[capacity]	6		unsigned long
[0]	11		int
[1]	22		int
[2]	33		int
[3]	44		int

Complex Is Simple (Using preview And children)

The screenshot shows the Microsoft Visual Studio interface during debugging. The top bar displays "meow (Debugging) - Microsoft Visual Studio (Administrator)". The menu bar includes File, Edit, View, Project, Build, Debug, Team, Data, Tools, Architecture, Test, Analyze, Window, and Help. The toolbar has standard icons for file operations.

The main window shows the source code file "meow.cpp". The code defines a namespace "math" containing a struct "gaussian_integer" with members "a" and "b". It also contains a main function that creates an instance of "gaussian_integer" with values 3 and 4, and then sets the value of "a" to 0.

The "Locals" window shows the current variable state:

Name	Value	Type
g.!	{a=3 b=4 }	math::gaussian_integer
a	3	int
b	4	int
g	3 + 4i	math::gaussian_integer
a [real]	3	int
b [imag]	4	int

The "autoexp.dat" window shows the custom visualization rules:

```
math::gaussian_integer {
    preview ( #($e.a, " + ", $e.b, "i") )
    children ( #( $(a [real]: $e.a), $(b [imag]: $e.b) ) )
}
```

Complex Is Complex

The screenshot shows a Microsoft Visual Studio interface with the title bar "meow (Debugging) - Microsoft Visual Studio (Administrator)". The menu bar includes File, Edit, View, Project, Build, Debug, Team, Data, Tools, Architecture, Test, Analyze, Window, and Help. The solution explorer shows "autoexp.dat" and "meow.cpp". The code editor displays the following C++ code:

```
namespace math { ... }

int main() {
    math::gaussian_integer r = { 0, 0 };
    math::gaussian_integer s = { 1, 0 };
    math::gaussian_integer t = { 0, 2 };
    math::gaussian_integer u = { 3, 4 };
    math::gaussian_integer v = { -5, 0 };
    math::gaussian_integer w = { 0, -6 };
    math::gaussian_integer x = { 7, -8 };
    math::gaussian_integer y = { -9, 10 };
    math::gaussian_integer z = { -11, -12 };

    z.a = 0;
}
```

The Locals window shows the following variable values:

Name	Value	Type
r	0 + 0i	math::gaussian_integer
s	1 + 0i	math::gaussian_integer
t	0 + 2i	math::gaussian_integer
u	3 + 4i	math::gaussian_integer
v	-5 + 0i	math::gaussian_integer
w	0 + -6i	math::gaussian_integer
x	7 + -8i	math::gaussian_integer
y	-9 + 10i	math::gaussian_integer
z	-11 + -12i	math::gaussian_integer

Complex Is Simple Again (Using #if, #elif, And #else)

The screenshot shows a Microsoft Visual Studio interface with the title bar "meow (Debugging) - Microsoft Visual Studio (Administrator)". The menu bar includes File, Edit, View, Project, Build, Debug, Team, Data, Tools, Architecture, Test, Analyze, Window, and Help. The toolbar has icons for Save, Undo, Redo, Cut, Copy, Paste, Find, Replace, Select All, and Stop.

The main window displays two tabs: "autoexp.dat" and "meow.cpp". The "meow.cpp" tab is active, showing the following C++ code:

```
math::gaussian_integer {
    preview (
        #if ($e.b == 0) ( ; REAL
            $e.a
        ) #elif ($e.a == 0) ( ; IMAG
            #($e.b, "i")
        ) #elif ($e.b > 0) ( ; REAL + IMAG
            #($e.a, " + ", $e.b, "i")
        ) #else ( ; REAL - IMAG
            #($e.a, " - ", -$e.b, "i")
        )
    )

    children ( #
        #(a [real]: $e.a),
        #(b [imag]: $e.b)
    )
}
```

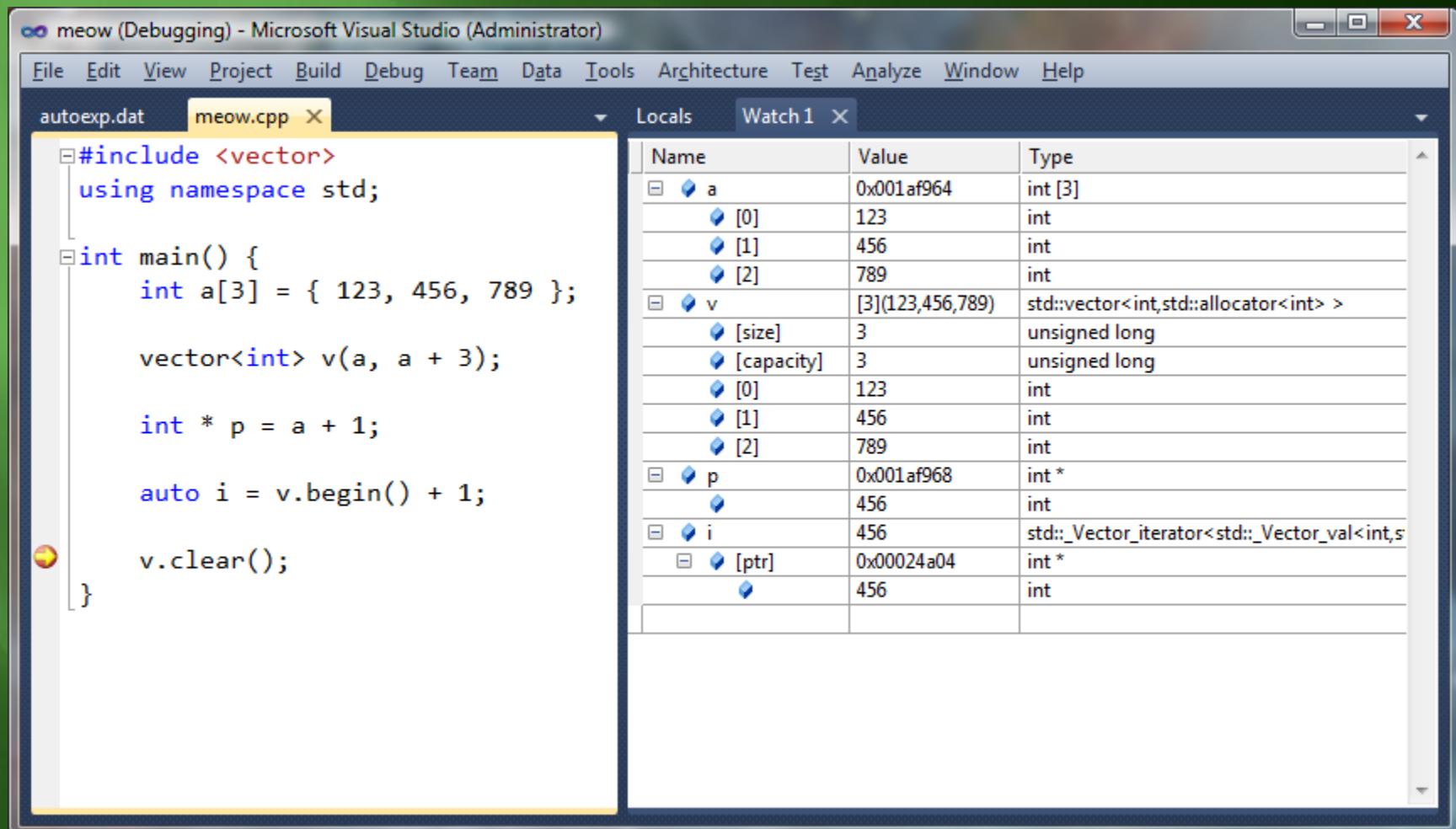
To the right of the code editor is a "Locals" window showing variable values:

Name	Value	Type
r	0	math::gaussian_integer
s	1	math::gaussian_integer
t	2i	math::gaussian_integer
u	3 + 4i	math::gaussian_integer
v	-5	math::gaussian_integer
w	-6i	math::gaussian_integer
x	7 - 8i	math::gaussian_integer
y	-9 + 10i	math::gaussian_integer
z	-11 - 12i	math::gaussian_integer

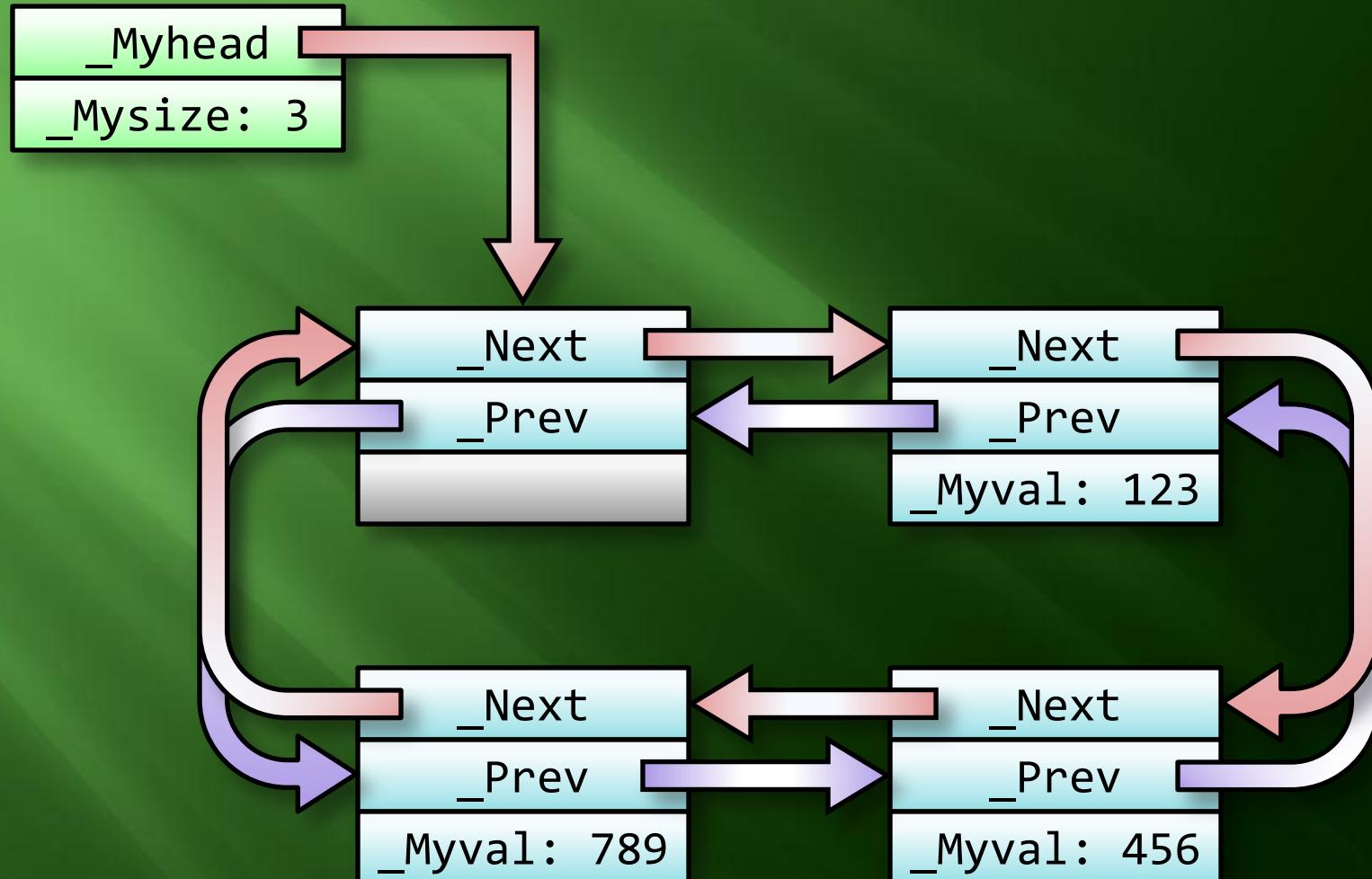
vector's Visualizer (Using #array)

```
std::vector<*>{
    preview ( #(
        "[" , $e._Mylast - $e._Myfirst, "](" ,
        #array(expr: $e._Myfirst[$i],
               size: $e._Mylast - $e._Myfirst), ")"
    ) )
    children ( #(
        #([size] : $e._Mylast - $e._Myfirst),
        #([capacity] : $e._Myend - $e._Myfirst),
        #array(expr: $e._Myfirst[$i],
               size: $e._Mylast - $e._Myfirst)
    ) )
}
std::_Vector_iterator<*>|std::_Vector_const_iterator<*>{
    preview ( *$e._Ptr )
    children ( #([ptr] : $e._Ptr) )
}
```

vector Visualized



list's Representation



list's Visualizer (Using #list)

```
std::list<*>{
    preview ( #(
        "[" , $e._Mysize, "](" ,
        #list(head: $e._Myhead->_Next,
              size: $e._Mysize,
              next: _Next) : $e._Myval, ")"
    ) )
    children (
        #list(head: $e._Myhead->_Next,
              size: $e._Mysize,
              next: _Next) : $e._Myval
    )
}
std::_List_iterator<*>|std::_List_const_iterator<*>{
    preview ( $e._Ptr->_Myval )
    children ( #([ptr] : &$e._Ptr->_Myval) )
}
```

list Visualized

The screenshot shows the Microsoft Visual Studio interface during a debugging session. The title bar reads "meow (Debugging) - Microsoft Visual Studio (Administrator)". The menu bar includes File, Edit, View, Project, Build, Debug, Team, Data, Tools, Architecture, Test, Analyze, Window, and Help. The toolbar has buttons for Stop, Start, Break, and Step. The left pane displays the code in "meow.cpp":

```
#include <list>
using namespace std;

int main() {
    list<int> lst;
    lst.push_back(123);
    lst.push_back(456);
    lst.push_back(789);

    auto i = lst.begin();
    ++i;

    lst.clear();
}
```

The "Locals" window is open, showing the state of variables at the current program point. The table lists the following variables and their values:

Name	Value	Type
lst	[3](123,456,789)	std::list<int, std::allocator<int>>
[0]	123	int
[1]	456	int
[2]	789	int
i	456	std::_List_iterator<std::_List_val<int, std::allocator<int>>::_Container_base12
[ptr]	0x00a64a98	int *
	456	int
lst._Myhead	{...}	std::list<int, std::allocator<int>>::_Container_base12
std::_List_val<int, std::allocator<int>>::_Container_base12	{...}	std::_List_val<int, std::allocator<int>>::_Container_base12
_Myproxy	0x00a64a00	std::_Container_base12
_Myhead	0x00a64a00	std::_List_nod<int, std::allocator<int>>::_Container_base12
_Next	0x00a64a48	std::_List_nod<int, std::allocator<int>>::_Container_base12
_Next	0x00a64a90	std::_List_nod<int, std::allocator<int>>::_Container_base12
_Prev	0x00a64a00	std::_List_nod<int, std::allocator<int>>::_Container_base12
_Myval	123	int
_Prev	0x00a64ad8	std::_List_nod<int, std::allocator<int>>::_Container_base12
_Myval	-842150451	int
_Mysize	3	unsigned int
_Alnod	{...}	std::allocator<std::_List_nod<int, std::allocator<int>>::_Container_base12>
_Alval	{...}	std::allocator<int>

map's Visualizer (Using #tree And A New "Feature")

```
std::map<*>|std::multimap<*>|std::set<*>|std::multiset<*>{
    preview ( #( "[", $e._Mysize, "](",
        #tree(head: $e._Myhead->_Parent, skip: $e._Myhead,
              left: _Left, right: _Right, size: $e._Mysize
        ) : $e._Myval, ")"
    ) )
    children ( #(
        #([comp] : $e.comp),
        #tree(head: $e._Myhead->_Parent, skip: $e._Myhead,
              left: _Left, right: _Right, size: $e._Mysize
        ) : $e._Myval
    ) )
}
std::_Tree_iterator<*>|std::_Tree_const_iterator<*>{
    preview ( $e._Ptr->_Myval )
    children ( #([ptr] : &$e._Ptr->_Myval) )
}
```

map Visualized

The screenshot shows the Microsoft Visual Studio interface during debugging. The title bar reads "meow (Debugging) - Microsoft Visual Studio (Administrator)". The menu bar includes File, Edit, View, Project, Build, Debug, Team, Data, Tools, Architecture, Test, Analyze, Window, and Help. The toolbar has icons for Stop, Start, Break, and Step. The solution explorer shows "autoexp.dat" and "meow.cpp". The code editor displays the following C++ code:

```
#include <functional>
#include <map>
#include <string>
using namespace std;

int main() {
    map<int, string> ml;
    ml[3] = "DS9";
    ml[2] = "TNG";
    ml[1] = "TOS";
    ml[4] = "VOY";

    map<int, string, greater<int>>
        mg(ml.begin(), ml.end());

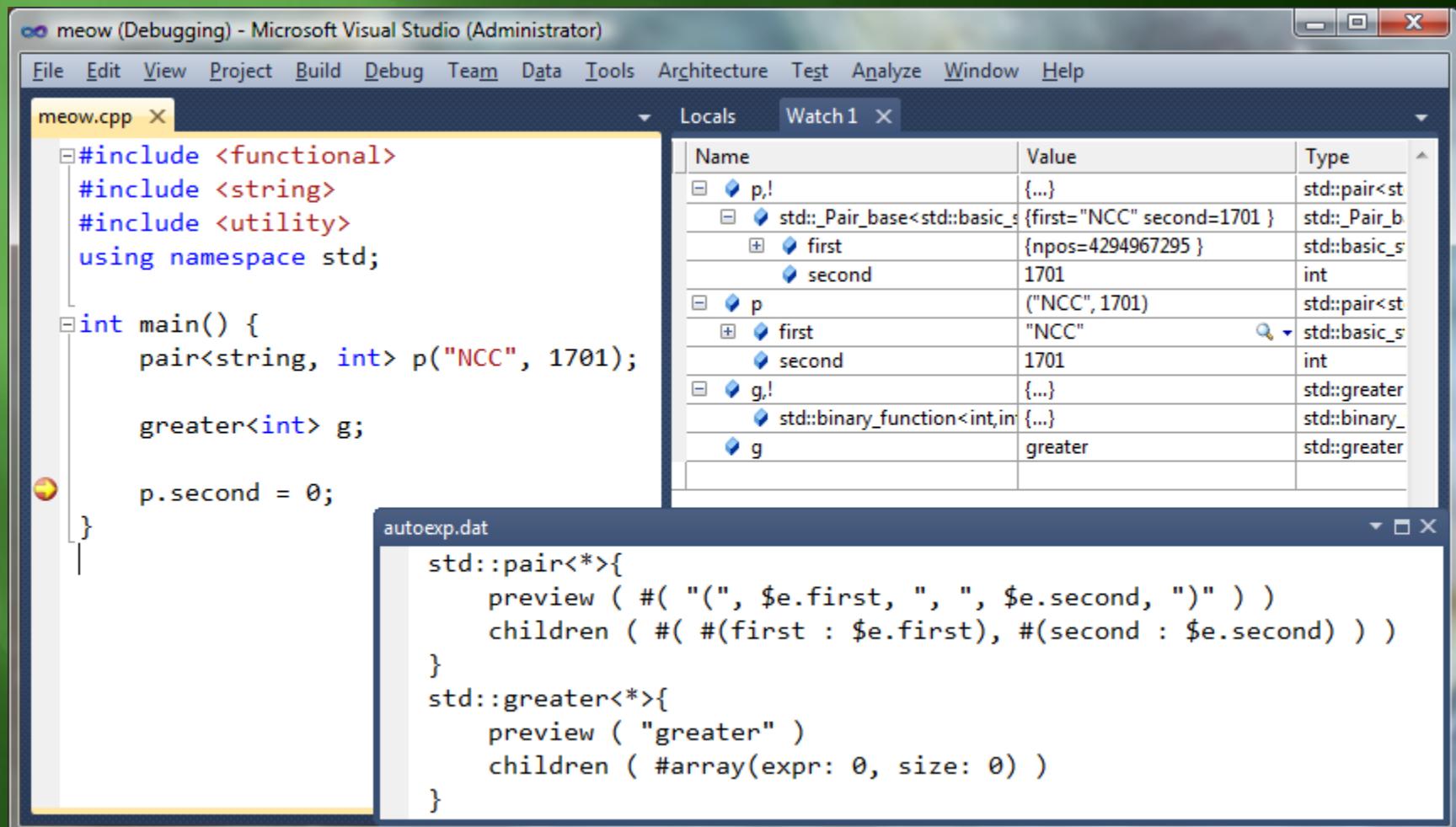
    ml.clear();
}
```

The Locals window shows the state of variables:

Name	Type
ml	std::map<int, std::string>
[comp]	std::less<int>
[0]	std::pair<int const, std::string>
[1]	std::pair<int const, std::string>
[2]	std::pair<int const, std::string>
[3]	std::pair<int const, std::string>
mg	std::map<int, std::string>
[comp]	std::greater<int>
[0]	std::pair<int const, std::string>
[1]	std::pair<int const, std::string>
[2]	std::pair<int const, std::string>
[3]	std::pair<int const, std::string>

The Watch1 window is currently empty.

pair and greater's Visualizers (Hiding Base Classes)



string's Visualizer (Using stringview)

```
std::basic_string<char,*>{
    preview    ( #if (($e._Myres) < ($e._BUF_SIZE)) (
        [$e._Bx._Buf,s] ) #else ( [$e._Bx._Ptr,s] ) )
    stringview ( #if (($e._Myres) < ($e._BUF_SIZE)) (
        [$e._Bx._Buf,sb] ) #else ( [$e._Bx._Ptr,sb] ) )
    children   ( #([size] : $e._Mysize),
                  #([capacity] : $e._Myres),
        #if (($e._Myres) < ($e._BUF_SIZE)) (
            #array(expr: $e._Bx._Buf[$i], size: $e._Mysize)
        ) #else (
            #array(expr: $e._Bx._Ptr[$i], size: $e._Mysize) ) ) )
std::_String_iterator<char,*>|std::_String_const_iterator<char,*>{
    preview    ( [$e._Ptr,s] )
    stringview ( [$e._Ptr,sb] )
    children   ( #([ptr] : $e._Ptr) ) }
```

string Visualized

The screenshot shows a debugger interface with two windows. On the left is the 'Text Visualizer' window, which displays the expression 'small' and its value 'meOw'. On the right is the 'Locals' window, which shows the state of variables in memory.

Text Visualizer:

- Expression: small
- Value: meOw
- Wrap Close Help

Locals Window:

Name	Value	Type
small	"meOw"	std::basic_string<char, std::char_traits<char>, std::allocator<char>>
[size]	4	unsigned int
[capacity]	15	unsigned int
[0]	109 'm'	char
[1]	101 'e'	char
[2]	79 'O'	char
[3]	119 'w'	char
i	"Ow"	std::String_iterator<char>
[ptr]	0x0029f836 "Ow"	const char *
	79 'O'	const char
large	"cute fluffy Kittens"	std::basic_string<char, std::char_traits<char>, std::allocator<char>>
j	"Kittens"	std::String_iterator<char>
small._Bx	{_Buf=0x0029f834 "meOw" _Ptr=0x774f656d <Bad Ptr>}	std::String_val<char, std::char_traits<char>, std::allocator<char>>
_Buf	0x0029f834 "meOw"	char [16]
_Ptr	0x774f656d <Bad Ptr>	char *
_Alias	0x0029f834 "meOw"	char [16]
large._Bx	{_Buf=0x0029f7f8 "pj\0" _Ptr=0x000f4a70}	std::String_val<char, std::char_traits<char>, std::allocator<char>>
_Buf	0x0029f7f8 "pj\0"	char [16]
_Ptr	0x000f4a70 "cute fluffy Kittens"	char *
_Alias	0x0029f7f8 "pj\0"	char [16]

reverse_iterator's Visualizers (Matching Specializations)

```
std::reverse_iterator<std::_Vector_iterator*> >
|std::reverse_iterator<std::_Vector_const_iterator*> >{
    preview ( #("reverse_iterator to ", $e.current._Ptr[-1]) )
    children ( #( #[to] : $e.current._Ptr - 1),
                #(current : $e.current)
    ) ) }
std::reverse_iterator<std::_List_iterator*> >
|std::reverse_iterator<std::_List_const_iterator*> >{
    preview ( #("reverse_iterator to ",
                $e.current._Ptr->_Prev->_Myval) )
    children ( #( #[to] : &$e.current._Ptr->_Prev->_Myval),
                #(current : $e.current)
    ) ) }
std::reverse_iterator<*>{
    preview ( #("reverse_iterator current ", $e.current) )
    children ( #(current : $e.current) )
}
```

reverse_iterator Visualized

The screenshot shows the Microsoft Visual Studio interface during a debug session. The title bar reads "meow (Debugging) - Microsoft Visual Studio (Administrator)". The menu bar includes File, Edit, View, Project, Build, Debug, Team, Data, Tools, Architecture, Test, Analyze, Window, and Help. The toolbar has icons for Stop, Start, Break, and Step. The solution explorer shows "autoexp.dat" and "meow.cpp". The code editor window displays the following C++ code:

```
#include <iterator>
#include <list>
#include <vector>
using namespace std;

int main() {
    int a[] = { 12, 34, 56, 78 };
    vector<int> v(a, a + 4);
    list<int> l(a, a + 4);

    auto vri = v.rbegin() + 1;
    auto lri = next(l.rbegin(), 2);
    reverse_iterator<int *> ari(a + 1);

    int nv = *vri;
    int nl = *lri;
    int na = *ari;

    v.clear();
}
```

The Locals window shows variable values:

Name	Value	Type
nv	56	int
vri	reverse_iterator to 56	std::reverse_iterator
[to]	0x00094a08	int *
	56	int
current	78	std::Vector_iterator
[ptr]	0x00094a0c	int *
	78	int
nl	34	int
lri	reverse_iterator to 34	std::reverse_iterator
[to]	0x00094b30	int *
	34	int
current	56	std::List_iterator
[ptr]	0x00094b78	int *
	56	int
na	12	int
ari	reverse_iterator current 0x0022fc44	std::reverse_iterator
current	0x0022fc44	int *
	34	int
ari.current[-1]	12	int

The Watch1 window is empty.

array's Definition (Dealing With Template Parameters And Macros)

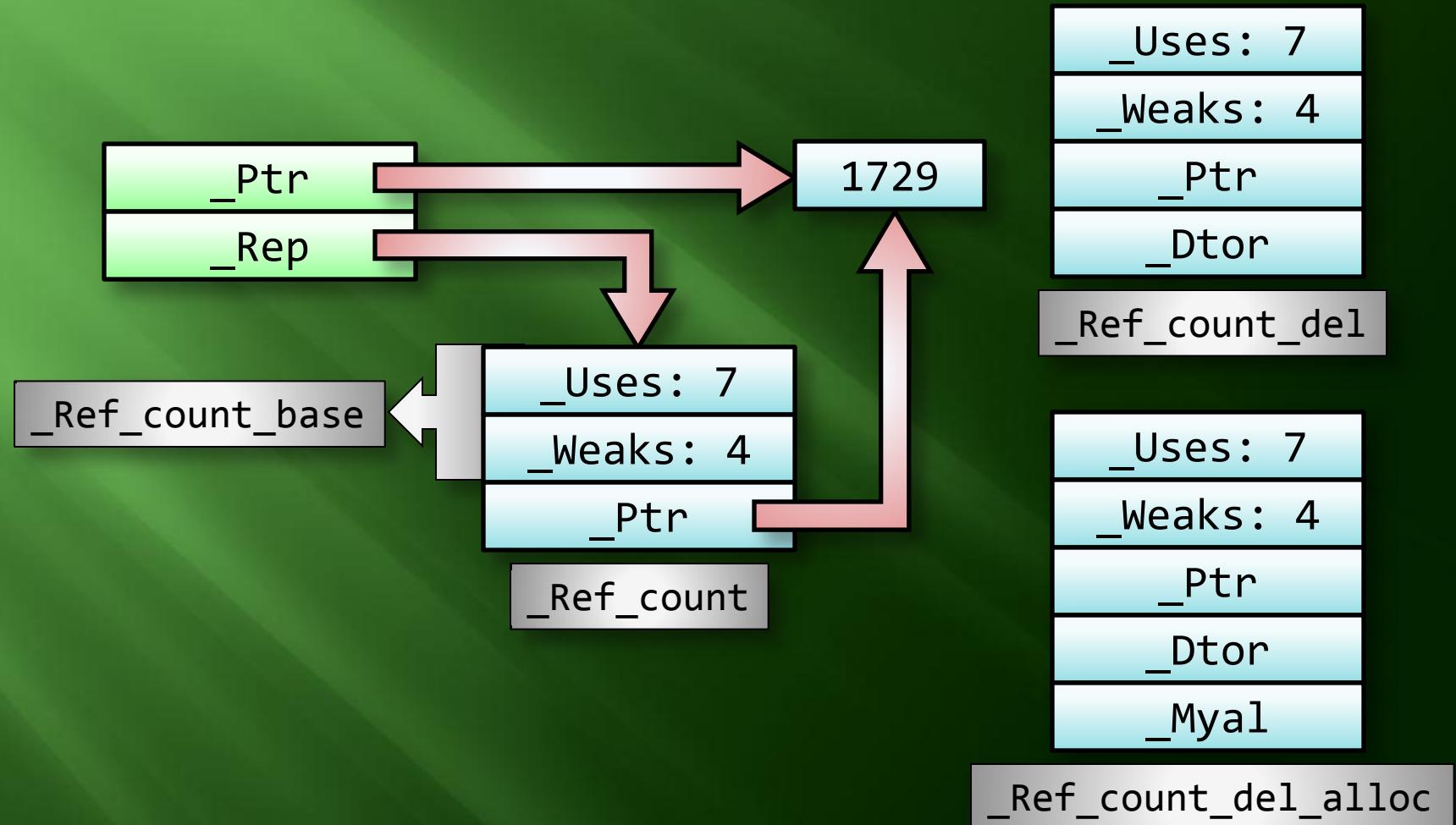
```
template <class _Ty, size_t _Size> class array {
    enum { _EEN_SIZE = _Size };
    _Ty _Elems[_Size];
};

template <class _Ty, size_t _Size>
class _Array_const_iterator {
    enum { _EEN_SIZE = _Size };
    enum { _EEN_IDL = _ITERATOR_DEBUG_LEVEL };
#if _ITERATOR_DEBUG_LEVEL == 0
    pointer _Ptr;
#else
    pointer _Ptr;
    size_t _Idx;
#endif
};
```

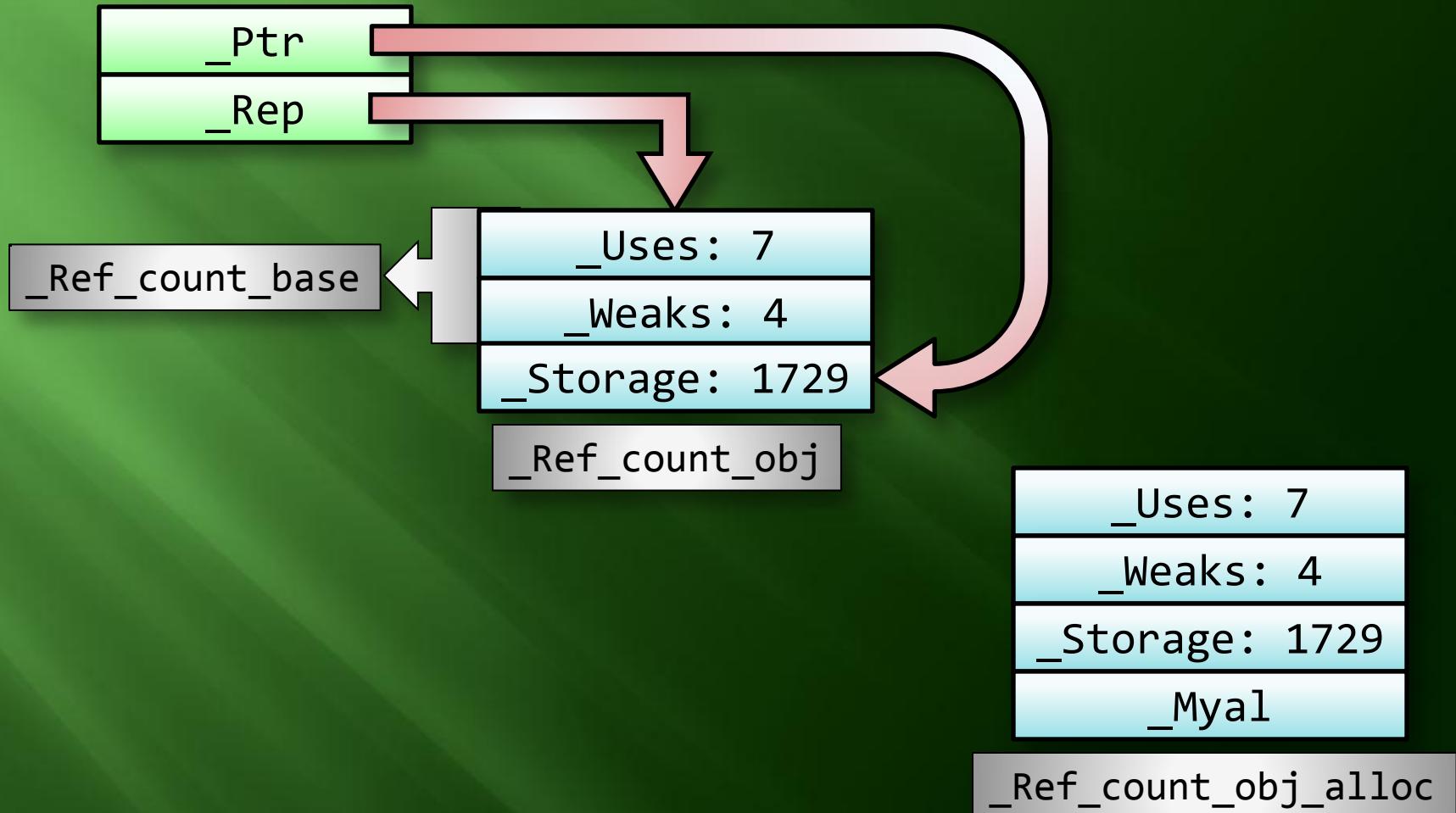
array's Visualizer

```
std::tr1::array<*>{
    preview ( #([", $e._EEN_SIZE, "])(
        #array(expr: $e._Elems[$i], size: $e._EEN_SIZE), ")" ) )
    children ( #array(expr: $e._Elems[$i], size: $e._EEN_SIZE) )
}
std::_Array_iterator<*>|std::_Array_const_iterator<*>{
    preview (
        #if ($e._EEN_IDL == 0) ( *$e._Ptr )
        #elif ($e._Idx == $e._EEN_SIZE) ( "end" )
        #else ( $e._Ptr[$e._Idx] ) )
    children (
        #if ($e._EEN_IDL == 0) ( #([ptr] : $e._Ptr) )
        #elif ($e._Idx == $e._EEN_SIZE) (#array(expr: 0, size: 0))
        #else ( #([ptr] : $e._Ptr + $e._Idx) ) )
    }
}
```

shared_ptr: Refcounts In Disguise



`make_shared<T>()` And `allocate_shared<T>()`



The Pointer-To-Base Brick Wall

meow (Debugging) - Microsoft Visual Studio

File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help

meow.cpp X

```
#include <memory>
using namespace std;

int main() {
    shared_ptr<int>
        meow(new int(
            19937));

    auto purr =
        make_shared<int>(
            65537);

    meow.reset();
}
```

Locals

Name	Value	Type
meow!	{...}	std::tr1::shared_ptr<int>
std::tr1::_Ptr_base<int>	{_Ptr=0x008a1b10}	std::tr1::_Ptr_base<int>
_Ptr	0x008a1b10	
	19937	
_Rep	0x008a33a0 {_Ptr=0x008a1b10}	std::tr1::_Ref_count_base*
[std::tr1::_Ref_count<int>]	{_Ptr=0x008a1b10}	std::tr1::_Ref_count<int>
__vfptr	0x000877b8 const std::tr1::_Object*	
__Uses	1	long
__Weak	1	long
purr!	{...}	std::tr1::shared_ptr<int>
std::tr1::_Ptr_base<int>	{_Ptr=0x008a33fc}	std::tr1::_Ptr_base<int>
_Ptr	0x008a33fc	
	65537	
_Rep	0x008a33f0 {_Storage=0x00087788}	std::tr1::_Ref_count_base*
[std::tr1::_Ref_count_obj<int>]	{_Storage={...}}	std::tr1::_Ref_count_obj<int>
__vfptr	0x00087788 const std::tr1::_Object*	
__Uses	1	long
__Weak	1	long

Watch 1

Brick Wall

Brick Wall

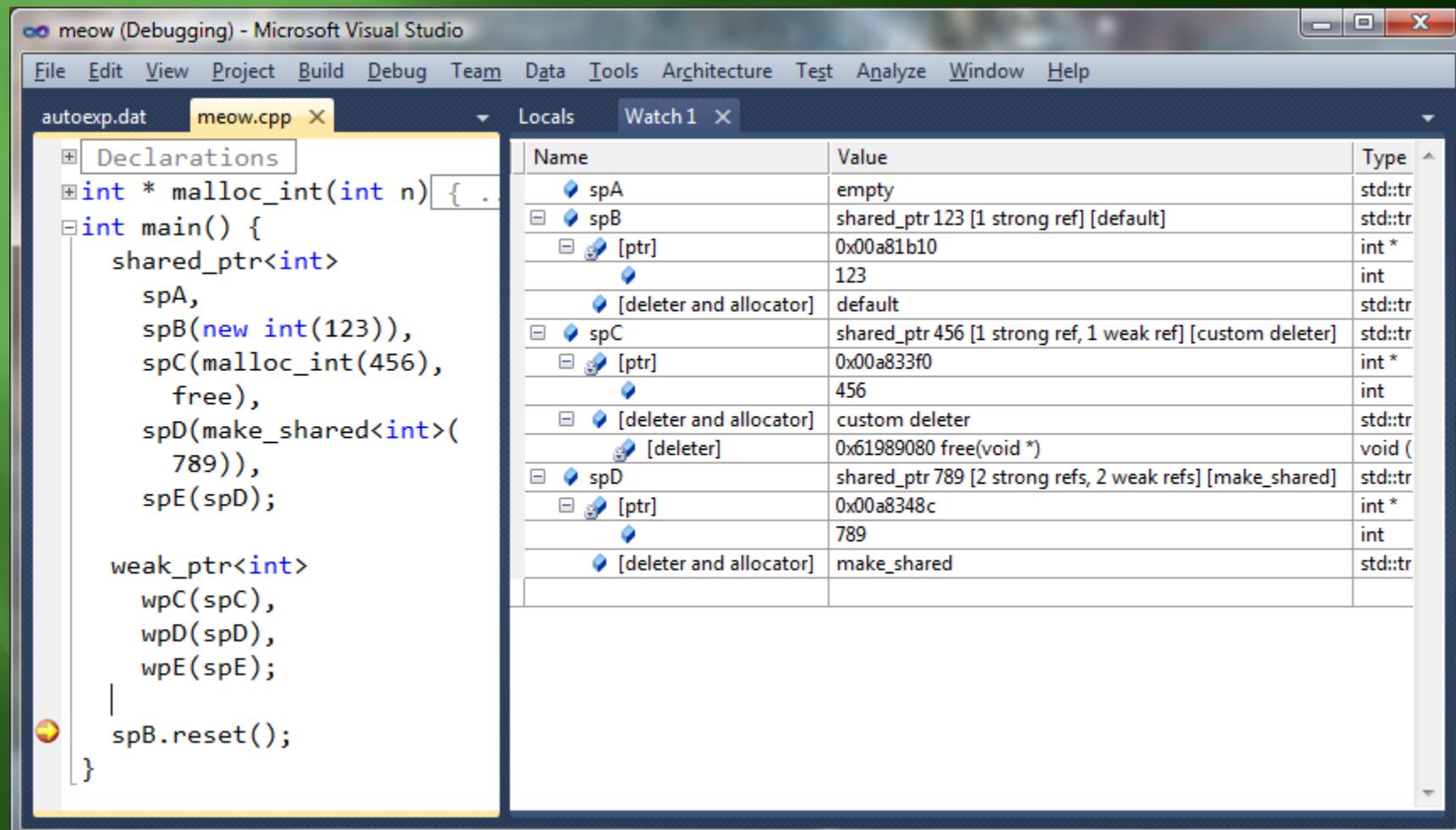
New Feature: Pointer-To-Base X-Ray Vision

```
std::tr1::_Ref_count<*>{
    preview ( "default" )
    children ( #array(expr: 0, size: 0) ) }
std::tr1::_Ref_count_del<*>{
    preview ( "custom deleter" )
    children ( #([deleter] : $e._Dtor) ) }
std::tr1::_Ref_count_del_alloc<*>{
    preview ( "custom deleter, custom allocator" )
    children ( #( #([deleter] : $e._Dtor),
                 #([allocator] : $e._Myal) ) ) }
std::tr1::_Ref_count_obj<*>{
    preview ( "make_shared" )
    children ( #array(expr: 0, size: 0) ) }
std::tr1::_Ref_count_obj_alloc<*>{
    preview ( "allocate_shared" )
    children ( #([allocator] : $e._Myal) ) }
```

shared_ptr's Visualizer

```
std::tr1::shared_ptr<*>{
    preview ( #if ($e._Ptr == 0) ( "empty" ) #else ( #(
        "shared_ptr ", *$e._Ptr, " [", $e._Rep->_Uses,
        #if ($e._Rep->_Uses == 1) (" strong ref")
                    #else (" strong refs"),
        #if ($e._Rep->_Weak - 1 > 0) ( #(
            ", ", $e._Rep->_Weak - 1,
            #if ($e._Rep->_Weak - 1 == 1) (" weak ref")
                #else (" weak refs")
            ) ),
            "] [", *$e._Rep, "]"
        ) ) )
    children ( #if ($e._Ptr == 0) ( #array(expr: 0, size: 0) )
        #else ( #( #[ptr] : $e._Ptr),
            #[[deleter and allocator] : *$e._Rep)
        ) ) )
}
```

shared_ptr Visualized



function: Another Brick Wall

meow (Debugging) - Microsoft Visual Studio

File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help

autoexp.dat meow.cpp

Locals Watch 1

Name	Value	Type
f1,! std::tr1::Function_impl!{_Space={...}_Impl=}	{...}	std::tr1::function<bool __cdecl(int)>
std::tr1::Function_impl!{_Space={...}_Impl=}	{...}	std::tr1::function<bool __cdecl(int)>
std::unary_function<i>{...}	{...}	std::unary_function<i>
_Space {Pfn=0x003bfbcc_P}	{_Pfn=0x003bfbcc_P}	std::tr1::Space
_Impl 0x003bfbcc {_Callee=}	0x003bfbcc {_Callee=}	std::tr1::Impl
[std::tr1::Impl_no] {_Callee={...}}	{_Callee={...}}	std::tr1::Impl_no
std::tr1::Impl_b {...}	{...}	std::tr1::Impl_b
_Callee {...}	{...}	std::tr1::Callable_fun<bool __cdecl(int)>
std::tr1::Callable_base<bool __cdecl(int)> 0x01351195	0x01351195	std::tr1::Callable_base<bool __cdecl(int)>
_Object 0x01351195 even(int)	0x01351195 even(int)	std::tr1::Object
_vptr 0x01359784 const std::tr1::Object	0x01359784 const std::tr1::Object	std::tr1::Object
f2,! std::tr1::Function_impl!{_Space={...}_Impl=}	{...}	std::tr1::function<bool __cdecl(int)>
std::tr1::Function_impl!{_Space={...}_Impl=}	{...}	std::tr1::function<bool __cdecl(int)>
std::unary_function<i>{...}	{...}	std::unary_function<i>
_Space {Pfn=0x003bfbac_P}	{_Pfn=0x003bfbac_P}	std::tr1::Space
_Impl 0x003bfbac {_Callee=}	0x003bfbac {_Callee=}	std::tr1::Impl
[std::tr1::Impl_no] {_Callee={...}}	{_Callee={...}}	std::tr1::Impl_no
std::tr1::Impl_b {...}	{...}	std::tr1::Impl_b
_Callee {...}	{...}	std::tr1::Callable_obj<Prime,0>
std::tr1::Callable_base<Prime,0> 0x013597cc	0x013597cc	std::tr1::Callable_base<Prime,0>
_Object {...}	{...}	std::tr1::Object
_vptr 0x013597cc const std::tr1::Object	0x013597cc const std::tr1::Object	std::tr1::Object

Brick Wall

Stored Functor

Brick Wall

Stored Functor

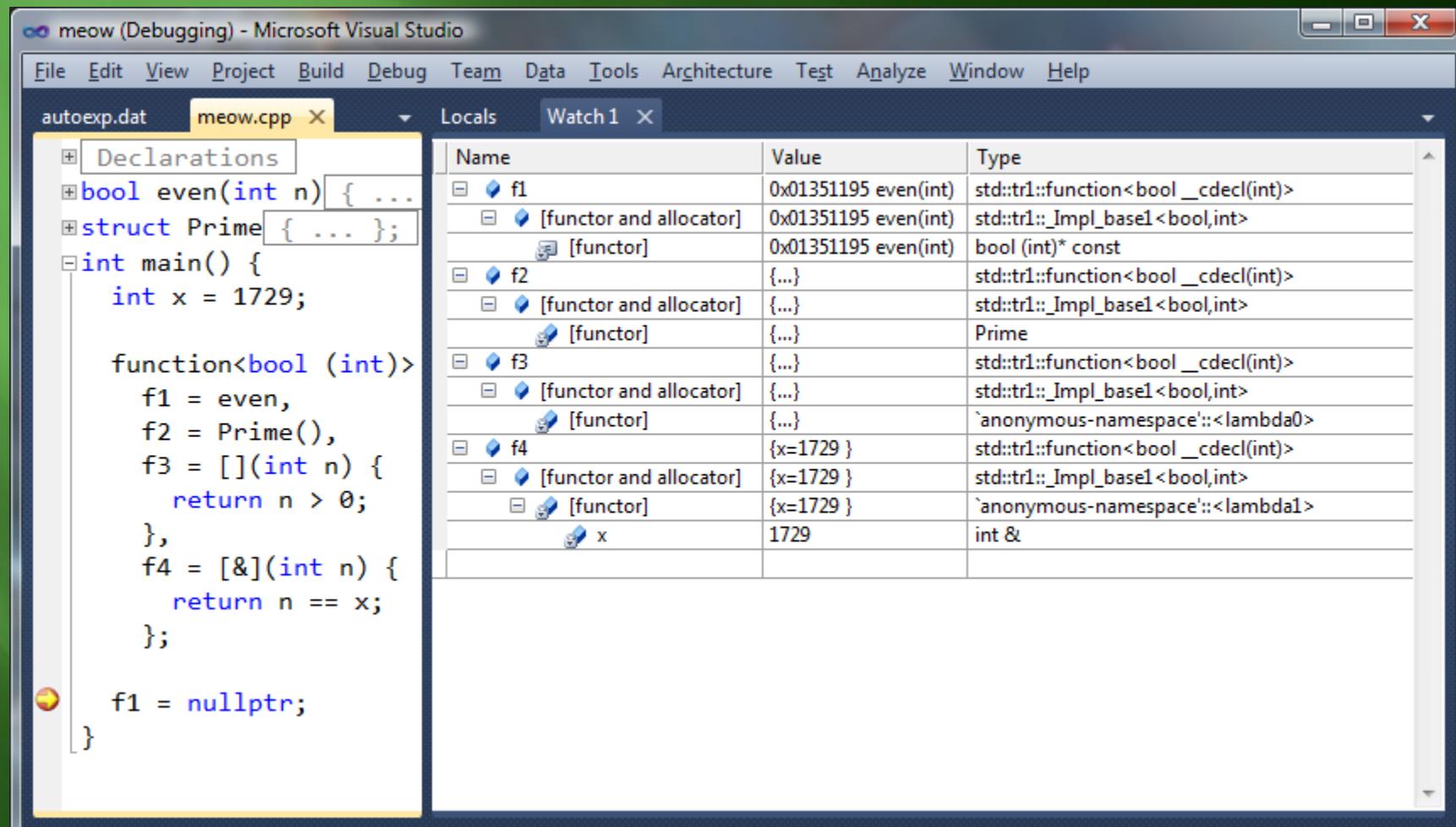
Prime

Stored Functor

function's Visualizer

```
std::tr1::_Impl_no_alloc0<*>|std::tr1::_Impl_no_alloc1<*>|ETC{
    preview ( $e._Callee._Object )
    children ( #([functor] : $e._Callee._Object) )
}
std::tr1::function<*>{
    preview (
        #if ($e._Impl == 0) (
            "empty"
        ) #else (
            *$e._Impl
        ) )
    children (
        #if ($e._Impl == 0) (
            #array(expr: 0, size: 0)
        ) #else (
            #([functor and allocator] : *$e._Impl)
        ) )
}
```

function Visualized



Questions?

- My E-mail address: stl@microsoft.com
- For more information, read Avery Lee's posts:
 - "Writing custom visualizers for Visual Studio 2005"
 - virtualdub.org/blog/pivot/entry.php?id=120
 - "How to fix debugger visualizers for VS2005 SP1 and VS2008"
 - virtualdub.org/blog/pivot/entry.php?id=172
 - "Watch out for number bases in Visual C++ visualizers"
 - virtualdub.org/blog/pivot/entry.php?id=302