# Debugging in C++

Devin A. Matthews
UT Austin

MolSSI Software Summer School 2017

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

- Brian W. Kernighan.

# What exactly is a “bug”?

- **Compile-time bugs**:
  - Syntax errors
  - Semantic errors
  - (Warnings)
- **Link-time bugs**:
  - Missing symbols
  - Duplicate symbols
- **Fatal run-time bugs**:
  - Segmentation fault
  - Assertion failure
  - Exceptions
- **Non-fatal run-time bugs**:
  - Garbage output
  - Subtly wrong output

# Compile-time bugs

- C++ is notorious for generating insanely long (but detailed) error messages.
- Fortunately, GCC and Clang have made a lot of progress in making these readable.
- If you don't have a recent version of one of these that can output errors in color, then upgrade!

# Syntax errors

- **Syntax errors** are when the compiler can't parse your code as valid C++.
- These are usually pretty easy to fix:
  - Missing ;
  - Unbalanced (), [], or {}
  - Unbalanced #if/#ifdef…#endif
  - Missing function return type
  - Return type on constructor or destructor
  - Cat on the keyboard

# Syntax error examples (make sample[1-3].x)

```
 1 #include <cstdio>
 2
 3 int main()
 4 {
 5     int x = 4;
 6     x &= 7;
 7     printf("Look Ma, I can do math: %d\n", x-3)
 8
 9     return 0;
10 }
11
```

```
test.cxx: In function 'int main()':
test.cxx:9:5: error: expected ';' before 'return'
     return;
     ^~~~~~
```

```
 1 #include <cstdio>
 2
 3 int main()
 4 {
 5     printf("Parentheses can help readability: %d\n",
 6         (((24-3)/4)+((7*4)+4));
 7
 8     return 0;
 9 }
10
```

```
test.cxx: In function 'int main()':
test.cxx:6:31: error: expected ')' before ';' token
         (((24-3)/4)+((7*4)+4));
                               ^
```

```
 1 #include <cstdio>
 2
 3 foo()
 4 {
 5     return 3;
 6 }
 7
 8 int main()
 9 {
10     printf("The value: %d\n", foo());
11
12     return 0;
13 }
14
```

```
test.cxx:3:5: error: ISO C++ forbids declaration of 'foo' with no type [-fpermissive]
 foo()
     ^
```

# Semantic errors

- Semantic errors parse as C++, but don't make sense to the compiler.
  - Undefined variable, function, member, or type.
    - This is usually a typo or missed header file
  - Discarded qualifier (usually **const**)
    - Your program is not const-correct!
  - Use of incomplete type
    - For a class, this means you have the forward declaration but not the definition. Can also refer to uses of **void**.
  - Member access in non-class type
    - Either a typo, or accidentally using "." instead of "->" on a pointer.
  - "No match for…"
    - This covers a number of errors related to **name lookup** and **overload resolution**

# “No match for …” (make sample4.x)

```
1 #include <vector>
2 #include <algorithm>
3
4 int main()
5 {
6     int x;
7     std::vector<std::vector<int>> v;
8     auto it = std::find(v.begin(), v.end(), x);
9 }
10
```

```
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:71:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h: In instantiation of 'bool __gnu_cxx::__ops::_Iter_equals_val<_Value>::operator()(_Iterator) [with _Iterator = __
gnu_cxx::__normal_iterator<std::vector<int>*, std::vector<std::vector<int> > >; _Value = const int]':
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algo.h:120:14:   required from '_RandomAccessIterator std::__find_if(_RandomAccessIterator, _RandomAccessIterator, _Predicate,
 std::random_access_iterator_tag) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<std::vector<int>*, std::vector<std::vector<int> > >; _Predicate = __gnu_cxx::__ops::_Ite
r_equals_val<const int>]'
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algo.h:161:23:   required from '_Iterator std::__find_if(_Iterator, _Iterator, _Predicate) [with _Iterator = __gnu_cxx::__norm
al_iterator<std::vector<int>*, std::vector<std::vector<int> > >; _Predicate = __gnu_cxx::__ops::_Iter_equals_val<const int>]'
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algo.h:3907:28:   required from '_IIter std::find(_IIter, _IIter, const _Tp&) [with _IIter = __gnu_cxx::__normal_iterator<std:
:vector<int>*, std::vector<std::vector<int> > >; _Tp = int]'
test.cxx:8:46:   required from here
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h:241:17: error: no match for 'operator==' (operand types are 'std::vector<int>' and 'const int')
   { return *__it == _M_value; }
            ~~~~^~~~~~~~~~
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:67:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_iterator.h:859:5: note: candidate: template<class _IteratorL, class _IteratorR, class _Container> bool __gnu_cxx::operator==(c
onst __gnu_cxx::__normal_iterator<_IteratorL, _Container>&, const __gnu_cxx::__normal_iterator<_IteratorR, _Container>&)
     operator==(const __normal_iterator<_IteratorL, _Container>& __lhs,
     ^~~~~~~~
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_iterator.h:859:5: note:   template argument deduction/substitution failed:
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:71:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h:241:17: note:   'std::vector<int>' is not derived from 'const __gnu_cxx::__normal_iterator<_IteratorL, _Container
>'
   { return *__it == _M_value; }
            ~~~~^~~~~~~~~~
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:67:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_iterator.h:866:5: note: candidate: template<class _Iterator, class _Container> bool __gnu_cxx::operator==(const __gnu_cxx::__n
ormal_iterator<_Iterator, _Container>&, const __gnu_cxx::__normal_iterator<_Iterator, _Container>&)
     operator==(const __normal_iterator<_Iterator, _Container>& __lhs,
     ^~~~~~~~
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_iterator.h:866:5: note:   template argument deduction/substitution failed:
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:71:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h:241:17: note:   'std::vector<int>' is not derived from 'const __gnu_cxx::__normal_iterator<_Iterator, _Container'
   { return *__it == _M_value; }
            ~~~~^~~~~~~~~~
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/x86_64-apple-darwin15.6.0/bits/c++allocator.h:33:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/allocator.h:46,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:61,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/ext/new_allocator.h:155:5: note: candidate: template<class _Tp> bool __gnu_cxx::operator==(const __gnu_cxx::new_allocator<_Tp>&, const
__gnu_cxx::new_allocator<_Tp>&)
     operator==(const new_allocator<_Tp>&, const new_allocator<_Tp>&)
     ^~~~~~~~
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/ext/new_allocator.h:155:5: note:   template argument deduction/substitution failed:
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:71:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h:241:17: note:   'std::vector<int>' is not derived from 'const __gnu_cxx::new_allocator<_Tp>'
   { return *__it == _M_value; }
            ~~~~^~~~~~~~~~
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:64:0,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_vector.h:1596:5: note: candidate: template<class _Tp, class _Alloc> bool std::operator==(const std::vector<_Tp, _Alloc>&, cons
t std::vector<_Tp, _Alloc>&)
     operator==(const vector<_Tp, _Alloc>& __x, const vector<_Tp, _Alloc>& __y)
     ^~~~~~~~
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_vector.h:1596:5: note:   template argument deduction/substitution failed:
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:71:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h:241:17: note:   mismatched types 'const std::vector<_Tp, _Alloc>' and 'const int'
   { return *__it == _M_value; }
            ~~~~^~~~~~~~~~
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:61:0,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/allocator.h:152:5: note: candidate: template<class _Tp> bool std::operator==(const std::allocator<_Tp1>&, const std::allocator<_Tp
1>&)
     operator==(const allocator<_Tp>&, const allocator<_Tp>&)
     ^~~~~~~~
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/allocator.h:152:5: note:   template argument deduction/substitution failed:
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:71:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h:241:17: note:   'std::vector<int>' is not derived from 'const std::allocator<_Tp1>'
   { return *__it == _M_value; }
            ~~~~^~~~~~~~~~
```

```
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:61:0,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/allocator.h:146:5: note: candidate: template<class _T1, class _T2> bool std::operator==(const std::allocator<_Tp1>&, const std::al
locator<_T2>&)
     operator==(const allocator<_T1>&, const allocator<_T2>&)
     ^~~~~~~~
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/allocator.h:146:5: note:   template argument deduction/substitution failed:
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:71:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h:241:17: note:   'std::vector<int>' is not derived from 'const std::allocator<_Tp1>'
   { return *__it == _M_value; }
            ~~~~^~~~~~~~~~
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:67:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_iterator.h:1124:5: note: candidate: template<class _Iterator> bool std::operator==(const std::move_iterator<_IteratorL>&, cons
t std::move_iterator<_IteratorL>&)
     operator==(const move_iterator<_Iterator>& __x,
     ^~~~~~~~
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_iterator.h:1124:5: note:   template argument deduction/substitution failed:
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:71:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h:241:17: note:   'std::vector<int>' is not derived from 'const std::move_iterator<_IteratorL>'
   { return *__it == _M_value; }
            ~~~~^~~~~~~~~~
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:67:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_iterator.h:1118:5: note: candidate: template<class _IteratorL, class _IteratorR> bool std::operator==(const std::move_iterator
<_IteratorL>&, const std::move_iterator<_IteratorR>&)
     operator==(const move_iterator<_IteratorL>& __x,
     ^~~~~~~~
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_iterator.h:1118:5: note:   template argument deduction/substitution failed:
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:71:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h:241:17: note:   'std::vector<int>' is not derived from 'const std::move_iterator<_IteratorL>'
   { return *__it == _M_value; }
            ~~~~^~~~~~~~~~
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:67:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_iterator.h:337:5: note: candidate: template<class _IteratorL, class _IteratorR> bool std::operator==(const std::reverse_iterat
or<_IteratorL>&, const std::reverse_iterator<_IteratorR>&)
     operator==(const reverse_iterator<_IteratorL>& __x,
     ^~~~~~~~
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_iterator.h:337:5: note:   template argument deduction/substitution failed:
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:71:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h:241:17: note:   'std::vector<int>' is not derived from 'const std::reverse_iterator<_IteratorL>'
   { return *__it == _M_value; }
            ~~~~^~~~~~~~~~
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:67:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_iterator.h:299:5: note: candidate: template<class _Iterator> bool std::operator==(const std::reverse_iterator<_Iterator>&, con
st std::reverse_iterator<_Iterator>&)
     operator==(const reverse_iterator<_Iterator>& __x,
     ^~~~~~~~
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_iterator.h:299:5: note:   template argument deduction/substitution failed:
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:71:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h:241:17: note:   'std::vector<int>' is not derived from 'const std::reverse_iterator<_Iterator>'
   { return *__it == _M_value; }
            ~~~~^~~~~~~~~~
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:64:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_pair.h:443:5: note: candidate: template<class _T1, class _T2> constexpr bool std::operator==(const std::pair<_T1, _T2>&, const
 std::pair<_T1, _T2>&)
     operator==(const pair<_T1, _T2>& __x, const pair<_T1, _T2>& __y)
     ^~~~~~~~
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_pair.h:443:5: note:   template argument deduction/substitution failed:
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:71:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h:241:17: note:   'std::vector<int>' is not derived from 'const std::pair<_T1, _T2>'
   { return *__it == _M_value; }
            ~~~~^~~~~~~~~~
```

# “No match for …” part I

The first part tells you **where** the error occurred and **how** the compiler got there.

The error itself occurred here:

```
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobas
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h: In instantiatio
gnu_cxx::__normal_iterator<std::vector<int>*, std::vector<std::vector<int> > >; _Val
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algo.h:120:14:   required fro
 std::random_access_iterator_tag) [with _RandomAccessIterator = __gnu_cxx::__normal_
r_equals_val<const int>]'
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algo.h:161:23:   required fro
al_iterator<std::vector<int>*, std::vector<std::vector<int> > >; _Predicate = __gnu_
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algo.h:3907:28:   required fr
:vector<int>*, std::vector<std::vector<int> > >; _Tp = int]'
test.cxx:8:46:   required from here
```

But you (the programmer) started it off here:

The problem could be anywhere in between.

# “No match for …” part II

The second part tells you **what** specific error occurred. This is generally in **red**, and is the most important part of the message. Go here first.

```
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_op
s.h:241:17: error: no match for 'operator==' (operand types are
'std::vector<int>' and 'const int')
  { return *__it == _M_value; }
           ~~~~~~^~~~~~~~~~~~
```

In this case, it’s telling us:

“You tried to do ‘`std::vector<int> == const int`’, but I can’t find an operator to do that.”

# “No match for …” part III

The last part lists all of the **candidates** that the compiler considered. For each one, it will give a reason why that candidate was rejected.

```
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:67:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_iterator.h:859:5: note: candidate: templ
ate<class _IteratorL, class _IteratorR, class _Container> bool __gnu_cxx::operator==(const __gn
u_cxx::__normal_iterator<_IteratorL, _Container>&, const __gnu_cxx::__normal_iterator<_Iterator
R, _Container>&)
     operator==(const __normal_iterator<_IteratorL, _Container>& __lhs,
     ^~~~~~~~
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_iterator.h:859:5: note:   template argum
ent deduction/substitution failed:
In file included from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/stl_algobase.h:71:0,
                 from /usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/vector:60,
                 from test.cxx:1:
/usr/local/Cellar/gcc/7.1.0/include/c++/7.1.0/bits/predefined_ops.h:241:17: note:   'std::vecto
r<int>' is not derived from 'const __gnu_cxx::__normal_iterator<_IteratorL, _Container>'
  { return *__it == _M_value; }
           ~~~~~~^~~~~~~~~~~
```

In this case, the candidate is an operator having to do with iterators, but neither `std::vector<int>` nor `const int` are iterators.

# “No match for …” part IV

- There are three main reasons for getting the “no match for…” error:
  - The compiler is trying to call a **different function** than you intended.
    - This is our case, because we want to call `operator==(int, int)` for sorting. The fact that it wants to call it with `std::vector<int>` is a clue.
  - The compiler is trying to call the **right function**, but it is **not a candidate**.
    - This could be a problem with **name lookup** or **visibility**.
  - The compiler is trying to call the **right function**, but it is **rejected** as a candidate.
    - Perhaps there is a problem with type conversion or type constraints through `std::enable_if`? This is the hardest one to diagnose.

# The gray area

- C++ is very complex to parse, in particular because ***how*** you parse an expression can depend on ***what types it involves***.

- Some semantic errors actually show up as syntax errors, which can be very confusing.

# Gray area examples (make sample5.x)

```
 1 template <typename T> struct base
 2 {
 3     template <typename U>
 4     U convert() const;
 5 };
 6
 7 template <typename T>
 8 struct foo : base<T>
 9 {
10     foo()
11     {
12         this->convert<double>();
13     }
14 };
15
```

GCC

```
test.cxx: In constructor 'foo<T>::foo()':
test.cxx:12:23: error: expected primary-expression before 'double'
         this->convert<double>();
                       ^~~~~~
test.cxx:12:23: error: expected ';' before 'double'
```

Clang is quite a bit more helpful:

```
test.cxx:12:15: error: use 'template' keyword to treat 'convert' as a dependent template name
        this->convert<double>();
              ^
              template
```

# Gray area examples (make sample6.x)

```
 1 struct member
 2 {
 3     int x = 0;
 4     member() {}
 5     member(int x) : x(x) {}
 6 };
 7
 8 struct foo
 9 {
10     member m(4);
11     foo() {}
12     foo(const member& m) : m(m) {}
13 };
14
```

GCC

```
test.cxx:10:14: error: expected identifier before numeric constant
     member m(4);
              ^
test.cxx:10:14: error: expected ',' or '...' before numeric constant
test.cxx: In constructor 'foo::foo(const member&)':
test.cxx:12:28: error: class 'foo' does not have any field named 'm'
     foo(const member& m) : m(m) {}
                            ^
```

Clang

```
test.cxx:10:14: error: expected parameter declarator
    member m(4);
             ^
test.cxx:10:14: error: expected ')'
test.cxx:10:13: note: to match this '('
    member m(4);
            ^
test.cxx:12:28: error: member initializer 'm' does not name a non-static data member or base class
    foo(const member& m) : m(m) {}
                           ^~~~
```

# Link-time errors

- The linker may complain that it cannot find a **symbol** (a symbol is either a variable or function). Here are some common causes:
  - Typo
  - Does the signature of the function definition match the prototype?
  - Are you including all of the right object files and libraries?
  - Is the template instantiation visible?
    - This can happen if the template is defined in a different place than it is declared.
- The linker may also complain that there are **duplicate symbols**. Common reasons:
  - A non-inline function in a header file.
  - Functions in different files accidentally share a name.
  - Object file included twice.

# Run-time errors

- If the program compiles and links, then one of three things can happen when you run it:
  - It runs correctly (yay!).
  - It finishes, but gets the wrong result.
  - It doesn't finish (crashes).

- The most common type of a crash is a **segmentation fault** (**segfault**, sometimes you also get the related **bus error**). This means the program tried to access a region of memory that it wasn't assigned.

# Debugging segfaults I (make sample7.x)

- The first step in debugging a segfault is to find out where the program crashed.
- The best thing to do is use a **debugger** for this, such as **gdb** or **lldb**.
  1. Compile with debugging turned on: **-g –O0**
  2. Run in the debugger: **gdb ./sample7.x**
  3. Get a backtrace:

```
(gdb) r
Starting program: /home/dmatthews/test.x

Program received signal SIGSEGV, Segmentation fault.
0x00000000004005cf in fail () at test.cxx:3
3           *(int*)0 = 0;
(gdb) bt
#0  0x00000000004005cf in fail () at test.cxx:3
#1  0x00000000004005e1 in main () at test.cxx:8
(gdb)
```

It died here

These are the **stack frames** at the time of failure. They represent all of the functions called to get to this point.

# Debugging segfaults II

- The reason that the program segfaulted may be immediately evident if the root cause it the same place the error occurs (a **direct segfault**).
- But, the real error may have occurred way back somewhere, and silently **corrupted** the memory in a way that caused the segfault later on (an **indirect segfault**).
  - This comes in two flavors: **heap corruption** and **stack corruption**.
- Lastly, the error could be due **undefined behavior**. This is generally caused by reading **uninitialized data**.

# Debugging segfaults III (make sample8.x)

- If the problem doesn't look like it is caused by a direct segfault, we have to follow the chain of causality backwards until we find the error.
- The debugger can help with this by **examining variables**, moving through the **stack frames**, and setting **breakpoints** and **watchpoints**.

```
(gdb) r
Starting program: /home/dmatthews/test.x

Program received signal SIGBUS, Bus error.
0x00000000004005f7 in main () at test.cxx:7
7             array[i+j*lda] = 1.0;
(gdb) print i
$1 = 0
(gdb) print lda
$2 = 10
(gdb) print j
$3 = 32767
(gdb) ptype array
type = double [100]
(gdb)
```

"print" is pretty versatile, for example it can print the result of an expression:

```
(gdb) print i+j*lda
$4 = 327670
```

# Debugging segfaults IV (make sample9.x)

- If the current function isn't the culprit, we can examine the function that call it, and the function that call it, and so on.

```
(gdb) r
Starting program: /home/dmatthews/test.x

Program received signal SIGSEGV, Segmentation fault.
0x0000000000400806 in get_matrix_element (array=0x614c20, i=3, j=894, lda=1000) at test.cxx:5
5           return array[i+j*lda];
(gdb) bt
#0  0x0000000000400806 in get_matrix_element (array=0x614c20, i=3, j=894, lda=1000) at test.cxx:5
#1  0x0000000000400885 in main () at test.cxx:13
(gdb) up
#1  0x0000000000400885 in main () at test.cxx:13
13          get_matrix_element(matrix.data(), 3, 894, n);
(gdb) list
8       int main()
9       {
10          int m = 10;
11          int n = 1000;
12          std::vector<double> matrix(m*n);
13          get_matrix_element(matrix.data(), 3, 894, n);
14      }
15
(gdb)
```

`get_matrix_element` is just fine, but we called it with the wrong arguments (row-major instead of column-major).

# Debugging segfaults V (make sample10.x)

- Sometimes it is extremely difficult to find the source of **heap corruption** or **uninitialized data**. In this case, a **dynamic analysis** tool can help pinpoint the cause. One popular tool for this is **valgrind**.

This code tries to access element (11,99) of a 10x100 matrix.

```
[dmatthews@twiddle ~]$valgrind --track-origins=yes ./test.x
==6480== Memcheck, a memory error detector
==6480== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==6480== Using Valgrind-3.10.0 and LibVEX; rerun with -h for copyright info
==6480== Command: ./test.x
==6480==
==6480== Invalid read of size 8
==6480==    at 0x400806: get_matrix_element(double*, int, int, int) (test.cxx:5)
==6480==    by 0x400884: main (test.cxx:13)
==6480==  Address 0x5a89bc8 is 8 bytes after a block of size 8,000 alloc'd
==6480==    at 0x4C29670: operator new(unsigned long) (in /usr/lib64/valgrind/vgpr
==6480==    by 0x400E57: __gnu_cxx::new_allocator<double>::allocate(unsigned long,
==6480==    by 0x400DC4: __gnu_cxx::__alloc_traits<std::allocator<double> >::alloc
==6480==    by 0x400D15: std::_Vector_base<double, std::allocator<double> >::_M_al
==6480==    by 0x400C02: std::_Vector_base<double, std::allocator<double> >::_M_cr
==6480==    by 0x400A6E: std::_Vector_base<double, std::allocator<double> >::_Vect
==6480==    by 0x40093B: std::vector<double, std::allocator<double> >::vector(unsi
==6480==    by 0x400855: main (test.cxx:12)
```

# Debugging segfaults VI

- Another powerful technique for diagnosing the cause of a segfault is **assertions**. An assertion states a condition that should always be true. When an assertion is triggered, you can also examine the reason in the debugger.

Assertions can be disabled with the compiler flag –DNDEBUG, so they won't slow down production code.

```
(gdb) r
Starting program: /home/dmatthews/test.x
test.x: test.cxx:14: double& matrix::get_element(int, int): Assertion `i > 0 && i < this->m' failed.

Program received signal SIGABRT, Aborted.
0x00007ffff71d00a7 in raise () from /lib64/libc.so.6
(gdb) up
#1  0x00007ffff71d1458 in abort () from /lib64/libc.so.6
(gdb) up
#2  0x00007ffff71c9126 in __assert_fail_base () from /lib64/libc.so.6
(gdb) up
#3  0x00007ffff71c91d2 in __assert_fail () from /lib64/libc.so.6
(gdb) up
#4  0x0000000000400872 in matrix::get_element (this=0x7fffffffc410, i=11, j=99) at test.cxx:14
14          assert(i > 0 && i < this->m);
(gdb) print i
$1 = 11
(gdb) print m
$2 = 10
(gdb)
```

# Dealing with incorrect results

- Incorrect results can be very tricky to debug, but when the cause is the same an indirect segfault:
  - **Stack corruption**
  - **Heap corruption**
  - **Uninitialized data**
  - **Assertion failure** (but assertions are disabled)
- Then the techniques for diagnosing segfaults also apply.
- If all else fails, there are always print statements!

# Dealing with incorrect results

- Proper testing, including unit tests, along with assertions and safe best practices is the best way to avoid bugs.

Questions?