C++ Concepts

Christopher Jefferson
History of C++

- 1969-1973 C developed at Bell Labs
- 1979 "C with Classes"
- 1983 C with Classes becomes C++
- 1990 Templates and Exceptions added to C++
- 1994 First complete ANSI/ISO Standard Draft
- 1998 C++ Standard released
- 2003 Bugfixed standard released
- 2005 Technical Report 1 released
- 2011 C++11 released
- 2014 C++14(?)
- 2017 C++17(?)
ISO Committee

- The C++ committee is known as: 'ISO JTC1/SC22/WG21'.
- Also the ANSI/NCITS/J16
- Meets 3 times a year, in various places around the world to discuss standardisation.
- Most decisions are made by vote, hopefully unanimously (or close).
ISO Committee

• ~20 nations represented.
• Most members:
  • Work in industry
  • Are volunteers (even company representatives)
• Every (major) OS and compiler is represented.
Concepts

- The biggest planned feature for C++11.
- Would have touched every part of the C++ standard.
- Thrown out at almost the last possible minute.
Templates

- Templates are one of the major features of C++.
- They provide a way of providing generic types and functions, without run-time cost.
- They use "duck-typing" rather than interfaces.
What are templates?

```cpp
template<typename T>
T mult(T first, T second)
{
    return first * second;
}

mult(1,2);
mult("Hello ", "world"); // Woops
```
template<typename T>
struct S;

template<>
struct S<int>
{
int f() { return 1; }
};

template<>
struct S<float>
{
float g() { return 1.0; }
};
Template Problems

- Templates provide a turing-complete compile time programming language.
- The biggest weakness of templates is error handling.
Example Error

/usr/include/c++/4.2.1/bits/stl_heap.h: In function `void std::__push_heap(_RandomAccessIterator, _Distance, _Distance, _Tp) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<X*, std::vector<X, std::allocator<X> > >, _Distance = long int, _Tp = X]':

/usr/include/c++/4.2.1/bits/stl_heap.h:227:   instantiated from `void std::__adjust_heap(_RandomAccessIterator, _Distance, _Distance, _Tp) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<X*, std::vector<X, std::allocator<X> > >, _Distance = long int, _Tp = X]'

/usr/include/c++/4.2.1/bits/stl_heap.h:364:   instantiated from `void std::make_heap(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<X*, std::vector<X, std::allocator<X> > >]'

/usr/include/c++/4.2.1/bits/stl_algo.h:2479:   instantiated from `void std::__heap_select(_RandomAccessIterator, _RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<X*, std::vector<X, std::allocator<X> > >]'

/usr/include/c++/4.2.1/bits/stl_algo.h:2551:   instantiated from `void std::partial_sort(_RandomAccessIterator, _RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<X*, std::vector<X, std::allocator<X> > >]'

/usr/include/c++/4.2.1/bits/stl_algo.h:2746:   instantiated from `void std::__introsort_loop(_RandomAccessIterator, _RandomAccessIterator, _Size) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<X*, std::vector<X, std::allocator<X> > >, _Size = long int]'

/usr/include/c++/4.2.1/bits/stl_algo.h:2829:   instantiated from `void std::sort(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<X*, std::vector<X, std::allocator<X> > >]'

/usr/include/c++/4.2.1/bits/stl_heap.h:121: error: no match for `operator<' in `__first.
__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator+ [with _Iterator = X*, _Container = std::vector<X, std::allocator<X> > ](((const ptrdiff_t&)((const ptrdiff_t&)(__parent))).__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator* [with _Iterator = X*, _Container = std::vector<X, std::allocator<X> > ]()) < __value'

/usr/include/c++/4.2.1/bits/stl_heap.h:227: instantiated from `void std::__adjust_heap(_RandomAccessIterator, _Distance, _Distance, _Tp) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<X*, std::vector<X, std::allocator<X> > >, _Distance = long int, _Tp = X]'

/usr/include/c++/4.2.1/bits/stl_heap.h:364: instantiated from `void std::make_heap(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<X*, std::vector<X, std::allocator<X> > >]'

/usr/include/c++/4.2.1/bits/stl_algo.h:2479: instantiated from `void std::__heap_select(_RandomAccessIterator, _RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<X*, std::vector<X, std::allocator<X> > >]'

/usr/include/c++/4.2.1/bits/stl_algo.h:2551: instantiated from `void std::partial_sort(_RandomAccessIterator, _RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<X*, std::vector<X, std::allocator<X> > >]'

/usr/include/c++/4.2.1/bits/stl_algo.h:2746: instantiated from `void std::__introsort_loop(_RandomAccessIterator, _RandomAccessIterator, _Size) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<X*, std::vector<X, std::allocator<X> > >, _Size = long int]'

/usr/include/c++/4.2.1/bits/stl_algo.h:2829: instantiated from `void std::sort(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<X*, std::vector<X, std::allocator<X> > >]'

t.cc:9: instantiated from here

/usr/include/c++/4.2.1/bits/stl_heap.h:121: error: no match for `operator<' in `_first.
__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator+ [with _Iterator = X*, _Container = std::vector<X, std::allocator<X> > ](((const ptrdiff_t&)((const ptrdiff_t&)(__parent))).__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator* [with _Iterator = X*, _Container = std::vector<X, std::allocator<X> > ]()) < __value'
Example Error

include/stl_heap.h: In function `void std::__push_heap`

include/stl_heap.h:227: instantiated from
`void std::__adjust_heap`

include/stl_heap.h:364: instantiated from `void std::make_heap`

include/stl_algo.h:2479: instantiated from `void std::__heap_select`

include/stl_algo.h:2551: instantiated from `void std::partial_sort`

include/stl_algo.h:2746: instantiated from `void std::__introsort_loop`

include/stl_algo.h:2829: instantiated from `void std::sort`

mycode.cc:9:sort_func.cpp instantiated from here

include/stl_heap.h:121: error: no match for `operator<` in `X`
Solution: Concepts

template<typename T>
requires Comparable<*T> && SwappableView<T>
void sort(T begin, T end);
Concepts in theory

- When a concepted function is parsed, we check it only uses the concepts listed.

- When we want to use a type in a concepted function, first check it satisfies the concepts.
What went wrong?

• Problems with concepts fell into 3 categories:
  • Allowing code from previous versions of C++ to interact.
  • Coming up with clean semantics.
  • Working with other new features in C++11.
Comparable<T>  requires T<T is bool

Does: int operator<(T, T); match? Lots of people write int return values.

But then, maybe we use the expression with:

void f(int);
void f(bool);
Convertible to bool

- Operations like \( x < y \) and \( x = y \) return a boolean value.
- Some "clever" users would write code like:
  ```c
  int notequal(int x, int y)
  { return x - y; }
  ```
- It was considered "free" to support such code, by saying comparisons only had to return a value which was convertible to bool, rather than actually a bool.
Convertible to bool problems

• Problem 1:
  • \((x \neq y) == (a < b)\)
  • If \(!=\) turns 1 for true, and \(<\) returns 2, this expression will be false.
Convertible to bool problems

- if( x < y && a == 0) { ... 
- This is fine if < and == return an int.
struct HorribleReturnType
{
    // Act like a bool except
    bool operator&&(bool) { abort(); }
};
Instantiating Concepts

- So, we can have a situation where:
  - Type satisfies concept
  - Function only uses concept
  - But type does not work with function.

- Solution:
  - Force every input to a function, and output from a function to exactly the type in the concept.
Addition Concept

class Addable<T>
{
    T operator+(T, T);
}
Addition Concept

template<typename T>
requires Addable<T>
T addthree(T t1, T t2, T t3)
{ return t1 + t2 + t3; }
Addition Concept

template<typename T>
requires Addable<T>
T addthree(T t1, T t2, T t3)
{
    return
    (T)(((T)t1 +
        (T)(((T)t2 + (T)t3)));
}
The problem of temporaries

String a, b, c;

... 

String d = (a+b)+c;

Is implemented as:

String temp = a+b;
String d = temp+c;

We would prefer something like:

String d = a+b;
String d += c;
RValue Reference

• Normal (lvalue) reference:  `Type&`
Rvalue reference:  `Type&&`

• Denotes "This is a reference to a temporary I don't care about".

• `string operator+(string&& a, string& c)`

• We reuse the memory of `a`, instead of allocating new memory.
Advantages of Rvalue References

• Users can get practical benefits without having to understand rvalue references:

• Compiler introduces rvalue references where possible.

• Users can do this explicitly with `move(x)`
String Addition

operator+(string&&, const string&);
operator+(const string&, string&&);
operator+(char*, const string&);
operator+(char*, const string&&);
operator+(const string&, char*);
operator+(const string&&, char*);
operator+(const string&, const string&);
operator+(string&&, string&&);
Long concepts

template<class InIter, class Pred, class T>

requires InputIterator<InIter> &&
Predicate<InIter::value_type> Pred &&
OutputIterator<Iter, Iter::reference> &&
OutputIterator<Iter, const T&> &&
CopyConstructible<Pred>

OutIter replace_if(InIter first, InIter last,
                  Pred pred, const T& new_value);

{
    for ( ; __first != __last; ++__first, ++__result)
        if (__pred(*__first))
            *__result = __new_value;
}
Concepting Everything

• Concepts were supposed to cover all of C++.
  • This means there has to be a concept to cover every kind of type of C++.
  • There are a lot of nasty corner cases!
C++14 Concepts

- Back to basics.
- What is the practical problem we are trying to solve?
Example Error

include/stl_heap.h: In function ‘\texttt{void std::__push_heap}’
include/stl_heap.h:227: instantiated from ‘\texttt{void std::__adjust_heap}’
include/stl_heap.h:364: instantiated from ‘\texttt{void std::make_heap}’
include/stl_algo.h:2479: instantiated from ‘\texttt{void std::__heap_select}’
include/stl_algo.h:2551: instantiated from ‘\texttt{void std::partial_sort}’
include/stl_algo.h:2746: instantiated from ‘\texttt{void std::__introsort_loop}’
include/stl_algo.h:2829: instantiated from ‘\texttt{void std::sort}’
\texttt{mycode.cc:9:sort_func.cpp} instantiated from here
include/stl_heap.h:121: error: no match for ‘operator<’ in \texttt{X}
C++14 Concepts

- Also known as 'Concepts Lite'
- Andrew Sutton, Bjarne Stroustrup, Gabriel Dos Reis

- Define predicates on types.
- Functions can require predicates are satisfied.
  - But there is no checks done on if functions require things other than these predicates.
Predicates

• A range of predicates are built in:
  
  `swappable<T>()`
  `streamable<T>()`
  ....

• Also, a simple way of checking if code will compile is provided.
Concept Example

template<typename T>
constexpr bool Equality_comparable()
{
    return requires (T a, T b) {
        bool = {a == b};
        bool = {a != b};
    };
}
Concepts Lite

• Concepts can be inherited, which provides a way of specialising functions.

• Error messages are much better than previously (and getting better).

• Code can still use full C++ optimisations inside functions.
Concepts Lite

- There are no guarantees that functions will compile even when concepts are satisfied.
- The error messages in these cases are as bad as existing C++.