MODERN C++ FOR FUN AND PROFIT

AN INTRODUCTION TO MODERN C++ AND WHY WE USE IT

ETE 2016

Created by Andy Webber / andy.webber@sig.com

- Motivation
- Tour
- Optimization
- Conclusion

MOTIVATION

Use the right language for the job.

- paraphrased from Bjarne Stroustrup

WHEN IS C++ THE RIGHT TOOL?

- Performance
- Cross platform
- Expressiveness
- Correctness?

ASPECTS OF C++

- Statically typed
- Natively compiled
- Deterministic object lifetime
- Pay for what you use
- Compile time computation

MULTI-PARADIGM

- object oriented
- generic
- functional
- procedural
- mix and match!

OBJECT LIFETIME

- stack vs. heap
- deterministic destruction

```
// stack variable
void foo()
{
   resource_lock lock(my_resource);
} // release lock on my resource here
```

```
// heap variable
void bar()
{
   resource_lock* pointer_to_lock(new resource_lock(my_resource));
} // keep my resource locked
```

OLD SCHOOL - ISO C++98/03

- Good performance
- Annoying boilerplate
- Minimal standard library
- Boost libraries

MODERN TIMES - C++11/14/17

- Better performance by default
- Easier to write
- Clearer code
- Standard libraries rolling in

TOUR

AUTO

```
auto i = 10; // i is int

auto s = "hello world"; // s is char const*

auto ss = "hello world"s; // ss is std::string

std::vector<double> vec;
...
auto iter = vec.cbegin(); // iter is std::vector<double>::const_iterator
```

INITIALIZER LISTS

```
std::vector<int> vec;
vec = { 2, 4, 6, 8, 10 };
```

```
struct Player
{
   std::string name;
   unsigned number;
};
using TeamList = std::unordered_map<std::string, std::vector<Player>>;
TeamList class_list =
{
   { "Wildcats", { { "Arcidiacono", 15 }, { "Jenkins", 2 } } },
   { "Terrapins", { { "Trimble", 2 }, { "Sulaimon", 0 } } }
};
```

RANGE-FOR

```
for(auto& team: team_list)
// class is reference to TeamList::value_type
{
   for(auto& player: team.second)
   // player is reference to std::vector<Player>::value_type
   {
      update_stats(player);
   }
}
```

LAMBDAS

```
std::vector<int> vec = { 8, 2, 6, 4, 10, 0 };

ranges::sort(vec);

// reversed?
ranges::sort(vec, [](int x, int y){ return x > y; });
```

RANGES

```
std::vector<int> vec = { 8, 2, 6, 4, 10, 0 };
auto mult = 2;
auto filter_below = 10;

for(auto x: vec | view::reverse
    | view::transform([mult](int i){ return i * mult; })
    | view::remove_if([filter_below](int i){ return i < filter_below; })
{
   std::cout << x << " ";
}
// 20 12 16
```

MOVE

```
std::vector<char> big_vec(2 * 1024 * 1024);

// expensive copy
auto my_copy = big_vec;

// cheap move
auto my_moved = std::move(big_vec);

// factories
auto big_object = big_object_factory(object_config);

// move only
auto my_thread = thread_factory(thread_config);
```

MOVE DETAILS

```
template<typename T>
class MyQueue
{
   // ...
   void Push(T const& ref)
   {
      // copy ref here
   }
```

```
   void Push(T&& rvalue_ref)
   {
      // steal rvalue_ref's guts here
   }
};
```

SMART POINTERS

```
std::unique_ptr<int> p = new int{5};
move_to_other_function(std::move(p));
```

```
auto p = std::make_unique<int>(5);
```

```
std::shared_ptr<double> p = new double{42.0};
share_with_other_function(p);
```

```
auto p = std::make_shared<int>(5);
```

VARIADIC TEMPLATES

```
template<typename... Ts>
struct allowed_for_printf
{
   static constexpr bool value =
      ((std::is_arithmetic_v<Ts> || std::is_pointer_v<Ts>) &&
      ... && true);
};
```

```
template<typename... Ts>
void my_printf(Ts... ts);
{
   static_assert(allowed_for_printf<Ts...>::value,
      "bad printf type");
   // do print
}
```

CONCEPTS

```
template<typename C>
bool binary_search(C const& container)
{
   auto middle = container.begin();
   for(auto i = 0; i < container.size() / 2; ++i)
      ++middle;
   // ...
}

// concepts
template<ContiguousContainer C>
bool binary_search(C const& container)
{
   auto middle = container.begin();
   middle += container.size() / 2;
   // ...
}
```

OPTIMIZATION

CODE V1

```cpp
#include <iostream>
#include <vector>

template<typename T>
auto pow(T base, T exponent)
{
   auto c = base;
   for(auto i = 1; i != exponent; ++i)
      c *= base;
   return c;
}


int main()
{
   auto constant = pow(10, 6);

   auto v = std::vector<int>(10);
   for(int i = 0; i <= 8; ++i)
   {
      v[i] = i * constant;
   }
}
```

CODE WITH CONSTEXPR

```
#include <iostream>
#include <vector>

template<typename T>
constexpr auto pow(T base, T exponent)
{
   auto c = base;
   for(auto i = 1; i != exponent; ++i)
      c *= base;
   return c;
}


int main()
{
   constexpr auto constant = pow(10, 6);

   auto v = std::vector<int>(10);
   for(int i = 0; i <= 8; ++i)
   {
      v[i] = i * constant;
   }
}
```

ASSEMBLY: MAIN

```
main:
.LFB1746:
   // ...
   movl  $1000000, -24(%rbp)
   leaq  -25(%rbp), %rax
   // ....
.LEHB0:
   call  _ZNSt6vectorIiSaIiEEC1EmRKS0_ // vector constructor
   // ...
   movl  $0, -20(%rbp)
.L7:
   cmpl  $8, -20(%rbp)
   jg .L6 // exit loop
   movl  -20(%rbp), %eax
   movslq   %eax, %rdx
   leaq  -64(%rbp), %rax
   movq  %rdx, %rsi
   movq  %rax, %rdi
   call  _ZNSt6vectorIiSaIiEEixEm // vector operator[]
   movq  %rax, %rdx
   movl  -20(%rbp), %eax
   imull -24(%rbp), %eax // do multiplication
   movl  %eax, (%rdx)
   addl  $1, -20(%rbp) // increment index
   jmp   .L7 // loop
.L6:
   leaq  -64(%rbp), %rax
```

```
movq  %rax, %rdi
call  _ZNSt6vectorIiSaIiEED1Ev // vector destructor
movl  $0, %eax
jmp   .L11
// ...
ret
```

GCC INTERNALS: -F-DUMP-TREE-ALL

- main.cpp.003t.original
- main.cpp.004t.gimple
- main.cpp.027t.einline
- main.cpp.073t.cunrolli
- main.cpp.190t.optimized

ASSEMBLY OPTIMIZED: -O3

```
movl  $40, %edi
call  _Znwm // new(int) on 8 x ints
movq  $0, 32(%rax)
movl  $0, (%rax)
movq  %rax, %rdi
movl  $1000000, 4(%rax) // constants saved directly
movl  $2000000, 8(%rax)
movl  $3000000, 12(%rax)
movl  $4000000, 16(%rax)
movl  $5000000, 20(%rax)
movl  $6000000, 24(%rax)
movl  $7000000, 28(%rax)
movl  $8000000, 32(%rax)
call  _ZdlPv
xorl  %eax, %eax
addq  $8, %rsp
.cfi_def_cfa_offset 8
ret
```

CONCLUSION

CONCLUSION

- Learn
- Experiment
- Have fun
- Profit!

THANK YOU!

QUESTIONS?

FURTHER READING

- Bjarne Stroustrup
  - A Tour of C++
- Scott Meyers
  - Effective C++ / Effective Modern C++
- Herb Sutter
  - GOTW / C++ Coding Standards / Exceptional C++
- isocpp.org

C++17

- Parallelism TS (Parallel STL)
- Filesystem TS (based on boost.filesystem)
- ```
  if constexpr(is_integral_v<T>) // static if
  ```
- template parameter deduction for constructors
  - ```
    auto p = std::pair("hello", "world")
    ```
- deterministic order of expression evaluation
- In TS or TS coming
  - Ranges
  - Modules
  - Concepts
  - Networking