PIMP MY LOG()

CHEAP, SIMPLE, AND SAFE LOGGING USING C++ EXPRESSION TEMPLATES

Marc Eaddy, Intel

Sponsors of Tomorrow.™

A SIMPLE LOG MACRO

```
#define LOG(msg)                \
    if (s_bLoggingEnabled) \
        std::cout << __FILE__ << "(" << __LINE__ << "): " << msg << std::endl


void foo() {
    string file = "blah.txt";
    int error = 123;
    ...
    LOG("Read failed: " << file << " (" << error << ")");
}

// Outputs:
// test.cpp(5): Read failed: blah.txt (123)
```

Convenient and type safe streaming interface

AFTER PRE-PROCESSING

```
void foo() {
    string file = "blah.txt";
    int error = 123;
    ...
    if (s_bLoggingEnabled)
        std::cout << "foo.cpp" << "(" << 53 << "): "
                  << "Read failed: " << file << " (" << error << ")")
                  << std::endl;
}
```

ASSEMBLER

```
void foo() {
    string file = "blah.txt";
    int error = 123;
    ...
movb  g_bLogging(%rip), %al
testb %al, %al
je    ..B2.14
movl  $_ZSt4cout, %edi
movl  $.L_2__STRING.3, %esi
call  ostream& operator<<(ostream&, ch
movq  %rax, %rdi
movl  $.L_2__STRING.0, %esi
call  ostream& operator<<(ostream&, char const*)
movq  %rax, %rdi
movl  $19, %esi
call  ostream::operator<<(int)
...
movq  %rax, %rdi
movl  $ostream& endl(ostream&), %esi
call  ostream::operator<<(ostream& (*)(ostream&))
}
```

```
% icc --std=c++11 -O3 -g -fcode-asm -S test.cpp
```

33 instructions
10 function calls

PROBLEM

- Log-related instructions...
  - may prevent compiler optimizations
  - may hurt icache performance
- Goal: Reduce instructions at call site but retain
  - Speed
  - Type safety
  - Convenience

A SOLUTION

- How to retain streaming interface without op<< function calls at call site?
  - Evaluate expressions at compile-time instead of runtime
- "Expression templates" use operator overloading to pack an expression into a type
  - `Matrix D = A + B * C;`
  - `polygons.Find(VERTICES == 4 && WIDTH >= 20);`
  - `LOG("Read failed: " << file << " (" << error << ")");`

Adapted from Vaughn Cato's solution
http://stackoverflow.com/questions/19415845/a-better-log-macro-using-template-metaprogramming

LOGDATA<>

```
#define LOG(msg)                      \
    if (s_bLoggingEnabled) \
        (Log(__FILE__, __LINE__, LogData<None>() << msg))

template<typename List>
struct LogData {
    typedef List type;
    List list;
};

struct None { };
```

LOGDATA<>

```
template<typename Begin, typename Value>
LogData<std::pair<Begin&&, Value&&>> operator<<(LogData<Begin>&& begin,
                                                  Value&& v) noexcept {
    return {{ std::forward<Begin>(begin.list), std::forward<Value>(v) }};
}
```

LOGDATA<>

```
LOG("Read failed: " << file << " (" << error << ")");
```

```
LogData<
        pair<
             pair<
                  pair<
                       pair<
                            pair<
                                 None,
                                 char const*>,      "Read failed: "
                            string const&>,         file
                       char const*>,                " ("
                  int const&>,                      error
             char const*>                           ")"
        >
```

LOG()

```
template<typename TLogData>
void Log(const char* file, int line, TLogData&& data) noexcept __attribute__((__noinline__)) {
    std::cout << file << "(" << line << "): ";
    Log_Recursive(std::cout, std::forward<typename TLogData::type>(data.list));
    cout << endl;
}

template<typename TLogDataPair>
void Log_Recursive(std::ostream& os, TLogDataPair&& data) noexcept {
    Log_Recursive(os, std::forward<typename TLogDataPair::first_type>(data.first));
    os << std::forward<typename TLogDataPair::second_type>(data.second);
}

inline void Log_Recursive(std::ostream& os, None) noexcept
{ }
```

HANDLE STREAM MANIPULATORS (EG ENDL)

```
typedef std::ostream& (*PfnManipulator)(std::ostream&);

template<typename Begin>
LogData<pair<Begin&&, PfnManipulator>> operator<<(LogData<Begin>&& begin,
                                                  PfnManipulator pfn)
                                                  noexcept {
    return {{ std::forward<Begin>(begin.list), pfn }};
}
```

STRING LITERAL OPTIMIZATION

```
template<typename Begin, size_t n>
LogData<pair<Begin&&, const char*>> operator<<(LogData<Begin>&& begin,
                                              const char (&sz)[n])
                                              noexcept {
    return {{ std::forward<Begin>(begin.list), sz }};
}
```

Specialization handles all string literals

ASSEMBLER

```
movb  g_bLogging(%rip), %al
testb %al, %al
je    ..B6.7
movb  $0, (%rsp)
movl  $.L_2__STRING.4, %ecx
movl  $.L_2__STRING.3, %edi
movl  $40, %esi
lea   128(%rsp), %r9
call  void Log<pair<pair<pair<pair<pair<None, char const*>,
string const&>, char const*>, int const&>, char const*> >(char
const*, int, LogData<pair<pair<pair<pair<pair<None, char
const*>, string const&>, char const*>, int const&>, char const*>
> const&)
```

9 instructions
1 pimp'd function call

SUMMARY

- Expression templates solution
  - Reduced instructions at call site by 73% (33 → 9)
  - Mo' args, mo' savings

THANK YOU!

VARIADIC TEMPLATE SOLUTION

```
#define LOG(...) Log(__FILE__, __LINE__, __VA_ARGS__)

template<typename... Args>
void Log_Variadic(const char* file, int line, const Args&... args) {
    std::cout << file << "(" << line << "): ";
    Log_Recursive(file, line, std::cout, args...);
    std::cout << std::endl;
}

template<typename T, typename... Args>
void Log_Recursive(const char* file, int line, std::ostream& os, T first, const Args&... rest) {
    os << first;
    Log_Recursive(file, line, os, rest...);
}

inline void Log_Recursive(const char* file, int line, std::ostream& os) { /* Empty */ }
```

VARIADIC TEMPLATE ASSEMBLER

```
movb  g_bLogging(%rip), %al
testb %al, %al
je    ..B1.7
addq  $-16, %rsp
movl  $.L_2__STRING.3, %edi
movl  $26, %esi
movl  $.L_2__STRING.4, %edx
movl  $.L_2__STRING.5, %r8d
lea   24(%rsp), %rcx
lea   32(%rsp), %r9
movq  $.L_2__STRING.6, (%rsp)
call  void Log_Variadic<char [14], string, char [3], int, char [2]>(
        char const*, int, char const (&) [14], string const&,
        char const (&) [3], int const&, char const (&) [2])
```

12 instructions
1 funky function call

VARIADIC TEMPLATE SOLUTION

- Con: lose streaming convenience

```
LOG("Read failed: " << file << " (" << error << ")");
```

⬇

```
LOG("Read failed: ",  file,     " (",   error,   ")");
```