Italian C++ Community

www.italiancpp.org

Lambda Out

Davide Di Gennaro

Bloomberg

Italian C++ Conference 2017
17 Giugno, Milano

Thanks to the sponsors!

Italian C++ Community

Bloomberg

JET BRAINS

think-cell

abacogroup.eu

KDAB

JFrog | CONAN C/C++ package manager

RECOGNITION ROBOTICS
The Visual Guidance Company

Algorithm I/O

- Un algoritmo è una device con input e output
- In C++ 98 lo scambio di dati è modellato con gli iteratori
  - `std::copy(iBegin, iEnd, oBegin);`
- Creare un input è piuttosto semplice, creare un output no
  - L'output potrebbe essere multiplo
  - Lo storage di output ha esigenze diverse a seconda dello step successivo dell'algoritmo
    - Copiare in un vector, list, set…
    - Spostare con move
    - Usare emplace

Algorithm I/O – C++11

- In C++11 vengono introdotte le lambda expression
  - c'è differenza fra funtori e lambda: la facilità d'uso cambia il linguaggio
    - pensate alla lingua degli sms
- Alcuni concetti del C++98 scricchiolano:
  - decltype(begin) != decltype(end)
  - non è obbligatorio usare lo stesso pattern [begin, end) per descrivere sia input che output

Iterator In / Iterator Out

```
template <class IN, class OUT>
OUT copy(IN first, IN last, OUT out)
{ while (first != last) { *out++ = *first++; } return out; }
```

Iterator In / Lambda Out

```
template <class IN, class OUT>
OUT copy(IN first, IN last, OUT out)
{ while (first != last) { *out++ = *first++; } return out; }
```

```
template <class IN, class OUT>
OUT copy(IN first, IN last, OUT out)
{ while (first != last) { out(*first++); } return out; }
```

Iterator In / Lambda Out

```
copy(v.cbegin(), v.cend(), [](const auto& x) { std::cout << x; });
```

Iterator In / Lambda Out

```
int count = 0;

copy(v.cbegin(), v.cend(), [&](const auto&) { ++count; });
```

Iterator In / Lambda Out

```
vector<string> v;
vector<string> out;
copy(v.cbegin(), v.cend(), [&](const auto& x) { out.push_back(x); });
```

Iterator In / Lambda Out

```
vector<size_t> v{1,2,3,4};
vector<string> out;
copy(v.cbegin(), v.cend(), [&](size_t x) { out.emplace_back(x, '*'); });

// genera: {*, **, ***, ****}
```

Iterator In / Lambda Out

```
vector<size_t> v{1,2,3,4};
vector<string> out;
copy(v.cbegin(), v.cend(), [&](size_t x) { out.emplace_back(x, '*'); });

// per compilare gli esempi precedenti:
#define copy      for_each
```

Adapter

```
#include <utility>

template <typename T> class lambda_iterator {
    T func_;
public:
    lambda_iterator(T f) : func_(std::move(f)) {}

    struct proxy {
        T& f;

        template <typename X> proxy& operator=(X&& x) {
            f(std::forward<X>(x)); return *this;
        }
    };

    proxy operator* () { return proxy{ func_ }; }

    lambda_iterator& operator++() { return *this; }
    lambda_iterator& operator++(int) { return *this; }
};
```

Adapter

```
#include <vector>
#include <string>
#include <algorithm>
#include <iostream>

template <typename T>
inline lambda_iterator<T> lambda2iter(T f)
{
    return lambda_iterator<T>(f);
}

int main()
{
    std::vector<std::string> v1;
    std::copy(v1.cbegin(), v1.cend(),
              lambda2iter([](const std::string& s) { std::cout << s << std::endl; }));
}
```

Output secondario

- Molti algoritmi hanno un valore di ritorno principale e un output secondario

Output secondario / Logging

- Molti algoritmi hanno un valore di ritorno principale e un output secondario

```
// returns # of bytes copied
size_t copy_file(const char* src, const char* dest)
```

Output secondario / Logging

- Molti algoritmi hanno un valore di ritorno principale e un output secondario

```
template <typename logger_output_t>
size_t copy_file(const char* src, const char* dest, logger_output_t LOG)
{…}

size_t copy_file(const char* src, const char* dest)
{
   return copy_file(src, dest, [](size_t, size_t){});
}
```

Output secondario / Logging

```
template <typename logger_output_t>
size_t copy_file(const char* src, const char* dest, logger_output_t LOG)
{
   handle<file> in = …, out = …;
   const size_t n = GetFileSize(in);
   LOG(0, n);

   char buffer[SIZE];
   for (size_t i=0; i<n; ++i)
   {
       if (!ReadChunk(in, buffer, SIZE) || !WriteChunk(out, buffer, SIZE))
          throw …

       LOG(i += SIZE, n);
   }
   LOG(n, n);
   return n;
}
```

Output secondario / Logging

```
copy_file(“large.txt”, “new.txt”,
        [](size_t i, size_t n) { cout << (100.0*i)/n << ‘%’ << endl; });
```

Output secondario / Eccezioni

```
bool read_header(header_t& header, const size_t* raw, size_t raw_bytes)
{
        const size_t actual_length = (raw_bytes & BITS_TO_MASK(RevIdxFileVersion::FILESIZE_BITS));

        mxt_VERIFY_READ(4, size_t);

        header.version = memcpy_cast<RevIdxFileVersion>(traits::change_version<traits_t>(0)(raw[0]));
        header.N = raw[1];
        header.L = raw[2];
        header.M = raw[3]; // number of pairs

        if ((header.version.traits_id ^ traits_t::VERSION) & BITS_TO_MASK(RevIdxFileVersion::VERSION_BITS))
                return false;

        if (header.version.file_size > 0 && header.version.file_size > actual_length)
                return false;
```

Output secondario / Eccezioni

```
bool read_header(header_t& header, const size_t* raw, size_t raw_bytes)
{
        const size_t actual_length = (raw_bytes & BITS_TO_MASK(RevIdxFileVersion::FILESIZE_BITS));

        mxt_VERIFY_READ(4, size_t);

        header.version = memcpy_cast<RevIdxFileVersion>(traits::change_version<traits_t>(0)(raw[0]));
        header.N = raw[1];
        header.L = raw[2];
        header.M = raw[3]; // number of pairs

        if ((header.version.traits_id ^ traits_t::VERSION) & BITS_TO_MASK(RevIdxFileVersion::VERSION_BITS))
                return false;

        if (header.version.file_size > 0 && header.version.file_size > actual_length)
                return false;
```

Output secondario / Eccezioni

```
template <typename false_t>
bool read_header(header_t& header, const size_t* raw, size_t raw_bytes, false_t F)
{
        const size_t actual_length = (raw_bytes & BITS_TO_MASK(RevIdxFileVersion::FILESIZE_BITS));

        mxt_VERIFY_READ(4, size_t);

        header.version = memcpy_cast<RevIdxFileVersion>(traits::change_version<traits_t>(0)(raw[0]));
        header.N = raw[1];
        header.L = raw[2];
        header.M = raw[3]; // number of pairs

        if ((header.version.traits_id ^ traits_t::VERSION) & BITS_TO_MASK(RevIdxFileVersion::VERSION_BITS))
                return F("version mismatch");

        if (header.version.file_size > 0 && header.version.file_size > actual_length)
                return F("file size mismatch");
```

```
bool read_header(header_t& header, const size_t* raw, size_t raw_bytes)
{
        return read_header(header, raw, raw_bytes, [](const char*) { return false; });
}
```

Output secondario / Eccezioni

```
template <typename false_t>
bool read_header(header_t& header, const size_t* raw, size_t raw_bytes, false_t F)
{
        const size_t actual_length = (raw_bytes & BITS_TO_MASK(RevIdxFileVersion::FILESIZE_BITS));

        mxt_VERIFY_READ(4, size_t);

        header.version = memcpy_cast<RevIdxFileVersion>(traits::change_version<traits_t>(0)(raw[0]));
        header.N = raw[1];
        header.L = raw[2];
        header.M = raw[3]; // number of pairs

        if ((header.version.traits_id ^ traits_t::VERSION) & BITS_TO_MASK(RevIdxFileVersion::VERSION_BITS))
                return F("version mismatch");

        if (header.version.file_size > 0 && header.version.file_size > actual_length)
                return F("file size mismatch");
```

```
bool read_header(header_t& header, const size_t* raw, size_t raw_bytes)
{
        return read_header(header, raw, raw_bytes, [](const char* msg) -> bool { throw std::invalid_argument(msg); });
}
```

Warning

- attenzione a non esagerare con i template
  - Troppi argomenti di tipo *unconstrained-T* rendono il codice difficile da leggere

Warning

- attenzione a non esagerare con i template
  - Troppi argomenti di tipo *unconstrained-T* rendono il codice difficile da leggere

```
template <typename B1, typename B2, typename... B>
static void check_unary(B1 b1, B2 b2, B... b)
{
     static_assert(B1::hash != B2::hash, "invalid binding");
     check_unary(b1, b...);
     check_unary(b2, b...);
}

template <typename B1>
static void check_unary(B1) {}
```

Warning

- attenzione a non esagerare con i template
  - Troppi argomenti di tipo *unconstrained-T*
    - Rendono il codice difficile da leggere
    - Riducono i controlli del compilatore
    - Possono aumentare il tempo di compilazione
    - Vulnerabili a casi di abuso

Anti-hacking

```
template <typename T>
size_t for_each_line(const char* filename, T OUTPUT)
{

	std::string cache;
	size_t i=0
	for (std::ifstream inf(filename); std::getline(inf, cache); ++i)
		OUTPUT(cache);

	return i;
}



std::vector<std::string> lines;
for_each_line("file.txt", [&](const std::string& line) { lines.push_back(line); });
```

Anti-hacking

```
template <typename T>
size_t for_each_line(const char* filename, T OUTPUT)
{

	std::string cache;
	size_t i=0
	for (std::ifstream inf(filename); std::getline(inf, cache); ++i)
		OUTPUT(cache);

	return i;
}
```

```
std::vector<std::string> lines;
for_each_line("file.txt", [&](std::string&& line) { lines.push_back(std::move(line)); });
```

Anti-hacking

```
template <typename T>
size_t for_each_line(const char* filename, T OUTPUT)
{
	static_assert(std::is_invocable<T, const std::string&>::value, "wrong lambda");
	std::string cache;
	size_t i=0
	for (std::ifstream inf(filename); std::getline(inf, cache); ++i)
		OUTPUT(const_cast<const std::string&>(cache));

	return i;
}



std::vector<std::string> lines;
for_each_line("file.txt", [&](const std::string& line) { lines.push_back(line); });
```

Warning

- attenzione a non esagerare con i template
  - Troppi argomenti di tipo *unconstrained-T*
    - Rendono il codice difficile da leggere
    - Riducono i controlli del compilatore
    - Possono aumentare il tempo di compilazione
    - Vulnerabili a casi di abuso
  - Possibili rimedi:
    - static_assert(std::is_invocable<T, ARG1, ARG2>::value , "wrong lambda");
    - const_cast<const ARG&> esplicito sugli argomenti
    - riuso delle lambda mediante auto

Una piccola dose di TMP

- A volte è utile dedurre quali sono gli argomenti dell'oggetto lambda
- Caso semplice: controllare che sia corretta

```
static_assert(std::is_invocable<T, const std::string&>::value,
"wrong lambda");
```

Una piccola dose di TMP

- L'algoritmo potrebbe modificarsi a seconda di un argomento della lambda

```
// OUTPUT può ricevere "string" oppure "vector<string>"
// nel secondo caso, la riga viene tokenizzata usando WHITESPACE

    template <char... WHITESPACE, typename T>
    size_t for_each_line(const char* filename, T OUTPUT)
    {
```

Una piccola dose di TMP

```
template <size_t N, typename T>
struct nth_argument;

// se T è un funtore che prende (A1,A2…)
// nth_argument<0, T>::type == A1
// nth_argument<1, T>::type == A2
// ecc.
```

Una piccola dose di TMP

```
template <size_t N, typename T>
struct nth_argument;

template <size_t N, typename... A>
struct helper;

template <size_t N, typename R, typename... A>
struct nth_argument<N, R(*)(A...)> : helper<N, A...> {};

template <size_t N, typename R, typename T, typename... A>
struct nth_argument<N, R(T::*)(A...) const> : helper<N, A...> {};

template <size_t N, typename T>
struct nth_argument : nth_argument<N, decltype(&T::operator())> {};
```

Una piccola dose di TMP

```
template <size_t N, typename... T>
struct helper;

template <size_t N, typename T1, typename... T>
struct helper<N, T1, T...> : helper<N-1, T...> {};

template <typename T1, typename... T>
struct helper<0, T1, T...>
{
	using type = std::remove_cv_t< std::remove_reference_t<T1> >;
};

template <>
struct helper<0>
{
	using type = void;
};
```

Una piccola dose di TMP

```
template <typename T>
void invoke(T f, string& s, vector<string>&  , true_type) { f(s); }

template <typename T>
void invoke(T f, string&  , vector<string>& v, false_type) { f(v); }

template <char... WHITESPACE, typename T>
size_t for_every_line(const char* filename, T OUTPUT)
{
   static_assert(std::is_same<void, typename nth_argument<1, T>::type>::value,
"invalid callback");
   using arg_t = typename nth_argument<0, T>::type;
   string s; vector v;

   if (std::is_same<std::string, arg_t>::value)
   { s = …; }
   else
   { v = …; }

   invoke(OUTPUT, s, v, std::is_same<std::string, arg_t>{});
```

```
for_each(questions.begin(), questions.end(),
[](question& q) { reply_to(q); });
```