Functional Programming in modern C++

Michel Steuwer

This program is still valid C++ …

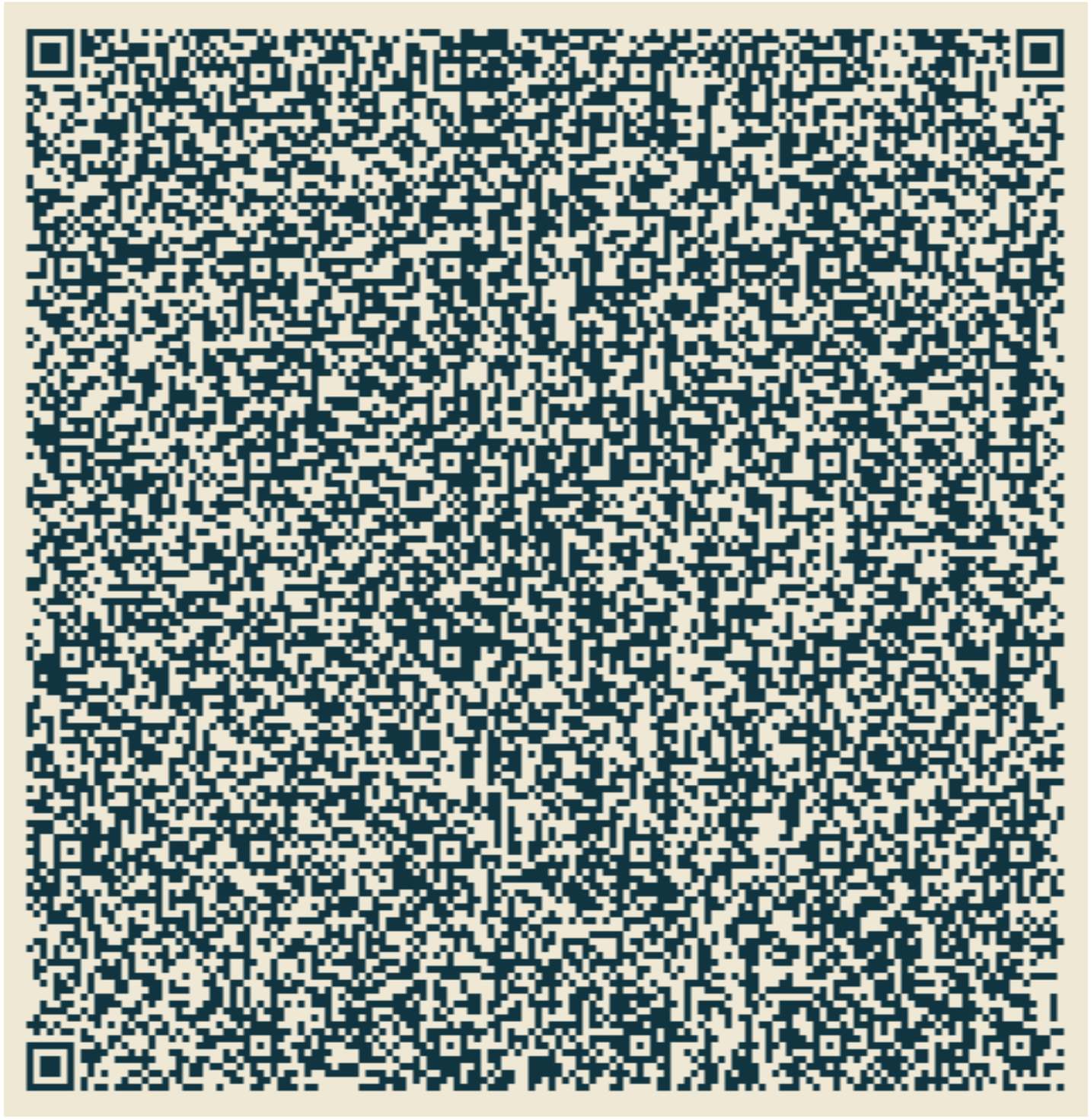
```
#include<stdio.h>/*IOCCC2014           2014    2014    IOCCC2014IOCCC2014IOCCC201*/
char*s="\"nsu{AntynCnuq}Bnu{           sEot    ln>b    )+c^g+@`+]_osk{;j@bkg&c<'^o\
r'Q]                    bh'l    vQ^k                    g&c:                    %n|\
N]_o                    ptj9    lwg+                    )d:b                    kg$\
c8#^    #g+)d8`a%g+)    d8`_                g&;bh'oq    Q^)g    +&kcNlyMc+)d    8`a\
`g+@    u)|d8ak=bl}(    Q^og                (O{MK6lM    L(rR    pOpM866OsRlm    N(q\
Q]##    OsR#M_(lQoOa    N9$m    vOwwRor~        }(cN    mkM:    q(Q]%_(uU]}_    {8b\
%mRh    #S^`#mOaaD%/    RI4$    4SNH$%N4        RlMG    /2MJ    24O3NF(tQ?1l    N*N\
+Q]l    mq9l8b$^#h$#    .d,d    xv#mSOPm    R8`/        lM;b    h&^/lM:k8b%&    Q^-\
h%c9    .#,/&N$McPc%    -d8,    $c8``7:b    %^h%        :79b    $^%$7ON8r%Qr    h$Q\
On%q                    N%O~        M$qN$OMp    RmPQ    O%rQ                    $rQ\
MkQN                    77O#        dj#Nkd$7    O%d8    ``(r                    RmM\
:(luN](%mRMmO%lNRmO_loa8%%O<    b(lQ    h'lQ    O^ln    ;b'%N&O^6sN#M9slltwzh&TmS$'\
%R#ON$oNS&pRM7O$'%S#ON$oNS&p    SMd9    ``(t    pkO9    $:nk8b#^%}kx#O%lMtpk#OQOP:#\
                                mR8`}(qQ        O<b\
                                '7M{N^kh        'Q]\
    7sN'    M8q;k'N_7vN'nSM8    'nR;    $%M'    TamS&7O#d#7O    &d8`z9b$    ^h$\
    rQ]6    yN$_$$pQaM8m{px$    TmSt    6O#N    sM$nRmUd#7Os    d77sNOsd    8`j\
=nk;        |8k9    b1g(    Q^)$c:%]    k*Q]g$9%_h*]+wnS        _+sTaa:&%{R\
M<%{        S{O_    &<#w    kP8#a;'(    0d:`184>b)^15)OM        :23h)QN<'/M\
    =b(^'h(Q]0h#    c_kah%0d9`b'/O(Q^(/SlM90%M(/            RM;&    $c&ckQ,    -$c\
    .&kccMcOP$&d    9g(=``6:69j=b5g(Q^3(2lNUO<b4            'Q^s    ;b&^%%pQ    M8k\
        $7O#cQ]0(2(Q]'2O    3'Q]'_3a    N_1'    5OMaMch&T$mR#nRMPmSk    U$7\
        O#d8_a%%mS]l_%mR    mS]$]h$9    j_la    _$6N]g$9j_laaaN:`g'<    ``7\
oM:6%N9b%g$Q        ^6%N8b%g    #Q^l            peoqemtemw#jQ7#QO$jQ    O7$Q    Ok#\
$7OMQ]k_#$7O        ckQaOrON    epne            luelue`lpeoqempepneu    e`kf    _a`\
                                lf))    **,**,*)-)/),0).0(6/2+667,(&    $##\
                                #;=@    D*0;#include<stdio.h>@intYH[    9`%\
.],*s,*c,d,t;`;'main(`-9n,ch            ar*v    []){        for(s=c=    H+3`XI;\
(*++s=Q[d++]););for(;n>1&&(*            ++s=    v[1]        [tgANs=H    ;d=*c++\
%93,                    d-9;    ){in    tYv=*s,g        []={    n+v,        v-n,n*_\
,#^_                    ,,<v    ,n?v    /n:0`,#%        `,,v    >>n,        v==n}ay\
>t=0    ;d<4&&d>=2*!    !n&&    (c-=d/3*2_KK3+*c++,t||v!=98+d);)        t+=v++/6-16\
?0:v    /2%3-1cq*d-1    4;t>    0_t$<3_z(105<*c_X'=t*21aq@-106;n        =d>76?s--,g\
[d-7    7]:d>55?H_1)    21]=    n,*_@_8(        9?*++s_4&12>d_C`f3]+=21-d*2\
:d<3    4?t:_Zadat57    <d?p    utchar(n        ),v:6<a](g+99]=a{d2#:`xa6,n\
;}re    turnY_.",*p,    b,d[        9338    ],*q,x,*r=d;            int main(){;for(
p=q=    5000+0+d;*s;    s++)        if(*    s>32)*p++=*s            -89?*s:32;for(p=
1152                    +q;(    b=*p++);){for(d[17]=            10;x    =*p++,b<
92&&                    34<b    --;*r++=x)if(x==9*9)            for(    ;*q;x=34
)*r++=*q++;for(p-=b<92;b-->4        *23;r++)    *r=r[36-    x];}puts    (d);
return(0);}/*IOCCC2014IOCCC2        014IOCCC    2014IOCC    C2014IOC    CC*/
```

Won the *Most square (YODA award)* at *The International Obfuscated C Code Contest 2014* by *Yusuke Endoh*

```
#include<stdio.h>
int H[99999],*s,*c,d,t;int main(int n,char*v[]){for(s=c=H+39999;(*++s="nsu{AntyncnuqjBnu{sEotln>b)
+c^g+@`+]_osk{;j@bk&&c<'^or'Q]bh'lvQ^k&&c:%n|N]_opt}9lwg+)d:bk&$c8#^#g+)d8`a%g+)d8`_&&;bh'oqQ^)g
+&kcNlyMc+)d8`a`g+@u)|d8ak=bl}(Q^og(O{MK6lML(rRpOpM866OsRlMN(qQ]##0sR#M_(lQooAN9$mvOwwRor~}
(cNmkM:q(Q]%_(uU]}_{8b%mRh#S^`#mOaaD%/RI4$4SNH$%N4RlMG/2MJ24O3NF(tQ?1lN*N+Q]lmq9l8b$^#h
$#.d,dxv#mSOPmR8`/lM;bh&^k8b%&Q^-h%c9.#,/&N$McPc%-d8,$c8`-7:b%^h%:79b$^%$7ON8r%Qrh$QOn%qN%O~M$qN
$OMpRmPQO%rQ$rQMkQN77O#dj#Nkd$7O%d8`-(rRmM:(luN](%mRMmO%lNRmO_loa8%
%O<b(lQh'lQO^ln;b'%N&O^6sN#M9slltwzh&TmS$'%R#ON$ONS&pRM7O$'%S#ON$ONS&pSMd9``(tpkO9$:nk8b#^%}lx#O
%lMtpk#OQOP:#mR8`}(qQO<b'7M{N^kh'Q]7sN'M8q;k'N_7vN'nSM8'nR;$%M'TamS&7O#d#7O&d8`z9b$^h$rQ]6yN$_$
$pQaM8m{px$TmSt6O#NsM$nRmUd#7Osd77sNOsd8`j=nk;|8k9b1g(Q^)$c:%]k*Q]g$9%_h*]+wnS_+sTaa:&%{RM<%
{S{O_&<#wkP8#a;'(0d:`184>b)^15)OM:23h)QN<'/M=b(^'h(Q]0h#c_kah%0d9`b'/O(Q^(/SlM90%M(/RM;&$c&kckQ,-
$c.&kccMcOP$&d9g(=``6:69j=b5g(Q^3(2lNUO<b4'Q^s;b&^%%pQM8k$7O#cQ]0(2(Q]'203'Q]'_3aN_1'5OMaMch&T
$mR#nRMPmSkU$7O#d8_a%%mS]l_%mRmS]$]h$9j_la_$6N]g$9j_laaaN:`g'<``7oM:6%N9b%g$Q^6%N8b
%g#Q^lpeoemtemw#jQ7#QO$jQO7$QOk#$7OMQ]k_#$7OckQaOrONepnelueltpeoempepneue`kf_a`lf))*
+,**,*)-)/),0).0(6/2+667,(&$###;=@D*0;#includeestdio.h>@int H[9`%.],*s,*c,d,t;`;'main(`-9n,char*v[])
{for(s=c=H+3`XI;(*++s=Q[d++]);)for(;n>1&&(*++s=v[1][tgANs=H;d=*c++%93,d-9;){int v=*s,g[]=[n+v,v-
n,n*_,#^_,,<v,n?v/n:0`,#%`,,v>>n,v==n}ay>t=0;d<4&&d=>2*i|n&&(c-=d/3*2_kK3+*c++,t||v|=98+d);)t+=v++/
6-16?0:v/2%3-1cq*d-14;t>0_t$<3_z(105<*c_X'=t*21aq@-106;n=d>76?s--,g[d-77]:d>55?H_1)21]=n,*_@_8(9?*+
+s_4&12>d_C`f3]+=21-d*2:d<34?t:_Zadat57<d?putchar(n),v:6<a][g+99]=a{d2#:`xa6,n;}return _.'"[d+
+])};for(;n>1&&(*++s=v[1][t++])};for(s=H;d=*c++%93,d-9;){int v=*s,g[]=[n+v,v-n,n*v,v^n,n<v,n?v/n:
0,n?v%n:0,v>>n,v==n};for(t=0;d<4&&d>=2*i|n&&(c-=d/3*2,v=3+*c++,t||v|=98+d);)t+=v++/6-16?0:v/
2%3-1;for(t=d-14;t>0&&d<34&&105<*c;)t=t*21+*c++-106;n=d>76?s--,g[d-77]:d>55?H[d-21]=n,*s--:d>9?*+
+s=n,12>d?H[*c++]+=21-d*2:d<34?t:H[d]:d>5?s--,7<d?putchar(n),v:6<d?H[*g+99]=*s--:H[*g+99]:2>d?
*s--:n;}return n;}
```

… which generates this valid C++ program …

… which prints this QR code on the terminal.



Functional Programming in Modern C++

- Type inference with **`auto`**
- Lambda expressions
- Specifying type constrains with *Concepts lite*

auto

`auto` in one Slide

Use:

```
auto iter = vec.begin();
```

instead of:

```
std::vector<int>::iterator iter = vec.begin();
```

> **Guideline:** Remember that preferring **auto** variables is motivated primarily by correctness, performance, maintainability, and robustness—and only lastly about typing convenience.

AAA (Almost Always Auto) Style by *Herb Sutter:*

`auto` type inference

Variables declared with **`auto`** are *independent values*.

Therefore:

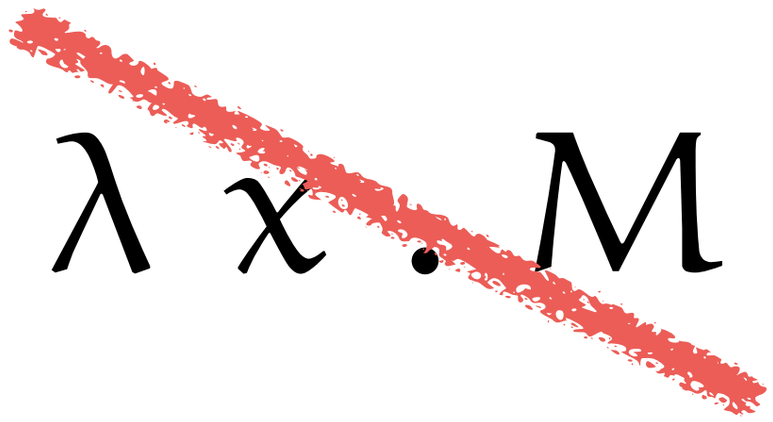
1. *references* are treated as *non-references*:

```
int& foo();
auto v = foo();   // type of v == int
auto& vr = foo(); // type of vr == int&
```

2. *const/volatile* are treated as *non-const/non-volatile:*

```
const int bar();
auto v = foo();          // type of v == int
const auto cv = foo(); // type of cv == const int
```

$$\lambda x . M$$


λ x . M

[ ] ( ) { }

[] () {} in one Slide

```
int numbers_in_range(const vector<int>& numbers,
                     int rangeBegin, int rangeEnd)
{
  return count_if(begin(numbers), end(numbers),
    [rangeBegin, rangeEnd](int n) {
      return n >= rangeBegin && n <= rangeEnd;
    } );
}

int main() {
  auto xs = vector<int>{4, 8, 15, 16, 23, 42};

  cout << numbers_in_range(xs, 15, 42); // prints out 4
}
```

[] () {} in one Slide

```
int numbers_in_range(const vector<int>& numbers,
                     int rangeBegin, int rangeEnd)
{
  return count_if(begin(numbers), end(numbers),
    [rangeBegin, rangeEnd](int n) {
      return n >= rangeBegin && n <= rangeEnd;
    } );
}

int main() {
  auto xs = vector<int>{4, 8, 15, 16, 23, 42};

  cout << numbers_in_range(xs, 15, 42); // prints out 4
}
```

Capture List

Parameter

Body

[] () {} in one Slide

```
int numbers_in_range(const vector<int>& numbers,
                     int rangeBegin, int rangeEnd)
{
  return count_if(begin(numbers), end(numbers),
    [rangeBegin, rangeEnd](int n) mutable -> bool {
      return n >= rangeBegin && n <= rangeEnd;
    } );
}

int main() {
  auto xs = vector<int>{4, 8, 15, 16, 23, 42};

  cout << numbers_in_range(xs, 15, 42); // prints out 4
}
```

Capture List

Parameter

Indicate mutation of captured values (optional)

Return Type (optional)

Body

Implementation

For every lambda expression …

```
count_if(…, [rangeBegin, rangeEnd](int n) {
  return n >= rangeBegin && n <= rangeEnd;
});
```

… the compiler generates a corresponding class:

```
class Lambda {
  const int rangeBegin;
  const int rangeEnd;
public:
 Lambda(int rangeBegin, int rangeEnd)
  : rangeBegin(rangeBegin), rangeEnd(rangeEnd) {};

 bool operator()(int n) {
  return n >= rangeBegin && n <= rangeEnd;
 }
};

count_if(…, Lambda(rangeBegin, rangeEnd));
```

Capturing

Variables can be captured *by-value*, by copying:

```
auto makeCounter(int x) {
  return [x] mutable { return x++; };
}
```

by moving (transferring ownership):

```
auto w = std::make_unique<Widget>();
auto l = [w = std::move(w)]() { w->doStuff(); };
```

or *by-reference:*

```
auto data = makeExpesiveToCopyData();
auto t = std::thread([&data] { data.modify(); });
// do other stuff (carefully not accessing data)
t.join();
```

requires

concept

What is the problem?

Generic programming with templates:

```
template <typename Iter, typename T>
auto find(Iter first, Iter last, const T& value) {
 for(; first != last; ++first) {
   if (*first == value) { return first; } }
 return last;
}
```

What happens if values of type `T` are not *comparable*?
Or if values of type `Iter` are not *iterators*?

What is the problem?

Generic programming with templates:

```
template <typename Iter, typename T>
auto find(Iter first, Iter last, const T& value) {
 for(; first != last; ++first) {
   if (*first == value) { return first; } }
 return last;
}
```

What happens if values of type `T` are not *comparable*?
Or if values of type `Iter` are not *iterators*?

Static type safety is maintained, but error messages are poor.

Specifying Constraints

Specifying constraints on types using a **`requires`** clause:

```
template <typename T>
  requires Integral<T>
T gcd(T a, T b) { … }

template <typename K, typename V>
  requires Hashable<K> && Copyable<V>
class HashMap { … }
```

Constrains are checked when templates are instantiated

Implementation is *not* checked against the constrains!

For more details see *Generic Programming with Concepts Lite* by *Andrew Sutton*

Specifying Concepts

Concepts give names to a set of constrains:

```
template <typename T>
concept bool Hashable = requires (T object) {
  { object.computeHash() } -> int;
};
```

Concepts introduce a syntactic requirements to model a semantic *concept*

Examples

```
template <typename T>
concept bool Range = requires (T range) {
  typename Iterator_type<T>;
  { std::begin(range) } -> Iterator_type<T>;
  { std::end(range)   } -> Iterator_type<T>;
};
```

```
template <typename T>
concept bool Equality = requires(T a, T b) {
  { a == b } -> bool;
  { a != b } -> bool;
};


template <typename F, typename… Args>
concept bool Function = requires(F f, Args… args) {
  { f(args…); };
};
```