C++14'S Relaxing Requirements for constexpr Functions

Michelle Bray, Callan Fisher, Marc Simpson

December 2014

From C++11…

- Constant expression functions restrictive
- Executed at compile time
- Only contain a single expression

. . . To C++14

- Relaxes restrictions for constexpr
- Allows variable mutation
- Additional restrictions for static and `local_thread` variables
- Still executed at compile time

Bjarne Stroustrup

- Major player in creation of C++

"C++14 is simply the completion of the work that became C++11"

Relaxed Restriction

1. declaring a variable that is not static or `local_thread`
2. the ability to use if (else/ if else) and switch
3. the use of loops (for/ ranged-for, do/ do-while)
4. objects whose lifetime began within the constexpr evaluation can mutate

Relaxed Restriction (2)

1. declaring a variable that is not static or `local_thread`
2. the ability to use if (else/ if else) and switch
3. the use of loops (for/ ranged-for, do/ do-while)
4. objects whose lifetime began within the constexpr evaluation can mutate

Old Definition of constexpr Function

Must contain only:

- Null statements
- `static_assert-declarations`
- Typedef and alias declarations that do not define:
    - Classes
    - Enumerations
- using-declarations
- using-directives
- One return statement

Revised Definition of constexpr Function

- Not Virtual
- Return type is of literal type
- Function body cannot contain:
  - asm-definition
  - goto

Example Code

```
constexpr int prev(int x) {
    return --x;
}

// C++14 OK, C++11 error: use of increment
```

```
constexpr int g(int x, int n) {
    int r = 1;
    while (--n > 0) r *= x;
    return r;
}

// C++14 OK, C++11 error: body not just "return expr"
```

constexpr and Multiple Variables

- Handle multiple variables
- Object mutation

Object Mutation

- Change objects within constant expressions
- Occurs until end of evaluation or lifetime of object

constexpr Function Definitions

“A literal constant expression is a prvalue core constant expression of literal type, but not pointer type (after conversions as required by the context).” – Richard Smith

- Literal, reference, and address constant expressions unified under constant expressions

Continued

C++14:

- A reference constant expression is either a glvalue or a prvalue
- An address constant expression is a prvalue of type `std::nullptr_t` or of pointer type

Static Local Variables

- Relaxing rules for constexpr leads to increased restrictions for `static_local` variables
- Prevents side effects
- Additional restrictions to ensure evaluation runs correctly

Example Code

```
constexpr int first_val ( int n ) {
    static int value = n;
    return value;
}

const int N = first_val(42);
int arr[first_val(422)];

// error: not a constant expression
```

Example Code (2)

```
constexpr int first_val ( int n ) {
    static int value = n;
    return value;
}

const int N = first_val(42);
int arr[first_val(422)];

// error: not a constant expression
```

Novice Friendly

"I hope that the tide has turned so that C++ is becoming more novice friendly." – Bjarne Stroustrup

- Less restrictions = more intuitive for beginners

Personal Opinion

- Beneficial update
- Simplifies language
- Natural combination and flow

Future of C++

- Less error prone
- Easier for beginners to learn
- **What do you think?**

Sources

- http://meetingcpp.com/index.php/br/items/looking-at-c14.html
- http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3597.html
- http://www.infoq.com/news/2014/08/cpp14-here-features
- http://electronicdesign.com/dev-tools/bjarne-stroustrup-talks-about-c14
- http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3652.html