Introduction to constexpr

Antal A. Buss

YEG C++ Meetup

February 11 2020

Outline

1. Overview
   - Constants
   - Template Metaprogramming

2. constexpr
   - Evolution
   - Metaprogramming with **constexpr**
   - Replacing `#ifdef` with **constexpr**

Constant values

- *Constants* refer to fixed values that the program may not alter.

Constant values

- *Constants* refer to fixed values that the program may not alter.

- Constants are treated just like regular variables except that their values cannot be modified after their definition.

Constant values

- *Constants* refer to fixed values that the program may not alter.
- Constants are treated just like regular variables except that their values cannot be modified after their definition.
- There are two simple ways in C++ to define constants:
  - Using **#define** preprocessor.
    #define SIZE 10
  - Using **const** keyword.
    const int size = 10;

`const` keyword

- **const int** size = 10;

const keyword

- **const int** size = 10;
- Pointer to a constant integer

```
const int* p;
int const* p;
```

const keyword

- `const int size = 10;`
- Pointer to a constant integer

  ```
  const int* p;
  int const* p;
  ```

- Constant pointer to an integer

  ```
  int* const q;
  ```

`const` keyword

- `const int size = 10;`

- Pointer to a constant integer

```
const int* p;
int const* p;
```

- Constant pointer to an integer

```
int* const q;
```

- Constant pointer to a const integer

```
const int* const r;
```

const keyword

```
template<typename T>
struct Foo {
  T value;

  T bar1(T& x) const { return value; }
  T bar2(const T& x) const { return x; }
  const T& bar3(T x) const { return value; }
  T bar4(const T& x) { return x; }
};
```

const keyword

```
template<typename T>
struct Foo {
  T value;

  T bar1(T& x) const { return value; }
  T bar2(const T& x) const { return x; }
  const T& bar3(T x) const { return value; }
  T bar4(const T& x) { return x; }
};
```

```
    int v = 8;
    const Foo<int> a{5};

    int b1 = a.bar1(v);
    int b2 = a.bar2(v);
    int b3 = a.bar3(v);
    int b4 = a.bar4(v);
```

const keyword

```
template<typename T>
struct Foo {
  T value;

  T bar1(T& x) const { return value; }
  T bar2(const T& x) const { return x; }
  const T& bar3(T x) const { return value; }
  T bar4(const T& x) { return x; }
};
```

```
    int v = 8;
    const Foo<int> a{5};

    int b1 = a.bar1(v);
    int b2 = a.bar2(v);
    int b3 = a.bar3(v);
    int b4 = a.bar4(v);   // <--- Error
```

Template Metaprogramming (TMP)

Metaprogramming is a technique in which templates are used by a compiler to generate temporary source code, which is merged by the compiler with the rest of the source code and then compiled.

The output of these templates include compile-time constants, data structures, and complete functions.

[0] https://en.wikipedia.org/wiki/Template_metaprogramming

Template Metaprogramming (TMP)

```
template <int N>
struct fibo
{ enum { value = fibo<N-1>::value + fibo<N-2>::value }; };

template <>
struct fibo<0>
{ enum { value = 1 }; };

template <>
struct fibo<1>
{ enum { value = 1 }; };

int main() {
  std::cout << "fibo(40): " << fibo<40>::value << std::endl;
  return 0;
}
```

constexpr keyword

- Introduced in C++11 and improved in C++14 and C+17.

constexpr keyword

- Introduced in C++11 and improved in C++14 and C+17.
- It means constant expression. Like **const**, it can be applied to variables.

constexpr keyword

- Introduced in C++11 and improved in C++14 and C+17.
- It means constant expression. Like **const**, it can be applied to variables.
- Unlike const, **constexpr** can also be applied to functions and class constructors. constexpr indicates that the value, or return value, is constant and, where possible, is computed at compile time.

constexpr keyword

- Introduced in C++11 and improved in C++14 and C+17.
- It means constant expression. Like **const**, it can be applied to variables.
- Unlike const, **constexpr** can also be applied to functions and class constructors. constexpr indicates that the value, or return value, is constant and, where possible, is computed at compile time.
- Computing at compile time instead of run time, helps your program run faster and use less memory.

const vs constexpr

```
int main(int argc, const char** argv) {
  constexpr int a = 10;
  const int b = 10;

  const int c = 10 + a;
  const int d = 10 + b;

  constexpr int e = 10 + a;
  constexpr int f = 10 + b;
}
```

Generated code

Evolution of `constexpr`

C++11 One (**return**) expression per function was allowed, loops using recursion, restricted branch control flow, math functions, etc

Evolution of `constexpr`

C++11 One (**return**) expression per function was allowed, loops using recursion, restricted branch control flow, math functions, etc

C++14 Generalized constexpr, use of constexpr in libraries

Evolution of `constexpr`

C++11 One (**return**) expression per function was allowed, loops using recursion, restricted branch control flow, math functions, etc

C++14 Generalized constexpr, use of constexpr in libraries

C++17 **if constexpr** for metaprogramming, **constexpr** lambdas, STL

constexpr example (C++11)

```
#include <iostream>

constexpr long fibo(int n) {
  return (n==0 || n==1) ? 1 : fibo(n-1)+fibo(n-2);
}

int main()
{
  const long res = fibo(42);
  std :: cout << ”fibo(42): ” << res << std::endl;
  return 0;
}
```

Generated code

`constexpr` example (C++14)

```
#include <iostream>

constexpr long fibo(int n) {
  if (n==0 || n==1)
    return 1;
  else
    return fibo(n-1)+fibo(n-2);
}

int main()
{
  long res = fibo(42);
  std::cout << "fibo(42): " << res << std::endl;
  return 0;
}
```

Generated code

constexpr example (C++17)

```
constexpr int N = 50;
using fibo_tbl_t = std::array<long,N>;

constexpr fibo_tbl_t gen_fibo_tbl() {
    fibo_tbl_t res = {};
   res[0] = 1; res[1] = 1;
   for(int i=2; i<N; ++i)
      res[i] = res[i-1] + res[i-2];
   return res;
}

int main() {
    fibo_tbl_t fibo_tbl = gen_fibo_tbl();
   std::cout << "fibo(42): " << fibo_tbl[42] << std::endl;
   return 0;
}
```

Generated code

Metaprogramming with `constexpr`

Before C++17, static if (if that works at compile time) was implemented using tag dispatching or SFINAE[1] (e.g., via std :: enable_if).

[1] Substitution Failure Is Not An Error

Metaprogramming with constexpr

Before C++17, static if (if that works at compile time) was implemented using tag dispatching or SFINAE[1] (e.g., via std :: enable_if).

```
struct Test { typedef int foo; };

template <typename T>
void f(typename T::foo) {}    // Definition #1

template <typename T>
void f(T) {}                  // Definition #2

int main() {
  f<Test>(10); // Call #1.
  f<int>(10);  // Call #2. Without error thanks to SFINAE.
}
```

[1] Substitution Failure Is Not An Error

Metaprogramming with `constexpr`

Example

```
template <typename T>
std::string str(T t) {
    if (std::is_convertible_v<T, std::string>)
        return t;
    else
        return std::to_string(t);   // Error, to_string is not
                                    // defined over std::string
}

int main() {
  std::string val = "10";
  auto t = str(val);
  std::cout << t+"!" << std::endl;
  return 0;
}
```

Metaprogramming (before C++17)

Example

```
template <typename T>
std::enable_if_t<std::is_convertible_v<T, std::string>, std::string>
str(T t) {
    return t;
}

template <typename T>
std::enable_if_t<!std::is_convertible_v<T, std::string>, std::string>
str(T t) {
    return std::to_string(t);
}

int main() {
  std::string val = "10";
  auto t = str(val);
  std::cout << t+"!" << std::endl;
}
```

Metaprogramming with `constexpr` (after C++17)

Example

```
template <typename T>
std::string str(T t) {
    if constexpr (std::is_convertible_v<T, std::string>)
        return t;
    else
        return std::to_string(t);
}

int main() {
  std::string val = "10";
  auto t = str(val);
  std::cout << t+"!" << std::endl;
  return 0;
}
```

Replacing #ifdef with constexpr

```
void do_something() {
    //do something general

    #ifdef __linux__
    //do something Linux
    #elif __APPLE__
    //do something Apple
    #elif __WIN32
    //do something Windows
    #endif

    //do something general
}
```

Replacing #ifdef with constexpr (C++17)

```
enum class OS { Linux, Mac, Windows };

//Translate the macros to C++ at a single point in the application
#ifdef __linux__
constexpr OS the_os = OS::Linux;
#elif __APPLE__
constexpr OS the_os = OS::Mac;
#elif __WIN32
constexpr OS the_os = OS::Windows;
#endif
```

Replacing #ifdef with constexpr (C++17)

```
void do_something() {
    //do something general

    if constexpr (the_os == OS::Linux) {
        //do something Linuxy
    }
    else if constexpr (the_os == OS::Mac) {
        //do something Appley
    }
    else if constexpr (the_os == OS::Windows) {
        //do something Windowsy
    }

    //do something general
}
```

Introduction to constexpr

Antal A. Buss

YEG C++ Meetup

February 11 2020