CS100 Lecture 28

Compile-time Computations and Metaprogramming

Contents

Example: Binary literals

Binary literals

Built-in binary literals support: C++14 and C23.

```
switch (rv32inst.opcode) { // 32-bit RISC-V instruction opcode
  case 0b0110011: /* R-format */
  case 0b0010011: /* I-format (not load) */
  case 0b0000011: /* I-format: load */
  case 0b0100011: /* S-format */
  // ...
}
```

How do people write binary literals when there is no built-in support?

Runtime solution? No!

This is not satisfactory: The value is computed at run-time!

```
int dec2bin(int x) {
  int result = 0, pow_two = 1;
  while (x > 0) {
    result += (x % 10) * pow_two;
    x /= 10;
    pow_two *= 2;
  }
  return result;
}

void foo(const RV32Inst &inst) {
  const int forty_two = dec2bin(101010); // correct, but slow.
  switch (inst.opcode) {
    case dec2bin(110011): // Error! 'case' label must be compile-time constant!
        // ...
  }
}
```

Preprocessor metaprogramming solution

`#` and `##` operators: Both are used in function-like macros.

`#x`: stringify `x`.

```cpp
#define SHOW_VALUE(x) std::cout << #x << " == " << x

int ival = 42;
SHOW_VALUE(ival); // std::cout << "ival" << " == " << ival;
```

`a##b`: Concatenate `a` and `b`.

```cpp
#define DECLARE_HANDLER(name) void handler_##name(int err_code)

DECLARE_HANDLER(overflow); // void handler_overflow(int err_code);
```

Preprocessor metaprogramming solution

https://stackoverflow.com/a/68931730/8395081

```
#define BX_0000 0
#define BX_0001 1
#define BX_0010 2
// ......
#define BX_1110 E
#define BX_1111 F

#define BIN_A(x) BX_##x

#define BIN_B(x, y) 0x##x##y
#define BIN_C(x, y) BIN_B(x, y)

#define BIN(x, y) BIN_C(BIN_A(x), BIN_A(y))

const int x = BIN(0010, 1010); // 0x##BX_0010##BX_1010 ==> 0x2a ==> 42
```

Template metaprogramming solution

```cpp
template <unsigned N> struct Binary {
  static const unsigned value = Binary<N / 10>::value * 2 + (N % 10);
};
template <> struct Binary<0u> {
  static const unsigned value = 0;
};

void foo(const RV32Inst &inst) {
  const auto x = Binary<101010>::value; // 42
  switch (inst.opcode) {
    case Binary<110011>::value: // OK.
      // ...
  }
}
```

Compared to preprocessor metaprogramming, template metaprogramming is more powerful, and less error-prone.

Modern C++: `constexpr` function

In modern C++, just mark the function `constexpr`, and the compiler will be able to execute it!

```cpp
constexpr int dec2bin(int x) {
  int result = 0, pow_two = 1;
  while (x > 0) {
    result += (x % 10) * pow_two; x /= 10; pow_two *= 2;
  }
  return result;
}
void foo(const RV32Inst &inst) {
  switch (inst.opcode) {
    case dec2bin(101010): // OK. Since 101010 is a compile-time constant,
                          // the function is executed at compile-time and
                          // produces a compile-time constant.
      // ...
  }
}
```

Metaprogramming

Metaprogramming is a programming technique in which computer programs have the ability to **treat other programs as their data**.

- Read, generate, analyze or transform other programs, and even itself.

Typical problems that needs metaprogramming

Write a function that selects different behaviors **at compile-time** according to the argument?

- e.g. The `std::distance` function (in this week's recitation).

Generate some code according to the members of my class, without too much manual modification?

- e.g. Serialization: Generate `operator>>` automatically for my class that prints the members one-by-one?
- e.g. "Metaclasses": Generate the getters and setters automatically for each of my data members?

......

Compile-time computations

How much work can be done in compile-time?

- Call to numeric functions with compile-time known arguments?
  - e.g. Can `std::acos(-1)` be computed in compile-time?
- Manipulation of compile-time known strings?
  - e.g. Preparation of compile-time known regular expressions: CTRE
- Even crazier: Compile-time raytracer?!
  - The computations are done entirely in compile-time. At run-time, the only work is to output the image.

Anything can be computed in compile-time, provided that the arguments are compile-time known!

`constexpr` functions