ECE326

PROGRAMMING LANGUAGES

Lecture 22 : Constant Expression

Kuei (Jack) Sun

ECE

University of Toronto

Fall 2019

Constant Expression

- Can be evaluated at compile time

```
const int a = 5 + 7;
// compiler would generate a = 12 directly
```

- constexpr keyword
  - Declares a compile-time variable, function, or class
    - May not exist at runtime (unlike constant variables)
  - Variable
    - Can only be assigned constant expression
  - Function
    - Arguments must only be constant expression

Constexpr

```
constexpr int a[] = { 1, 2, 3, 4 };
constexpr int sum(const int a [], unsigned n) {
      return (n == 0) ? 0 : a[0] + sum(a+1, n-1);
}

// a good compiler should generate x = 10 directly
int x = sum(a, sizeof(a)/sizeof(int));

template<int X>          // template argument only accepts
void print_const() {     // compile-time constant values
      cout << X << endl;
}

print_const<sum(a, 3)>();      // OK, prints 6
print_const<sum(a, 5)>();      // FAIL
error: array subscript value is outside the bounds of array
```

Constexpr Function

- Tells compiler to evaluate function at compile time
- Can significantly increase compile time
  - Compiler must ensure computation cannot crash itself
    - Performs extensive type-checking
    - E.g. Array out of bound check
- C++11
  - Restrictive on what's allowed in a constexpr function
    - No loops – must rely on recursion
    - Exactly one return statement allowed in body
    - No local variables, arguments only

Constexpr Examples

```
constexpr int factorial(int n) {
    return n <= 1 ? 1 : (n * factorial(n - 1));
}

/* lexicographical comparison of two constant strings */
/* returns positive if a > b, negative if a < b, 0 if equal */
constexpr int
constcmp(const char * a, const char * b)
{
    return (a[0] == '\0') ? (a[0] - b[0]) : (
        (a[0] == b[0]) ? constcmp(a+1, b+1) : a[0] - b[0]
    );
}

constcmp("he", "hello")       // -108
constcmp("hello", "hell“)     // 111  (ASCII for o)
```

Constexpr Function

- Depends on compiler implementation
  - May or may not be turned into a runtime function
- Depends on argument

```
print_const<constcmp("hello", argv[0])>();
error: ‘argv’ is not a constant expression

// function is also used at runtime
cout << constcmp("hello", argv[0]) << endl;     // 58
```

- Upgrade to C++14
  - Allows loops and local variables!

Compile-Time Function

- Useful for pre-calculating values
  - E.g. crc64 hash of constant strings
- Can be used in conjunction with templates
- Referentially transparent
  - Does not have side effects
    - Note: this is only true if the function is run at compile time. If it is converted to a run time function, it can modify global variables!
- Haskell does this *a lot*
  - The entire program may be optimized down to constants

Constexpr Class

- Its instances can be compile-time objects
  - Same restrictions apply to methods, but can use members

```
class Rectangle  {
    int _h, _w;
public:
    // a constexpr constructor
    constexpr Rectangle (int h, int w) : _h(h), _w(w) {}
    constexpr int area ()  { return _h * _w; }
};

constexpr Rectangle rekt(10, 20);   // compile-time
print_const<rekt.area()>();         // 200

Rectangle rect(5, argc);            // runtime Rectangle
cout << rect.area() << endl;        // 5 (if argc == 1)
```