LAZY GENERATORS: TEMPLATE DEDUCTION ON THE LEFT-HAND SIDE

Simon Brand

Codeplay Software Ltd.

Agenda

- Motivation
- Implicit conversion operators
- Lazy generators
- Gotchas
- Real-world applications

MOTIVATION

Template magic usually occurs on the right-hand side of the expression.

```cpp
auto i = parse<int>();
auto s = parse<std::string>();
```

MOTIVATION

What if we could do the deduction on the left hand side?

```
int i = parse();
std::string s = parse();
```

IMPLICIT CONVERSION OPERATORS

An implicit conversion operator allows an object to convert to another type without an explicit cast.

```
struct Log {
    void bind (std::ostream&);
    bool is_bound();
    Log& operator<< (Log&, const std::string&);


    operator bool() { return is_bound(); }
};
```

```
Log log;
if (log) log << "Shouldn't happen";
log.bind(std::cout);
if (log) log << "Yay";
```

You can implicitly convert to user-defined types too.

```cpp
struct Foo{};
struct Bar {
    operator Foo() { return {}; }
};

Bar b{};
Foo f = b;
```

Implicit conversion operators can even be templates.

```
struct Foo {
    template <typename T>
    operator T() {
        T t;
        std::cin >> t;
        return t;
    }
};
```

Implicit conversion operators can even be templates.

```cpp
struct lazy_parser {
    template <typename T>
    operator T() {
        T t;
        std::cin >> t;
        return t;
    }
};
```

```cpp
lazy_parser parse();

int i = parse();
std::string s = parse();
```

IMPLEMENTING PARSE

```
lazy_parser parse() { return {}; }
```

IMPLEMENTING PARSE

```
lazy_parser parse() {
    lazy_parser parser{std::cin};
    parser.ignore_whitespace(true);
    return parser;
}
```

A SIMPLE LAZY GENERATOR

```
struct lazy_parser {
    template <typename T>
    operator T() {
        T t;
        std::cin >> t;
        return t;
    }
};

lazy_parser parse() { return {}; }

int i = parse();
std::string s = parse();
```

A lazy generator function returns an object which generates the desired value on implicit conversion.

CONSTRAINING T

```
template <typename T>
operator T*();
```

```
template <typename T,
          class=std::enable_if_t<std::is_arithmetic<T>::value>>
operator T();
```

```
template <Arithmetic T>
operator T();
```

GOTCHAS

```cpp
//problem 1
const auto& p = parse();

//problem 2
auto p = parse();

//problem 2
parser p{};

int i = p;
std::string s = p;
```

```cpp
class parser {
public:
    template <typename T>
    operator T() &&;
    //problem 1  ^^

    //problem 2
    parser (const parser&) = delete;
    parser& operator= (const parser&) = delete;

private:
    //problem 3
    parser(){}
    friend parser parse();
};
```

```
auto&& p = parse();
int i = std::move(p);
int s = std::move(p);
```

REAL WORLD EXAMPLES

- boost::nfp::named_parameter trace invalid parameters.
- boost::python::override convert Python returns.
- boost::detail::winapi::detail communicate with the Windows SDK.
- boost::initialized_value generic value initialization.
- boost::spirit::hold_any allows implicit conversion.

Blog: tartanllama.github.io

Email: simon@codeplay.com

Twitter: @TartanLlama

Codeplay: www.codeplay.com