Compile-Time Function Execution (CTFE)

Sebastian Wilzbach
seb@wilzba.ch
github.com/wilzbach

Templates - Functions

```
auto retro(Range)(Range r)
```

Templates - Functions (Invocation)

```
// [5, 4, 3, 2, 1]
[1, 2, 3, 4, 5].retro;
```

```
auto retro(Range)(Range r)
if (isBidirectionalRange!(Unqual!Range))
```

Templates - Structs

```
struct BoyerMooreFinder(Range)
{
    Range needle;
```

```
struct BoyerMooreFinder(alias pred, Range)
{
private:
    size_t[] skip;
    ptrdiff_t[ElementType!(Range)] occ;
    Range needle;
```

Templates - Classes

```
final class HashMixerRNG(Hash, uint factor) : RandomNumberStream
if(isDigest!Hash)
{
    static assert(factor, "factor must be larger than 0");
```

Templates - Classes

```
enum hash = SHA1;
enum uint factor = 5;
auto rng = new HashMixerRNG!(hash, factor)();
```

Templates - Classes

```
enum hash = SHA1;
enum uint factor = 5;
auto rng = new HashMixerRNG!(hash, factor);
```

Templates - Classes

```
final class HashMixerRNG(Hash, uint factor = 5) : RandomNumberStream
```

```
auto rng = new HashMixerRNG!SHA1;
```

Templates - Classes

```
final class HashMixerRNG(uint factor = 5, Hash) : RandomNumberStream
```

```
auto rng = new HashMixerRNG!SHA1;
```

Templates

```
template all(alias pred = "a")
{
    bool all(Range)(Range range)
    {
    }
}
```

std.algorithm.searching.all

```
template all(alias pred = "a")
{
    bool all(Range)(Range range)
    if (isInputRange!Range && is(typeof(unaryFun!pred(range.front))))
    {
        import std.functional : not;

        return find!(not!(unaryFun!pred))(range).empty;
    }
}
```

```
all!"a & 1"([1, 3, 5, 7, 9])
```

std.traits

isSafe

isArray

isPointer

isSigned

isAbstractClass

isFinalFunction

isCallable

isConvertibleToString

hasMember

https://dlang.org/phobos/std_traits.html

std.traits

Parameters

ReturnType

Fields

ConstOf

CommonType

Unqual

Promoted

TemplateArgsOf

InterfacesTuple

https://dlang.org/phobos/std_traits.html

std.meta

AliasSeq

```
alias TL = AliasSeq!(int, double);

alias Types = AliasSeq!(TL, char);
static assert(is(Types == AliasSeq!(int, double, char)));
```

https://dlang.org/phobos/std_meta.html

std.meta

AliasSeq

allSatisfy | anySatisfy

```
import std.traits : isIntegral;

static assert(!allSatisfy!(isIntegral, int, double));
static assert( allSatisfy!(isIntegral, int, long));
```

https://dlang.org/phobos/std_meta.html

std.meta

AliasSeq

allSatisfy | anySatisfy

Filter

```
import std.traits : isNarrowString;

alias Types = AliasSeq!(string, wstring, dchar[], char[], dstring, int);
alias TL1 = Filter!(isNarrowString, Types);
static assert(is(TL1 == AliasSeq!(string, wstring, char[])));
```

std.meta

AliasSeq

allSatisfy | anySatisfy

Filter

templateAnd | templateOr | templateNot

```
import std.traits : isNumeric, isUnsigned;

alias storesNegativeNumbers = templateAnd!(isNumeric, templateNot!isUnsigned);
static assert(storesNegativeNumbers!int);
static assert(!storesNegativeNumbers!string && !storesNegativeNumbers!uint);
```

CTFE - Compile Time Function evaluation

```cpp
#include <type_traits>
#include <array>


template <typename ... T>
void ignore(T&& ...);

template <typename ... T>
struct type_list
{
    constexpr type_list(T...) {}
};

template <typename ... T>
type_list(T...) -> type_list<T...>;

template <typename T, typename ... Ts>
constexpr auto head(type_list<T, Ts...>)
{
    return T{};
}

template <typename T, typename ... Ts>
constexpr auto tail(type_list<T, Ts...>)
{
    return type_list{Ts{}...};
}

template <typename ... Ts, typename ... Us>
constexpr auto operator|(type_list<Ts...>, type_list<Us...>)
{
    return type_list{Ts{}..., Us{}...};
}
```

```cpp
template <typename Compare, typename P, typename ... Ts>
constexpr auto partition(Compare compare, P pivot, type_list<Ts...> tl)
{
    if constexpr (sizeof...(Ts) == 0)
    {
        return std::pair(type_list{}, type_list{});
    }
    else
    {
        constexpr auto h = head(tl);
        constexpr auto r = partition(compare, pivot, tail(tl));
        if constexpr (compare(h, pivot))
        {
            return std::pair(type_list{h} | r.first, r.second);
        }
        else
        {
            return std::pair(r.first, type_list{h} | r.second);
        }
    }
}

template <typename Compare, typename ... T>
constexpr auto sort(type_list<T...> tl, Compare compare)
{
    if constexpr (sizeof...(T) == 0)
    {
        return type_list{};
    }
    else
    {
        constexpr auto pivot = head(tl);
        constexpr auto r = partition(compare, pivot, tail(tl));
        return sort(r.first, compare) | type_list{pivot} | sort(r.second, compare);
    }
}
```

```cpp
template <typename T, typename... Ts>
constexpr auto mkarray(type_list<Ts...>)
{
    return std::array<T, sizeof...(Ts)>{{Ts{}...}};
}

template <auto N>
std::integral_constant<decltype(N), N> c;

int main()
{
    static auto a = mkarray<int>(sort(type_list(c<2>, c<5>, c<1>, c<8>, c<4>),
                                      std::less{}));
    ignore(a);
}
```

http://playfulprogramming.blogspot.de/2017/06/constexpr-quicksort-in-c17.html

CTFE - Compile Time Function evaluation

```d
void main() {
    import std.algorithm, std.stdio;
    enum a = [3, 1, 2, 4, 0];
    static immutable b = sort(a);
    writeln(b); // [0, 1, 2, 3, 4]
}
```

https://run.dlang.io/is/BlaojH

https://dlang.org/blog/2017/06/05/compile-time-sort-in-d

CTFE - Compile Time Function evaluation

```
auto ctr = ctRegex!(`^.*/([^/]+)/?$`);
auto tr = regex(`^.*/([^/]+)/?$`);
```

CTFE - Compile Time Function evaluation

```
int[] genFactorials(int n) {
    auto result = new int[n];
    result[0] = 1;
    foreach (i; 1 .. n)
        result[i] = result[i - 1] * i;
    return result;
}

enum factorials = genFactorials(13);
```

CTFE - factorial as a pure template

```
template factorial(int n) {
    static if (n == 1) const factorial = 1;
    else const factorial = n * factorial!(n-1);
}

int x = factorial!4; // 24
```

CTFE - Compile Time Function evaluation

```
static immutable ushort[64] offsettable =
    (){
        ushort[64] t;
        t[] = 1024;
        t[0] = t[32] = 0;
        return t;
    }();
```

CTFE - Import world

```
ubyte[] fileContent = cast(ubyte[]) import("myfile.raw");
```

Static if

```
static if(is(T == int)) {
```

Static if

```
static if (hasMember!(r, "length"))
    return r.length; // O(1)
else
    return r.walkLength; // O(n)
```

```d
immutable string[] timeStrings = ["hnsecs", "usecs", "msecs", "seconds", "minutes",
                                  "hours", "days", "weeks", "months", "years"];

@safe unittest
{
    static foreach (i; 0 .. timeStrings.length)
    {
        static assert(CmpTimeUnits!(timeStrings[i], timeStrings[i]) == 0);

        static foreach (next; timeStrings[i + 1 .. $])
            static assert(CmpTimeUnits!(timeStrings[i], next) == -1);

        static foreach (prev; timeStrings[0 .. i])
            static assert(CmpTimeUnits!(timeStrings[i], prev) == 1);
    }
}
```

Mixin

```
mixin("int b = 5;");
assert(b == 5);

mixin(genCode());
```

Mixin

```
auto calculate(string op, T)(T lhs, T rhs)
{
    return mixin("lhs " ~ op ~ " rhs");
}

calculate!"+"(5, 12);
```

In Action

String interpolation

```
import scriptlike;
int num = 21;
writeln(mixin(interp!"The number ${num} doubled is ${num * 2}."));
```

String interpolation

```
string interp(string str)()
{
    enum State
    {
        normal,
        dollar,
        code,
    }

    auto state = State.normal;

    string buf = `';
```

```
foreach(c; str)
final switch(state) {
case State.normal:
    if(c == '$')
        state = State.dollar;
    else if(c == '`')
        buf ~= "`~\"`\"~`";
    else
        buf ~= c;
    break;
```

https://github.com/Abscissa/scriptlike/blob/master/src/scriptlike/core.d

String interpolation

```
case State.dollar:
    if(c == '{') {
        state = State.code;
        buf ~= "`~_interp_text(";
    } else {
        buf ~= '$'; // Copy the previous $
        if(c != '$') {
            buf ~= c;
            state = State.normal;
        }
    }
    break;
```

```
case State.code:
    if(c == '}') {
        buf ~= ")~`";
        state = State.normal;
    } else
        buf ~= c;
    break;
}
```

```
final switch(state)
{
case State.normal:
    buf ~= '`';
    break;

case State.dollar:
    buf ~= "$`"; // Copy the previous $
    break;

case State.code:
    throw new Exception(
        "Interpolated string contains an unterminated expansion. "~
        "You're missing a closing curly brace."
    );
}

return buf;
```

String interpolation

```
alias _interp_text = std.conv.text;
```

std.algorithm.searching.commonPrefix

```
auto commonPrefix(alias pred = "a == b", R1, R2)(R1 r1, R2 r2)
if (isForwardRange!R1 && isInputRange!R2 &&
    !isNarrowString!R1 &&
    is(typeof(binaryFun!pred(r1.front, r2.front))))
```

```
import std.algorithm.comparison : min;
static if (isRandomAccessRange!R1 && isRandomAccessRange!R2 &&
           hasLength!R1 && hasLength!R2 &&
           hasSlicing!R1)
{
    immutable limit = min(r1.length, r2.length);
    foreach (i; 0 .. limit) {
        if (!binaryFun!pred(r1[i], r2[i])) {
            return r1[0 .. i];
        }
    }
    return r1[0 .. limit];
}
```

```
else
{
    import std.range : takeExactly;
    auto result = r1.save;
    size_t i = 0;
    for (;
         !r1.empty && !r2.empty && binaryFun!pred(r1.front, r2.front);
         ++i, r1.popFront(), r2.popFront())
    {}
    return takeExactly(result, i);
}
```

```
assert(commonPrefix([1, 2, 3], [1, 2, 4]) == [1, 2]);
```

std.bitmanip

```
struct A
{
    int a;
    mixin(bitfields!(
        uint, "x",    2,
        int,  "y",    3,
        uint, "z",    2,
        bool, "flag", 1));
}
A obj;
obj.x = 2;
obj.z = obj.x;
```

https://dlang.org/phobos/std_bitmanip.html

std.bitmanip

$$(-1)^{\text{sign}} \times 2^{e-1023} \times 1.\text{fraction}$$

```d
import std.bitmanip : bitfields;

struct DoubleRep
{
    union
    {
        double value;
        mixin(bitfields!(
                    ulong,   "fraction", 52,
                    ushort,  "exponent", 11,
                    bool,    "sign",      1));
    }
    enum uint bias = 1023, signBits = 1, fractionBits = 52, exponentBits = 11;
}

auto d = DoubleRep(20.5);
writeln(d.fraction, " ", d.exponent, " ", d.sign); // 1266637395197952 1027 false
d.fraction = 3;
writeln(d.value); // ?
```

https://dlang.org/phobos/std_bitmanip.html

Variadic min (std.algorithm.comparison.min)

```
MinType!T min(T...)(T args)
if (T.length >= 2)
{
    // Get "a"
    static if (T.length <= 2)
        alias a = args[0];
    else
        auto a = min(args[0 .. ($+1)/2]);
    alias T0 = typeof(a);

    // Get "b"
    static if (T.length <= 3)
        alias b = args[$-1];
    else
        auto b = min(args[($+1)/2 .. $]);
    alias T1 = typeof(b);

    import std.algorithm.internal : algoFormat;
    static assert(is(typeof(a < b)),
        algoFormat("Invalid arguments: Cannot compare types %s and %s.",
        T0.stringof, T1.stringof));

    // Do the "min" proper with a and b
    import std.functional : lessThan;
    immutable chooseA = lessThan!(T0, T1)(a, b);
    return cast(typeof(return)) (chooseA ? a : b);
```

std.range.retro

```
auto retro(Range)(Range r)
if (isBidirectionalRange!(Unqual!Range)) {
    static if (is(typeof(retro(r.source)) == Range) {
        return r.source;
    } else {
        static struct Result() {
            private alias R = Unqual!Range;

            // User code can get and set source, too
            R source;

            static if (hasLength!R) {
                size_t retroIndex(size_t n) {
                    return source.length - n - 1;
                }
            }
```

std.range.retro

```
public:
    alias Source = R;

    @property bool empty() { return source.empty; }
    @property auto save() {
        return Result(source.save);
    }
    @property auto ref front() { return source.back; }
    void popFront() { source.popBack(); }
    @property auto ref back() { return source.front; }
    void popBack() { source.popFront(); }
```

```
    static if (is(typeof(source.moveBack()))) {
        ElementType!R moveFront() {
            return source.moveBack();
        }
    }

    static if (is(typeof(source.moveFront()))) {
        ElementType!R moveBack() {
            return source.moveFront();
        }
    }
```

std.range.retro

```
static if (hasAssignableElements!R) {
    @property void front(ElementType!R val) {
        source.back = val;
    }

    @property void back(ElementType!R val) {
        source.front = val;
    }
}
```

```
static if (isRandomAccessRange!(R) && hasLength!(R)) {
    auto ref opIndex(size_t n) { return source[retroIndex(n)]; }

    static if (hasAssignableElements!R)
    void opIndexAssign(ElementType!R val, size_t n) {
        source[retroIndex(n)] = val;
    }

    static if (is(typeof(source.moveAt(0))))
    ElementType!R moveAt(size_t index) {
        return source.moveAt(retroIndex(index));
    }

    static if (hasSlicing!R)
    typeof(this) opSlice(size_t a, size_t b) {
        return typeof(this)(source[source.length - b .. source.length - a]);
    }
}
```

std.range.retro

```
        static if (hasLength!R) {
            @property auto length() {
                return source.length;
            }

            alias opDollar = length;
        }
    }

    return Result!()(r);
```

std.experimental.allocator.makeArray

```
T[] makeArray(T, Allocator)(auto ref Allocator alloc, size_t length, T init)
{
    if (!length) return null;
    auto m = alloc.allocate(T.sizeof * length);
    if (!m.ptr) return null;
    auto result = () @trusted { return cast(T[]) m; } ();
    import std.traits : hasElaborateCopyConstructor;
```

```
static if (hasElaborateCopyConstructor!T) {
    scope(failure) {
        static if (canSafelyDeallocPostRewind!T)
            () @trusted { alloc.deallocate(m); } ();
        else
            alloc.deallocate(m);
    }

    size_t i = 0;
    static if (hasElaborateDestructor!T) {
        scope (failure) {
            foreach (j; 0 .. i) {
                destroy(result[j]);
            }
        }
    }
    import std.conv : emplace;
    for (; i < length; ++i) {
        emplace!T(&result[i], init);
    }
}
```

vibe.web.web

```
class SampleService {
    @noAuth {
        @path("/") void getHome(scope HTTPServerRequest req)
        {
            import std.typecons : Nullable;

            Nullable!AuthInfo auth;
            if (req.session && req.session.isKeySet("auth"))
                auth = req.session.get!AuthInfo("auth");

            render!("home.dt", auth);
        }
}
```

```
private void handleRequest(string M, alias overload, C, ERROR...)
(HTTPServerRequest req, HTTPServerResponse res, C instance, WebInterfaceSettings settings, ERROR error)
    if (ERROR.length <= 1)
{
    import std.algorithm : countUntil, startsWith;
    import std.traits;
    import std.typetuple : Filter, staticIndexOf;
    import vibe.core.stream;
    import vibe.data.json;
    import vibe.internal.meta.funcattr;
    import vibe.internal.meta.uda : findFirstUDA;

    alias RET = ReturnType!overload;
    alias PARAMS = ParameterTypeTuple!overload;
    alias default_values = ParameterDefaultValueTuple!overload;
    alias AuthInfoType = AuthInfo!C;
    enum param_names = [ParameterIdentifierTuple!overload];
    enum erruda = findFirstUDA!(ErrorDisplayAttribute, overload);

    static if (findFirstUDA!(NestedNameStyleAttribute, C).found)
        enum nested_style = findFirstUDA!(NestedNameStyleAttribute, C).value.value;
    else enum nested_style = NestedNameStyle.underscore;
```

vibe.web.web

```
s_requestContext = createRequestContext!overload(req, res);
enum hasAuth = isAuthenticated!(C, overload);

static if (hasAuth) {
    auto auth_info = handleAuthentication!overload(instance, req, res);
    if (res.headerWritten) return;
}
```

```
static foreach (i, PT; PARAMS) {
    bool got_error = false;
    ParamError err;
    err.field = param_names[i];
    try {
        static if (hasAuth && is(PT == AuthInfoType)) {
            params[i] = auth_info;
        } else static if (IsAttributedParameter!(overload, param_names[i])) {
            params[i].setVoid(computeAttributedParameterCtx!(overload, param_names[i])(instance, req, res));
            if (res.headerWritten) return;
        }
        else static if (param_names[i] == "_error") {
            static if (ERROR.length == 1)
                params[i].setVoid(error[0]);
            else static if (!is(default_values[i] == void))
                params[i].setVoid(default_values[i]);
            else
                params[i] = typeof(params[i]).init;
        }
```

vibe.web.web

```
else static if (is(PT == InputStream)) params[i] = req.bodyReader;
else static if (is(PT == HTTPServerRequest) || is(PT == HTTPRequest)) params[i] = req;
else static if (is(PT == HTTPServerResponse) || is(PT == HTTPResponse)) params[i] = res;
else static if (is(PT == WebSocket)) {} // handled below
else static if (param_names[i].startsWith("_")) {
    if (auto pv = param_names[i][1 .. $] in req.params) {
        got_error = !webConvTo(*pv, params[i], err);
        if (got_error) return;
    } else static if (!is(default_values[i] == void)) params[i].setVoid(default_values[i]);
    else static if (!isNullable!PT) enforceHTTP(false, HTTPStatus.badRequest, "Missing request parameter
} else static if (is(PT == bool)) {
    params[i] = param_names[i] in req.form || param_names[i] in req.query;
} else {
```

Backup - Phobos

std.algorithm.mutation.fill

```
void fill(Range, Value)(auto ref Range range, auto ref Value value)
if ((isInputRange!Range && is(typeof(range.front = value)) ||
    isSomeChar!Value && is(typeof(range[] = value))))
{
    alias T = ElementType!Range;

    static if (is(typeof(range[] = value))) {
        range[] = value;
    } else static if (is(typeof(range[] = T(value)))) {
        range[] = T(value);
    } else {
        for ( ; !range.empty; range.popFront()) {
            range.front = value;
        }
    }
}
```

```
@safe unittest
{
    int[] a = [ 1, 2, 3, 4 ];
    fill(a, 5);
    assert(a == [ 5, 5, 5, 5 ]);
}
```

```
InputRange find(alias pred = "a == b", InputRange, Element)(InputRange haystack, scope Element needle)
if (isInputRange!InputRange &&
    is (typeof(binaryFun!pred(haystack.front, needle)) : bool))
{
    alias R = InputRange;
    alias E = Element;
    alias predFun = binaryFun!pred;
    static if (is(typeof(pred == "a == b")))
        enum isDefaultPred = pred == "a == b";
    else
        enum isDefaultPred = false;
    enum  isIntegralNeedle = isSomeChar!E || isIntegral!E || isBoolean!E;

    alias EType  = ElementType!R;

    import std.range : SortedRange;
    static if (is(InputRange : SortedRange!TT, TT) && isDefaultPred)
    {
        auto lb = haystack.lowerBound(needle);
        if (lb.length == haystack.length || haystack[lb.length] != needle)
            return haystack[$ .. $];

        return haystack[lb.length .. $];
    }
```

std.algorithm.searching.find

std.algorithm.searching.countUntil

```
ptrdiff_t countUntil(alias pred, R)(R haystack)
if (isInputRange!R && is(typeof(unaryFun!pred(haystack.front)) : bool))
{
    typeof(return) i;
    static if (isRandomAccessRange!R)
    {
        // Count *and* iterate at the same time
        static if (hasLength!R) {
            immutable len = cast(typeof(return)) haystack.length;
            for ( ; i < len ; ++i )
                if (unaryFun!pred(haystack[i])) return i;
        } else {
            for ( ;  ; ++i )
                if (unaryFun!pred(haystack[i])) return i;
        }
    }
    else static if (hasLength!R) {
        // It is faster to quick find, and then compare the lengths
        auto r2 = find!pred(haystack.save);
        if (!r2.empty) return cast(typeof(return)) (haystack.length - r2.length);
    }
    else {
        alias T = ElementType!R; // For narrow strings forces dchar iteration
        foreach (T elem; haystack) {
            if (unaryFun!pred(elem)) return i;
            ++i;
        }
    }

    return -1;
```

std.algorithm.searching.countUntil

```
  mov rax, -1
  test rdi, rdi
  jle .LBB1_4
  xor ecx, ecx
.LBB1_2:
  cmp dword ptr [rsi + 4*rcx], 2
  jle .LBB1_3
  add rcx, 1
  cmp rcx, rdi
  jl .LBB1_2
.LBB1_4:
  ret
.LBB1_3:
  mov rax, rcx
  ret
```

```
  mov r8, qword ptr [rsp + 24]
  mov rdi, qword ptr [rsp + 8]
  mov rsi, qword ptr [rsp + 16]
  mov rcx, -1
.LBB1_1:
  lea rax, [r8 + rcx]
  add rax, 1
  mov rdx, rax
  or rdx, rdi
  shr rdx, 32
  je .LBB1_2
  xor edx, edx
  div rdi
  add rcx, 1
  cmp dword ptr [rsi + 4*rdx], 2
  jg .LBB1_1
  jmp .LBB1_5
.LBB1_2:
  xor edx, edx
  div edi
  add rcx, 1
  cmp dword ptr [rsi + 4*rdx], 2
  jg .LBB1_1
.LBB1_5:
  mov rax, rcx
  ret
```