Fun with C and C++ syntax

Jonathan Müller — @foonathan

We develop an add-in for Microsoft PowerPoint that makes it easier, faster and more enjoyable to create graphical presentations.

think-cell

think-cell

We develop an add-in for Microsoft PowerPoint that makes it easier, faster and more enjoyable to create graphical presentations.

- We reverse engineer PowerPoint and inject our own code.

think-cell

think-cell

We develop an add-in for Microsoft PowerPoint that makes it easier, faster and more enjoyable to create graphical presentations.

- We reverse engineer PowerPoint and inject our own code.
- We solve complex mathematical problems like automatic point cloud labeling.

think-cell

think-cell

We develop an add-in for Microsoft PowerPoint that makes it easier, faster and more enjoyable to create graphical presentations.

- We reverse engineer PowerPoint and inject our own code.
- We solve complex mathematical problems like automatic point cloud labeling.
- We use C++ to its fullest: the latest feature, custom standard library, participation in the standardization process,

think-cell

Fun with C and C++ syntax

C's syntax is weird.

And C++ did not make it better.

think-cell

[] is commutative

```
int array[SIZE];

array[17] = 42;
```

think-cell

[] is commutative

```
int array[SIZE];
```

```
array[17] = 42;
```

```
*(array + 17) = 42;
```

think-cell

[] is commutative

```
int array[SIZE];

array[17] = 42;

*(array + 17) = 42;

*(17 + array) = 42;
```

think-cell

[] is commutative

```
int array[SIZE];

array[17] = 42;

*(array + 17) = 42;

*(17 + array) = 42;

17[array] = 42;
```

think-cell

[] is commutative

```c
int array[SIZE];

array[17] = 42;

*(array + 17) = 42;

*(17 + array) = 42;

17[array] = 42;
```

```cpp
std::array<int, SIZE> array;

array[17] = 42;
```

think-cell

[] is commutative

```cpp
int array[SIZE];

array[17] = 42;

*(array + 17) = 42;

*(17 + array) = 42;

17[array] = 42;
```

```cpp
std::array<int, SIZE> array;

array[17] = 42;

array.operator[](17) = 42;
```

think-cell

[] is commutative

```c
int array[SIZE];

array[17] = 42;

*(array + 17) = 42;

*(17 + array) = 42;

17[array] = 42;
```

```c
std::array<int, SIZE> array;

array[17] = 42;

array.operator[](17) = 42;

17[array] = 42; // compiler error :(
```

think-cell

[] is commutative

That's why array indexing starts with `0`:

```
int array[SIZE];
array[0]     = 11;
*(array + 0) = 11;
*array       = 11;
```

think-cell

Control flow in C++

- `if`

think-cell

Control flow in C++

- `if`
- `else`

think-cell

Control flow in C++

- `if`
- `else`
- `switch`

think-cell

Control flow in C++

- `if`
- `else`
- `switch`
- `for`

think-cell

Control flow in C++

- `if`
- `else`
- `switch`
- `for`
- `while`

think-cell

Control flow in C++

- `if`
- `else`
- `switch`
- `for`
- `while`
- (`goto`)

think-cell

`else if` doesn't exist

think-cell

`else if` doesn't exist

[stmt.select.general]/1

```
if constexpr? ( init-statement? condition ) statement
if constexpr? ( init-statement? condition ) statement else statement
```

think-cell

`else if` doesn't exist

```
if (a) {
    …
} else if (b) {
    …
} else {
    …
}
```

think-cell

`else if` doesn't exist

```
if (a) {
    …
} else if (b) {
    …
} else {
    …
}
```

```
if (a) {
    …
} else { if (b) {
  …
} else {
  …
} }
```

think-cell

`else if` doesn't exist

Common coding guideline

Use braces even for single statements.

think-cell

`else if` doesn't exist

```
bool is_beautiful(std::optional<color> color)
{
    if (!color)
        return false; // lack of color is not beautiful
    else switch (*color) {
        case red:
        case blue:
        case yellow:
            return true;
        default:
            return false;
    }
}
```

think-cell

`else if` doesn't exist

```
bool is_beautiful(std::optional<color> color)
{
    if (!color)
        return false; // lack of color is not beautiful
    else switch (*color) {
        case red:
        case blue:
        case yellow:
            return true;
        default:
            return false;
    }
}
```

Who needs pattern matching?!

think-cell

switch

```
switch (i)
{
case 1:
case 2:
case 3:
    std::puts("i is 1, 2, or 3");
    break;

default:
    std::puts("i is something else");
    break;
}
```

think-cell

switch

jMGrKbMKh

```
switch (i)
{
default:
    std::puts("i is something else");
    break;

case 1:
case 2:
case 3:
    std::puts("i is 1, 2, or 3");
    break;
}
```

think-cell

switch

zb7cGGfze

```
switch (i)
{
    std::puts("I'm never executed");

case 1:
case 2:
case 3:
    std::puts("i is 1, 2, or 3");
    break;

default:
    std::puts("i is something else");
    break;
}
```

think-cell

Aside: `using enum`

```
enum class foo { a, b, c };
```

```
const char* to_string(foo f)
{
    switch (f)
    {
    case foo::a:
        return "a";
    case foo::b:
        return "b";
    case foo::c:
        return "c";
    }
}
```

think-cell

Aside: `using enum`

```
enum class foo { a, b, c };
```

```
const char* to_string(foo f)
{
    using enum foo;

    switch (f)
    {
    case a:
        return "a";
    case b:
        return "b";
    case c:
        return "c";
    }
}
```

Aside: `using enum`

```
enum class foo { a, b, c };
```

```
const char* to_string(foo f)
{
    switch (f)
    {
        using enum foo;
    case a:
        return "a";
    case b:
        return "b";
    case c:
        return "c";
    }
}
```

switch

sK3rKq1s6

```
switch (i)
    case 1:
    case 2:
    case 3:
        std::puts("i was 1, 2, or 3");

std::puts("after the switch");
```

think-cell

switch

Prd8bT5Gd

```
switch (i)
    if (i == 0)
    {
        std::puts("I'm never executed");
    }
    else
    {
case 0:
        std::puts("i is zero");
    }
```

think-cell

Duff's Device

raz65nTva

Copy `count` bytes from `from` to `to`.

```
int n = count;
    do
    {
      *to = *from++;
    } while (--n > 0);
```

think-cell

Duff's Device

raz65nTva

Copy `count` bytes from `from` to `to`.

```
int n = count / 8;
    do
    {
      *to = *from++;
      *to = *from++;
      *to = *from++;
      *to = *from++;
      *to = *from++;
      *to = *from++;
      *to = *from++;
      *to = *from++;
    } while (--n > 0);
```

think-cell

Duff's Device

raz65nTva

Copy `count` bytes from `from` to `to`.

```
int n = (count + 7) / 8;
switch (count % 8)
    do
    {
case 0: *to = *from++;
case 7: *to = *from++;
case 6: *to = *from++;
case 5: *to = *from++;
case 4: *to = *from++;
case 3: *to = *from++;
case 2: *to = *from++;
case 1: *to = *from++;
    } while (--n > 0);
```

switch_no_default

4caKMf78x

```
#define switch_no_default(...) \
    switch( __VA_ARGS__ ) \
    default: \
        if (true) UNREACHABLE("missing switch case"); \
        else
```

```
switch_no_default (i)
{
case 1:
case 2:
case 3:
    std::puts("i was 1, 2, or 3");
    break;
}
```

think-cell

Declaration specifier ordering

```
const int a;
```

think-cell

Declaration specifier ordering

```
const int a;
```

```
int const a;
```

think-cell

Declaration specifier ordering

```
const int a;
```

```
int const a;
```

```
constexpr explicit b(…);
```

```
explicit constexpr b(…);
```

think-cell

Declaration specifier ordering

```
const int a;
```

```
int const a;
```

```
constexpr explicit b(…);
```

```
explicit constexpr b(…);
```

```
unsigned int c;
```

```
int unsigned c;
```

think-cell

Declaration specifier ordering

cs4bKcz3x

```
decl-specifier-seq:
    decl-specifier
    decl-specifier decl-specifier-seq
```

think-cell

Declaration specifier ordering

cs4bKcz3x

```
decl-specifier-seq:
    decl-specifier
    decl-specifier decl-specifier-seq
```

```
int typedef a;
```

think-cell

Declaration specifier ordering

cs4bKcz3x

```
decl-specifier-seq:
    decl-specifier
    decl-specifier decl-specifier-seq
```

```
int typedef a;
```

```
volatile inline float static b;
```

think-cell

Declaration specifier ordering

cs4bKcz3x

```
decl-specifier-seq:
    decl-specifier
    decl-specifier decl-specifier-seq
```

```
int typedef a;
```

```
volatile inline float static b;
```

```
int constexpr c;
```

think-cell

Declaration specifier ordering

cs4bKcz3x

```
decl-specifier-seq:
    decl-specifier
    decl-specifier decl-specifier-seq
```

```
int typedef a;
```

```
volatile inline float static b;
```

```
int constexpr c;
```

```
long thread_local unsigned extern long d;
```

think-cell

Declarator

```
constexpr unsigned int name;
```

think-cell

Declarator

```
constexpr unsigned int name;
```

Philosophy: Mirror expression syntax.

```
int *ptr;
int array[10];
int function(int);
```

```
*ptr;
array[0];
function(2);
```

think-cell

Declarator

```
constexpr unsigned int name;
```

Philosophy: Mirror expression syntax.

```
int *ptr;
int array[10];
int function(int);
```

```
*ptr;
array[0];
function(2);
```

C++: `int& reference;` …

think-cell

Parenthesized declarators

```
int *array_of_ptrs[10];
int (*ptr_to_array)[10];
```

```
*array_of_ptrs[0];
(*ptr_to_array)[0];
```

think-cell

Parenthesized declarators

```
int *array_of_ptrs[10];
int (*ptr_to_array)[10];
```

```
*array_of_ptrs[0];
(*ptr_to_array)[0];
```

```
int (parens);
```

think-cell

Parenthesized declarators

```
int *array_of_ptrs[10];
int (*ptr_to_array)[10];
```

```
*array_of_ptrs[0];
(*ptr_to_array)[0];
```

```
int (parens);
```

```
int ((((function))))();
```

think-cell

Parenthesized declarators

```
int *array_of_ptrs[10];
int (*ptr_to_array)[10];
```

```
*array_of_ptrs[0];
(*ptr_to_array)[0];
```

```
int (parens);
```

```
int ((((function))))();
```

```
int b(int arg);
int c = 11;
T a(b(c)); // constructor?
```

think-cell

Parenthesized declarators

```
int *array_of_ptrs[10];
int (*ptr_to_array)[10];
```

```
*array_of_ptrs[0];
(*ptr_to_array)[0];
```

```
int (parens);
```

```
int ((((function))))();
```

```
int b(int arg);
int c = 11;
T a(b c); // function!
```

think-cell

Multiple declarators

```
int a, b;
```

think-cell

Multiple declarators

```
int a, b, *c;
```

think-cell

Multiple declarators

```
int a, b, *c;
int* a, b;
```

think-cell

Multiple declarators

```
int a, b, *c, d = 42;
```

think-cell

Multiple declarators

```
int a, b, *c, d = 42, e();
```

think-cell

Multiple declarators

```
int a, b, *c, d = 42, e(), f(int arg);
```

think-cell

Multiple declarators

```
int a, b, *c, d = 42, e(), f(int arg), (*g(float arg))(int* arg);
```

think-cell

Function pointer syntax

Variable:

```
int (*ptr)(int);
int result = (*ptr)(11);
```

think-cell

Function pointer syntax

Variable:

```
int (*ptr)(int);
int result = (*ptr)(11);
```

Function return type:

```
int (*f(int))(int);
int result = (*f(0))(11);
```

think-cell

Function pointer syntax

Variable:

```
int (*ptr)(int);
int result = (*ptr)(11);
```

Function return type:

```
int (*f(int))(int);
int result = (*f(0))(11);
```

Array:

```
int (*array[10])(int);
int result = (*array[0])(11);
```

think-cell

Function pointer syntax

Conversion operator:

```
auto lambda = [](int arg) -> int { return 2 * arg; };
```

```
struct lambda_t
{
    operator int(*)(int) ();
};
```

think-cell

Function pointer syntax

Conversion operator:

```
auto lambda = [](int arg) -> int { return 2 * arg; };
```

```
struct lambda_t
{
    int (*operator())(int);
};
```

think-cell

Function pointer syntax

Conversion operator:

```
auto lambda = [](int arg) -> int { return 2 * arg; };
```

```
struct lambda_t
{
    operator int(*())(int);
};
```

think-cell

Function pointer syntax

Conversion operator:

```
auto lambda = [](int arg) -> int { return 2 * arg; };
```

```
struct lambda_t
{
    (*operator int())(int);
};
```

think-cell

Function pointer syntax

Conversion operator:

```
auto lambda = [](int arg) -> int { return 2 * arg; };
```

```
struct lambda_t
{
    operator auto();
};
```

think-cell

Use a function pointer

```c
int f(int arg) { return 2 * arg; }
```

```c
int (*ptr)(int) = &f;

(*ptr)(0);
```

think-cell

Use a function pointer

```c
int f(int arg) { return 2 * arg; }
```

```c
void (*ptr)(int) = f;

ptr(0);
```

think-cell

Use a function pointer

```
int f(int arg) { return 2 * arg; }
```

```
void (*ptr)(int) = f;

ptr(0);
```

[conv.func]/1

> *An lvalue of* ***function type T can be converted to*** *a prvalue of* ***type "pointer to T"****. The result is a pointer to the function*

[expr.call]/1

> ***A function call*** *is a postfix expression followed by parentheses containing a possibly empty, comma-separated list of initializer-clauses which constitute the arguments to the function. The postfix expression* ***shall have function type or function pointer type****.*

think-cell

Use a function pointer

[conv.func]/1

An lvalue of ***function type T can be converted to*** *a prvalue of* ***type “pointer to T”****. The result is a pointer to the function*

```
void f(int);
```

```
f(0);
```

think-cell

Use a function pointer

[conv.func]/1

An lvalue of ***function type T can be converted to*** *a prvalue of* ***type "pointer to T"****. The result is a pointer to the function*

```
void f(int);
```

```
(*f)(0);
```

think-cell

Use a function pointer

[conv.func]/1

An lvalue of ***function type T can be converted to*** *a prvalue of* ***type "pointer to T"****. The result is a pointer to the function*

```
void f(int);
```

```
(**f)(0);
```

think-cell

Use a function pointer

[conv.func]/1

An lvalue of ***function type T can be converted to*** *a prvalue of* ***type "pointer to T"****. The result is a pointer to the function*

```
void f(int);
```

```
(***f)(0);
```

think-cell

Use a function pointer

[conv.func]/1

An lvalue of ***function type T can be converted to*** *a prvalue of* ***type "pointer to T"****. The result is a pointer to the function*

```
void f(int);
```

```
(**********************f)(0);
```

think-cell

Use a function pointer

[conv.func]/1

An lvalue of ***function type T can be converted to*** *a prvalue of* ***type "pointer to T"****. The result is a pointer to the function*

```
void f(int);
```

```
(**********************
 */* function call */*
 **********************f)(0);
```

think-cell

Local global declarations

ea1jzq3Pv

```
extern int global;
void g();

void f()
{
    ++global;
    g();
}
```

think-cell

Local global declarations

ea1jzq3Pv

```
void f()
{
    extern int global;
    void g();

    ++global;
    g();
}
```

think-cell

`static` in C has only a single meaning

```c
static int file_local = 42;

void f()
{
    ++file_local;
}
```

```c
void f()
{
    static int file_local = 42;
    ++file_local;
}
```

Only difference: visibility of `file_local`.

think-cell

think-cell is hiring

C++ Developer/Internship

- An international team of brilliant minds
- Support with relocation to Berlin or work fully remotely
- EUR 130,000 annually after only one year
- No scheduled meetings, no deadlines, no overtime

Come to our booth!

think-cell

Integer overflow

Is there UB?

```
int f(int a, int b)
{
    return a + b;
}
```

think-cell

Integer overflow

Is there UB?

```
int f(int a, int b)
{
    return a * b;
}
```

think-cell

Integer overflow

Is there UB?

```
int f(int a, int b)
{
    return a * b;
}
```

Sean Parent: overflow on 99.9999993% of all possible inputs.

think-cell

Integer overflow

Is there UB?

```
int f(int a, int b)
{
    return a / b;
}
```

think-cell

Integer overflow

Is there UB?

```
int f(int a, int b)
{
    assert(b != 0);
    return a / b;
}
```

think-cell

Aside: Two's complement

Positive values: `0b0'xxxxxxx`

Negative values: `0b1'xxxxxxx`

think-cell

Aside: Two's complement

Positive values: `0b0'xxxxxxx`

Negative values: `0b1'xxxxxxx`

What about zero?

think-cell

Aside: Two's complement

Positive values: `0b0'xxxxxxx`

Negative values: `0b1'xxxxxxx`

What about zero?

```
-128
-127
…
-1
0
1
…
126
127
```

```
0b1'0000000
0b1'0000001
…
0b1'1111111
0b0'0000000
0b0'0000001
…
0b0'1111110
0b0'1111111
```

think-cell

Integer overflow

Is there UB?

```
int f(int a, int b)
{
    assert(b != 0);
    return a / b;
}
```

```
f(INT_MIN, -1) // integer overflow!
```

think-cell

Integer overflow

Is there UB?

```
int f(int a, int b)
{
    assert(b != 0);
    return a % b;
}
```

think-cell

Integer overflow

Is there UB?

```
int f(int a, int b)
{
    assert(b != 0);
    return a % b;
}
```

```
f(INT_MIN, -1) // integer overflow!?
```

think-cell

Integer overflow

[expr.mul]/4

> *The binary / operator yields the quotient, and the binary % operator yields the remainder from the division of the first expression by the second. If the second operand of / or % is zero the behavior is undefined. For integral operands the / operator yields the algebraic quotient with any fractional part discarded; if the quotient `a/b` is representable in the type of the result, `(a/b)*b + a%b` is equal to `a`;* ***otherwise, the behavior of both `a/b` and `a%b` is undefined.***

think-cell

Integer overflow

```
int f(int a, int b)
{
    assert(b != 0);
    return a % b;
}
```

```
mov     eax, DWORD PTR [rbp-4]
cdq
idiv    DWORD PTR [rbp-8]
mov     eax, edx
```

`idiv` computes quotient in `eax` and remainder in `edx`.

think-cell

Integer overflow

```c
int f(int a, int b)
{
    assert(b != 0);
    return a % b;
}
```

```
ldr     w8, [sp, #12]
ldr     w10, [sp, #8]
sdiv    w9, w8, w10
mul     w9, w9, w10
subs    w0, w8, w9
```

```c
return a - (a / b) * b;
```

think-cell

Integer overflow

```
$ lldb ./a.out
(lldb) target create "./a.out"
Current executable set to '/home/foonathan/Downloads/a.out' (x86_64).
(lldb) r
Process 645117 launched: '/home/foonathan/Downloads/a.out' (x86_64)
Process 645117 stopped
* thread #1, name = 'a.out',
    stop reason = signal SIGFPE: integer divide by zero
    frame #0: 0x0000555555555180 a.out`f(int, int) + 64
a.out`f:
->  0x555555555180 <+64>: idivl  -0x8(%rbp)
    0x555555555183 <+67>: movl   %edx, %eax
    0x555555555185 <+69>: addq   $0x10, %rsp
    0x555555555189 <+73>: popq   %rbp
```

think-cell

think-cell is hiring

C++ Developer/Internship

- An international team of brilliant minds
- Support with relocation to Berlin or work fully remotely
- EUR 130,000 annually after only one year
- No scheduled meetings, no deadlines, no overtime

Come to our booth!

think-cell