SO YOU WANT TO REMOVE ALL UB

Martin Hořeňovský

PEX

SO YOU WANT TO REMOVE ALL UB

Martin Hořeňovský

PEX

We all hate UB, right?

```cpp
#include <cstdio>

int main(){
    int i = 1;
    while (i > 0){
        std::puts("Hello\n");
        i *= 2;
    }
}
```

```cpp
#include <cstdio>

int main(){
    int i = 1;
    while (i > 0){
        std::puts("Hello\n");
        i *= 2;
    }
}
```

```asm
main:
        push    rbx
        lea     rbx, [rip + .L.str]
.LBB0_1:
        mov     rdi, rbx
        call    puts@PLT
        jmp     .LBB0_1
.L.str:
        .asciz  "Hello\n"
```

```
#include <cstdio>

static int f(int i) { return i+1;}

int main(){
    int i = 1;
    while (f(i) != i){}
    puts("How did we get here?\n");
}
```

```
#include <cstdio>

static int f(int i) { return i+1;}

int main(){
    int i = 1;
    while (f(i) != i){}
    puts("How did we get here?\n");
}
```

```
main:
```

```
static int elements[] = {1, 2, 3, 4};

static bool contains(int elem) {
    for (int i = 0; i <= 4; ++i) {
        if (elements[i] == elem) { return true; }
    }
    return false;
}

int main() {
    int num;
    while (std::cin >> num) {
        std::cout << std::boolalpha
                  << contains(num) << '\n';
    }
}
```

```
$ clang++ -O2 out-of-bounds.cpp && ./a.out
1
true
12
true
500
true
65465465
true
66666666666666666666666666666
$
```

WHAT IS UB?

UB is a contract between the programmer ...

UB is a contract between the programmer ...

... and the compiler.

The programmer promises that certain things will never happen.

The programmer promises that certain things will never happen.

The compiler promises to optimize the code as much as it can, assuming no UB.

WHY IS UB?

Many useful properties either cannot be proven locally, or at all.

Many useful properties either cannot be proven locally, or at all.

UB moves this responsibility to the programmer instead.

CAN WE GET RID OF UB?

NO

NO

```
template <typename T>
void debug_print(T const* ptr) {
    if (ptr) {
        print(*ptr);
    } else {
        print("nullptr");
    }
}

void f(int const& i) {
    debug_print(&i); // laziness ftw
}
```

```
template <typename T>
void debug_print(T const* ptr) {
    if (ptr) {
        print(*ptr);
    } else {
        print("nullptr");
    }
}

void f(int const& i) {
    debug_print(&i); // laziness ftw
}
```

```
void f(int const& i) {
    print(i);
}
```

```
void scan(int* length, float* array) {
    float sum = 0.0f;
    for (int i = 0; i < *length; i++) {
        sum += array[i];
        array[i] = sum;
    }
}
```

```
void scan(int* length, float* array) {
    float sum = 0.0f;
    for (int i = 0; i < *length; i++) {
        sum += array[i];
        array[i] = sum;
    }
}
```

```
scan(int*, float*):
        mov     eax, dword ptr
        test    eax, eax
        jle     .LBB0_3
        xorps   xmm0, xmm0
        xor     ecx, ecx
.LBB0_2:                 # =>This Inne
        addss   xmm0, dword pt
        movss   dword ptr [rsi
        inc     rcx
        cmp     rax, rcx
        jne     .LBB0_2
.LBB0_3:
        ret
```

CAN WE GET RID OF SOME UB?

INTEGER OVERFLOW?

INTEGER OVERFLOW?

Sure, two's complement has won.

INFINITE LOOPS?

INFINITE LOOPS?

```
for (p = q; p != 0; p = p->next) {
    ++count;
}
for (p = q; p != 0; p = p->next) {
    ++count2;
}
```

INFINITE LOOPS?

```
for (p = q; p != 0; p = p->next) {
    ++count;
}
for (p = q; p != 0; p = p->next) {
    ++count2;
}
```

```
for (p = q; p != 0; p = p->next) {
        ++count;
        ++count2;
}
```

Others?

¯_(ツ)_/¯

THE END