

CSC 347 - Concepts of Programming Languages

Undefined Behavior

Instructor: Stefan Mitsch



Learning Objectives

- ❓ What should happen if the language does not specify the meaning of some construct?
- ❓ What should happen when `x` is holding 8 bits, `x=7` and then `x = x+1` ?
 - Identify undefined behavior in C



Datatype Bounds

- Natural numbers are only partly useful as integer arithmetic semantics

$$\frac{}{\langle n, \xi \rangle \Downarrow \langle n, \xi \rangle} \text{ (Num)}$$

- Add integer bounds

$$\frac{-2^{63} \leq n \leq 2^{63} - 1}{\langle n, \xi \rangle \Downarrow \langle n, \xi \rangle} \text{ (Num)}$$

- What to do with overflow?

- Error:
$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle \quad \neg(-2^{63} \leq v_1 + v_2 \leq 2^{63} - 1)}{\langle e_1 + e_2, \xi_0 \rangle \Downarrow \langle \text{error}, \xi_2 \rangle} \text{ (Add), } \dots$$



Under and Overflow

```
1 #include <stdio.h>
2
3 int isMinValue (int x) {
4     return (x-1) > x;
5 }
6 int main () {
7     int i = -2000000000;
8     while (!isMinValue(i))
9         i--;
10    printf ("Min value is %d\n", i);
11 }
```

```
1 $ gcc -O1 undefined.c && ./a.out
2 Min value is -2147483648
```

```
1 $ gcc -O2 undefined.c && ./a.out
2 ^C #infinite loop
3
```



Order of Operations

- Recall [rule for sequential composition](#)
- Rule define an order of evaluating subexpressions

$$\frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 ; e_2, \xi_0 \rangle \Downarrow \langle v_2, \xi_2 \rangle} (:) \qquad \frac{\langle e_1, \xi_0 \rangle \Downarrow \langle v_1, \xi_1 \rangle \quad \langle e_2, \xi_1 \rangle \Downarrow \langle v_2, \xi_2 \rangle}{\langle e_1 + e_2, \xi_0 \rangle \Downarrow \langle v_1 + v_2, \xi_2 \rangle} (\text{Add})$$

- by chaining the stores
- ξ_0 is used in evaluating e_1 and produces ξ_1
- ξ_1 is used in evaluating e_2 and produces ξ_2



Undefined Order of Operations

```
1 #include <stdio.h>
2 int count = 0;
3 int f () {
4     count += 1;
5     return count;
6 }
7 int main () {
8     int z = f() + f();
9     printf ("%d\n", z);
10    z = (z += 1) + (z = z*z);
11    printf ("%d\n", z);
12 }
```

```
1 $ clang -Wall undefined3.c
2 undefined3.c:11:21: warning: unsequenced modification and access to 'z'
3   z = (z += 1) + (z = z*z);
4             ~~~ ^
5 1 warning generated.
6 $ ./a.out
7 3
8 20
```

- `z=z+1; z=z+z*z`



Undefined Order of Operations

```
1 #include <stdio.h>
2 int count = 0;
3 int f () {
4     count += 1;
5     return count;
6 }
7 int main () {
8     int z = f() + f();
9     printf ("%d\n", z);
10    z = (z += 1) + (z = z*z);
11    printf ("%d\n", z);
12 }
```

```
1 $ gcc -Wall -O3 undefined3.c
2 undefined3.c: In function 'main':
3 undefined3.c:11:5: warning: operation on 'z' may be undefined
4     z = (z += 1) + (z = z*z);
5         ^
6 $ ./a.out
7 3
8 32
```

- `z=z+1; z=z*z; z=z+z;`



Compiler Optimizations

- For undefined executions, the compiler can do what it likes
- This can lead to some surprising compiler optimizations
- [C null pointer optimization 1](#)



Summary

- Undefined behavior resolved in the compiler (compiler decides what is the meaning of undefined programs)
- Undefined behavior often presents security risks
- [More examples of undefined behavior](#)
- [More undefined behavior](#)