Creative Software Design

Undefined Behaviors

Yunho Kim
yunhokim@hanyang.ac.kr
Dept. of Computer Science

Topics Covered

- Undefined, Unspecified, Implementation-Defined behaviors
- Various examples

Pop Quiz

```
double f = 18 / 0.0;
int i = h(), g();
const int i = 42; *(int*)&i = 43;
#define _Alloc MyAlloc
double d[1024][1024][1024][1024];
*d++ = t[*s++] * 16 + t[*s++];
delete 0;
int x = 78765; x *= 201531;
```

Definition: Undefined Behavior

- “Behavior, such as might arise upon use of an erroneous program construct or erroneous data, for which the Standard imposes no requirements”
- “If a program contains no violations of the rules of the Standard, a conforming implementation shall accept and correctly execute the program”
  - “Correct execution can include undefined behavior”
- Examples: division by zero, accessing an array out of bounds

Definition: Unspecified Behavior

- “Behavior, for a well-formed program construct and correct data, that depends on the implementation”
- “The implementation is not required to document which behavior occurs”
- Examples: order in which container elements are destroyed, order of evaluation between sequence points

Definition: Implementation-Defined Behavior

- “Behavior, for a well-formed program construct and correct data, that depends on the implementation and that each implementation shall document”

- Examples: sizeof(int), whether signed integers use two’s complement

Common Areas of Danger

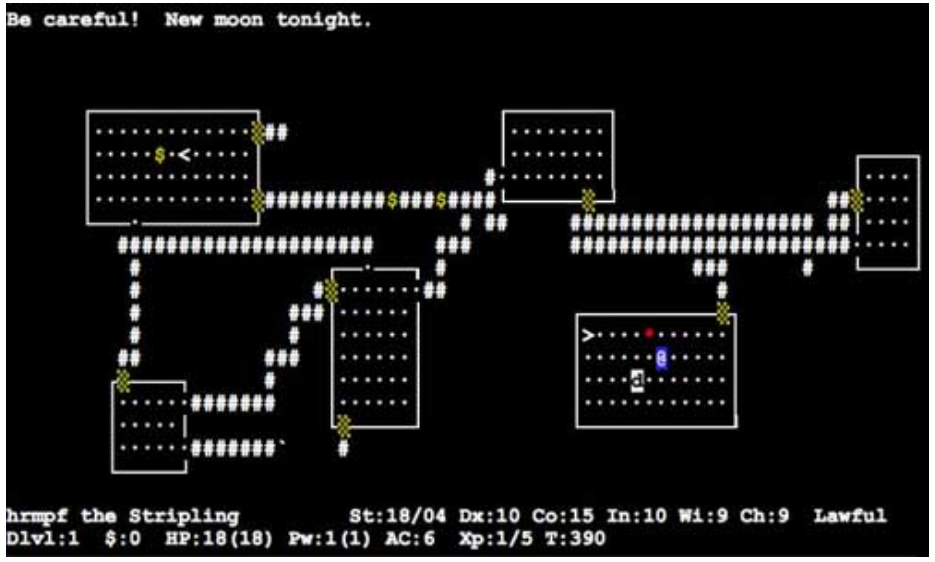
- Object boundaries
- Object lifetime
- Function parameters
- Memory
- Mathematics
- Order of evaluation
- Conversions
- Incomplete builds
- Const
- Pointers
- Padding
- Standard type sizes
- Inheritance
- Algorithms
- Iterators

Potential Outcomes

- Compiler error (best outcome)
- Compiler warning
- Analysis tool error
- Analysis tool warning
- Assertion
- Consistent crash, data loss, or corruption
- Program termination
- Completely unpredictable results
- It works (worst outcome)

Classic Outcome

```
// when GCC 1.17 encountered specific forms of
// undefined behavior, here's the code it executed:
execl("/usr/games/hack", "#pragma", 0);
execl("/usr/games/rogue", "#pragma", 0);
execl("/usr/new/emacs",  "-f","hanoi","9","-kill",0);
execl("/usr/local/emacs","-f","hanoi","9","-kill",0);
fatal("You are in a maze of twisty compiler features,
       all different");
```



Reasons for Undefined Behavior

- Simplifies the Standard
- Allows extra latitude for implementers
- Makes your programs run faster
- Examples:
  - Array bounds checking
  - Constants
  - Division by zero

Conventions

```
code = “looks like this”;
// ☠ undefined behavior
// 💣 unspecified behavior
// 💻 implementation-defined behavior
```

Object Boundaries

```
char a[N];          // C-style array
char c = *(a+N); // ☠ outside bounds
char* p = a+N;    // OK, points at end
c = *p;             // ☠ outside bounds
++p;                // ☠ outside bounds

vector<int> v; v.push_back(1);
int i = v[10];    // ☠ outside bounds

memcpy(a, a+1, N-1); // ☠ overlap
```

Object Lifetime

```
int* p = new int; delete p;
int i = *p;              // ☠ lifetime ended

int& foo() { int f = 1; return f; }
i = foo();               // ☠ lifetime ended

{ int a = 42; p = &a; }
i = *p;                  // ☠ lifetime ended
```

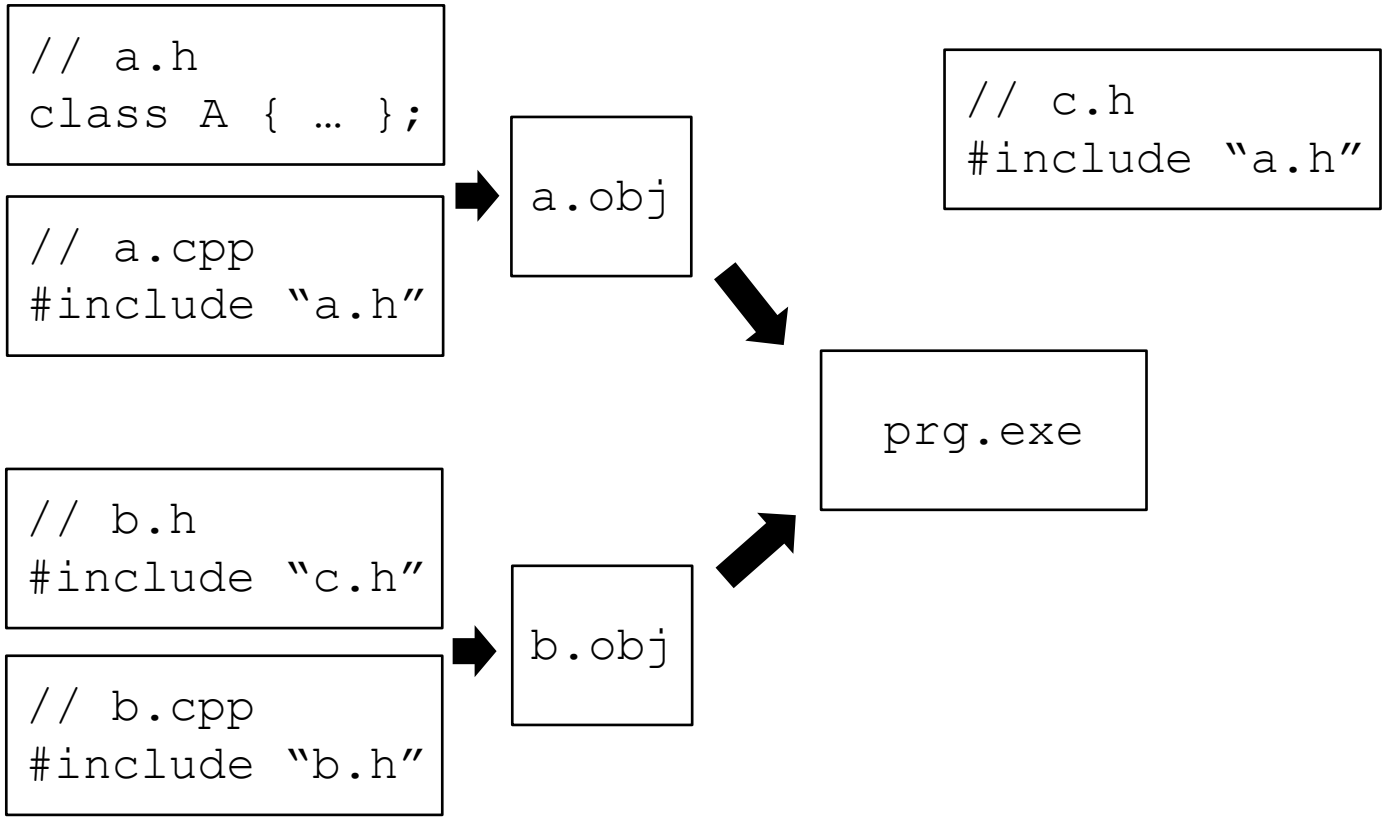
Memory

```
int* p = new int [0]; // OK (really)
int i = *p;           // ☠ zero size
delete p;             // ☠ need delete []

int* q = new int;
int* r = q;
delete q;
delete r;     // ☠ double delete
delete 0;     // req. to have no effect
```

More Memory Dragons

```
int* p = (int*)malloc( sizeof(int) );
delete p;                    // ☠ doesn't match

delete (void*)rand(); // ☠ ptr didn't
                      // come from new
```

One Definition Rule



- Suppose A changes and only a.obj is rebuilt…

One Definition Rule

- There must be one and only one definition for each non-inline function or object in your program
- More than one is OK if they are in separate translation units (CPP files) and they are each absolutely identical
- ODR not met = undefined behavior

Mathematics

```
int z = 0;
int j = 5 % z;          // ☠/💻 div by zero
double f = 1.0 / z; // ☠/💻 div by zero

unsigned int a;
a << -1; a << 128; // ☠ invalid shifts

signed int b = -1234;
b << 1;          // ☠ no left-shift neg#
b >> 1;          // 💻 impl-defined
```

Overflow

```
unsigned int i = 123456;
i *= 123456;          // well-defined
// 123456² mod 2^(sizeof(i)*CHAR_BIT)

signed int i = 123456;
i *= 123456; // ☠ not mathmat. defined
```

Conversions

```
char c = 1234;          // 💻 impl-defined
float f = 16777217; // 💻 impl-defined
c = 12345.1;            // ☠ doesn't fit

fpos_t pos; fgetpos( f, &pos );
          // 💣 fpos_t could be any size
__int64 d = *(__int64*)(&pos);
```

Order of Evaluation

```
i = g(), h();       // g(), i=g(), h()
i = (g(), h());     // g(), h(), i=h()
f(g(), h());        // 💣 g() h() order
new A(g(), h());    // 💣 alloc order
i = ++i + i++;      // ☠ stored val changed
f(++i, ++i);        // ☠ UB until C++17,
                    // 💣 unspecified after C++17
```

Order of Evaluation Solutions

- Be explicit about the order; let the compiler handle optimizations
- `a = g(); b = h(); f( a, b );`

- `i = v[i]; ++i;`

- `f( i+1, i+2 ); i += 2;`

String Literals

```
char* sz  = "Nice house";
char* uz  = "ice house";
char vz[] = "house";
sz[n] = 'x'; // ☠ modifying str literal
uz == sz+1 ; // 💻 maybe or maybe not
vz[n] = 'x'; // only affects vz

void f( char* s ) { s[0] = 'x'; }
f( "abc" );     // ☠ literals are const
```

More Const

```
const int i = 42;
*(int*)&i = 1234;          // ☠ i is const

int* p = const_cast<int*>( &i );
*p = 1234;                 // ☠ i is const
```

Padding

```
struct H { char a;
           short b;
           int c; };
H h;
char* p = (char*)&h;
char a = *p;                    // OK
                                // for Plain Old Data(POD)
int off = sizeof(char)+sizeof(short);
int c = *(int*)(p+off); // ☠ padding
```

Padding Info

- No leading padding in a POD struct
- Within access specifiers (public, private), data member addresses are in order
- `sizeof` operator includes padding
- Padding bytes contain unspecified values
- Use `offsetof`(struct, member) to determine byte offset in POD structs

Algorithms

```
int arr[6] = { 1,5,7,2,4,3 };
                    // ☠ list isn't sorted
bool b = binary_search(arr, arr+6, 3);

set<int, less_equal<int> > s;
s.insert(42);
s.insert(42); // ☠ compare returns 1
                    // for equivalent values
```

Iterators

```
typedef list<int>::iterator itr;
for(itr i=l.begin(); i!=l.end(); ++i)
{
    if( *i == 3 )
        l.erase( i ); // ☠ invalidates i
}
```

Reserved Identifiers

```
// ☠ re-defines a reserved function
int atoi(const char*) { return 1; }

// ☠ xor is an alt. reserved word
int xor(int i, int j) { return i^j; }

#define new MY_NEW     // ☠ reserved
#define __cplusplus 1 // ☠ reserved
#define _Amtyp A_TYPE // ☠ reserved
```

Recommendations

- Always always avoid undefined behavior
- Separate out implementation-defined behavior
- Get a copy of the Standard. No, really.
- Compile at the highest warning level
- Use multiple compilers
- Use static/dynamic analysis tools (/analyze in VS, lint, sanitizer)
- Enable STL iterator debugging flags

Pop Quiz Answers

```
double f = 18 / 0.0;                                    ☠
int i = h(), g();    // h(), i=h(), g()                 ☺
const int i = 42; *(int*)&i = 43;                       ☠
#define _Alloc MyAlloc                                  ☠
double d[1024][1024][1024][1024];                       💻
*d++ = t[*s++] * 16 + t[*s++];                          ☠
delete 0;                                               ☺
int x = 78765; x *= 201531;                             ☠
```

Lecture Evaluation

- Please do not forget to do lecture evaluation
  - Your faithful lecture evaluation is a good motivation for better lectures next semesters
  - Please give me a feedback in detail to improve next lectures

- Do not worry about the disadvantages
  - Of course, I cannot identify who gives which scores
  - Also, I cannot see (final) lecture evaluation before I finish grading

- You do not need to give me a good score
  - Just evaluate what you have been experienced this lecture