Secure Programming

Peter Chapin
Vermont Technical College

C Language Behaviors

Taxonomy of Error Behavior

- The C standard defines various kind of behavior
  - *Implementation Defined Behavior*
    - Behavior that is correct, but may vary from one implementation to another.
    - Implementation defined behavior must be documented.
  - *Unspecified Behavior*
    - Behavior that is correct, but not specified nor documented.
    - Unspecified behavior may vary from moment to moment.
  - *Undefined Behavior*
    - Completely undefined: “Anything goes!”

Implementation Defined Behavior

- Example: The range on primitive types…
  - C requires minimal ranges:
    - -32767 <= short int <= 32767 (16 bits)
    - -32767 <= int <= 32767 (16 bits)
    - -2147483647 <= long int <= 2147483647 (32 bits)
  - C also requires that the actual sizes be such that:

    ```
    sizeof(short) <= sizeof(int) <= sizeof(long)
    ```

  - C leaves the actual ranges Implementation Defined

Implementation Defined Behavior

- A typical compiler for 32 bit targets uses:
  - Straightforward 2’s complement ranges (note the extra negative):
    - -32768 <= short int <= 32767 (16 bits)
    - -2147483648 <= int <= 2147483647 (32 bits)
    - -2147483648 <= long int <= 2147483647 (32 bits)

Implementation Defined Behavior

- Consider a program that assumes int has a large range:
  - ```
    int line_count;
    …
    line_count = 50000;  // Implementation defined!
    ```
  - The problem is that not all compilers use a range for int that includes 50000. On such machines the value is truncated (CWE-197).
  - The program is fine and works perfectly on machines using 32 bit int.
    ```

Implementation Defined Behavior

- Java has much less implementation defined behavior
  - For example, the range on basic types is fixed by the language:
    - The type int is definitely 32 bit using 2's complement
    - The type long is definitely 64 bit using 2's complement
  - This simplifies life for the programmer…
  - … BUT it prevents Java from taking advantage of diverse systems effectively
    - On small machines, large integers are imposed.
    - On large machines, the full capacity of the machine is harder to access.

Implementation Defined Behavior

- The Ada language allows programmers to specify ranges:
  - ```
    type Line_Counter_Type is range 0 .. 1_000_000;
    …
    Line_Counter : Line_Counter_Type;
    …
    Line_Counter := 50_000;  -- Works on all machines.
    ```
  - Makes range information explicit in the program.
    ```

Unspecified Behavior

- Classic example: order of evaluation of function arguments.
  - `x = f( a + b, c - d );`
    - Which is evaluated first: a + b or c – d?
    - The C standard says the order is “unspecified.”
    - The compiler is allowed either order, it does not have to document it
    - The order might be different for different function calls
    - The order might be different each time the program runs!
    - In this case it doesn’t matter and nobody cares.

Unspecified Behavior

- Now consider this example:
  - `x = f( g( ), h( ) );`
    - Which function is called first? g() or h()?
    - The order is still unspecified
    - … BUT it might make a difference: suppose g() outputs “Hello” and h() outputs “World.” Does the program output “Hello World” or “World Hello?”
    - The program *might* do what is intended, but that would be by accident. Don’t write code that relies on unspecified behavior!

Undefined Behavior

- Anything can happen. Usually the program crashes.
  - ```
    char buffer[128];
    …
    buffer[128] = 'x';  // Array out of bounds is UB.
    ```
  - Even reading an array out of bounds is UB (not just writing to it).
  - It might “work.” Program continues and computes a reasonable result. Or… program might output garbage ultimately. Or… program might crash immediately. Or… program might crash much later.
    ```

Undefined Behavior

- In C, many things are undefined
  - Illegal array access
  - Integer overflow
  - Comparing pointers into different arrays
  - Many others...

Undefined Behavior

- Integer overflow… (example assumes 32 bit integers)
  - ```
    int x = 1000000;
    int y = 1000000;
    int z;
    …
    z = x * y; // Result overflows integer. UB!
    ```
  - Most compilers will wrap the result and continue executing.
  - In theory the program might crash. Most like it will produce garbage
    ```

Undefined Behavior

- Pointers into different arrays can’t be compared…
  - ```
    char buffer1[128];
    char buffer2[128];
    char *p1 = &buffer1[64];
    char *p2 = &buffer2[64];
    …
    if( p1 < p2 ) { … // Undefined behavior!
    ```
  - Program will probably evaluate this to true/false depending on the relative positions of the arrays in memory. OTOH, the program might crash.
    ```

Strictly Conforming

- A program is *Strictly Conforming* if…
  - … it engages in no implementation defined, unspecified, or undefined behavior.
- Such programs are highly portable
  - They should compile and work on every system that supports standard C.

Help!!

- How is a programmer to keep all of this straight?
  - In reality: It is very difficult.
  - Consequently many C programs crawl with implementation defined, unspecified, and even undefined behavior.
  - Many C programs have rampant issues because of this and suffer from reliability and security problems.
- Tools can help!
  - We will talk about this later in the class.

Other Languages?

- Other languages have similar issues, to a lesser degree.
  - Java is more defined: accessing an array out of bounds throws an ArrayIndexOutOfBoundsException
    - The behavior is well defined.
    - Still probably not desirable to have this happening, though.
  - In general languages vary greatly in these areas; learning about them is part of learning a language.