[Use intmax\_t or uintmax\_t for formatted IO on programmer-defined integer types](https://www.securecoding.cert.org/confluence/display/cplusplus/INT15-CPP.+Use+intmax_t+or+uintmax_t+for+formatted+IO+on+programmer-defined+integer+types)

Few programmers consider the issues around formatted I/O and type definitions. A programmer-defined integer type might be any type supported by the implementation, even a type larger than unsigned long long.

For example, given an implementation that supports 128-bit unsigned integers and provides a uint\_fast128\_t type, a programmer may define the following type:

|  |
| --- |
| **typedef uint\_fast128\_t mytypedef\_t;** |

Furthermore, the definition of programmer-defined types may change. This creates a problem using these types with formatted output functions such as printf()and formatted input functions such as scanf().

The C99 **intmax\_t** **and** **uintmax\_t** types are capable of representing any value representable by any other integer types of the same signedness. This allows conversion between programmer-defined integer types (of the same signedness) and intmax\_t and uintmax\_t. For example:

|  |
| --- |
| mytypedef\_t x;  uintmax\_t temp;  /\* ... \*/  temp = x; /\* always secure\*/    /\* ... change the value of temp ... \*/    if (temp <= MYTYPEDEF\_MAX) {    x = temp;  } |

Formatted I/O functions can be used to input and output greatest-width integer typed values. **The j length modifier in a format string indicates that** the following d, i, o, u, x, X, or n conversion specifier will apply to an argument with type intmax\_t or uintmax\_t. **C99 also specifies the z length modifier for use with arguments of type size\_t and the t length modifier for arguments of type ptrdiff\_t.**

Under the language standard, the intmax\_t and uintmax\_t data types are, respectively, signed and unsigned integer types with the largest length possible. They can be represented through extended integer types. Section 7.18.1.5 of the Standard(**C Programming languages standard**) only requires that intmax\_t and uintmax\_t should be large enough to store values represented by any other integer data types. Like extended integer types, they are defined in the stdint.h header file together with their smallest and largest values INTMAX\_MIN, INTMAX\_MAX, and UINTMAX\_MAX. For intmax\_t and uintmax\_t, the "j" letter serves as an input/output modifier.

In addition to programmer-defined types, there is no requirement that an implementation provides format length modifiers for **implementation-defined** integer types. For example, a machine with an implementation-defined 48-bit integer type may not provide format length modifiers for the type. Such a machine would still have to have a 64-bit long long, with intmax\_t being at least that large.

Noncompliant Code Example (printf())

This noncompliant code example prints the value of x as an unsigned long long value, even though the value is of a programmer-defined integer type.

|  |
| --- |
| #include <stdio.h>    mytypedef\_t x;    /\* ... \*/    printf("%llu", (unsigned long long) x); |

Consequently, there is no guarantee that this code prints the correct value of x, as x is not necessarily an unsigned long long value and may be too large to represent as an unsigned long long.

Compliant Solution (printf())

The C99 **intmax\_t and uintmax\_t** can be safely used to perform formatted I/O with programmer-defined integer types. This is accomplished by converting signed programmer-defined integer types to intmax\_t and unsigned programmer-defined integer types to uintmax\_t, then **outputting these values using the j length modifier**. Similarly, programmer-defined integer types can be input to variables of intmax\_t or uintmax\_t (whichever matches the signedness of the programmer-defined integer type) and then converted to programmer-defined integer types using appropriate range checks.

This compliant solution guarantees that the correct value of x is printed, regardless of its length, provided that mytypedef\_t is an unsigned type.

|  |
| --- |
| #include <stdio.h>  #include <inttypes.h>    mytypedef\_t x;    /\* ... \*/    printf("%ju", (uintmax\_t) x); |

Noncompliant Code Example (scanf())

The following noncompliant code example reads an unsigned long long value from standard input and stores the result in x, which is of a programmer-defined integer type.

|  |
| --- |
| #include <stdio.h>    mytypedef\_t x;  /\* ... \*/  if (scanf("%llu", &x) != 1) {    /\* handle error \*/  } |

This noncompliant code example can result in a buffer overflow, if the size of mytypedef\_t is smaller than unsigned long long, or it might result in an incorrect value if the size of mytypedef\_t is larger than unsigned long long.

Compliant Solution (scanf())

This compliant solution guarantees that a correct value in the range of mytypedef\_t is read, or an error condition is detected, assuming the value of MYTYPEDEF\_MAX is correct as the largest value representable by mytypedef\_t.

|  |
| --- |
| #include <stdio.h>  #include <inttypes.h>    mytypedef\_t x;  uintmax\_t temp;    /\* ... \*/    if (scanf("%ju", &temp) != 1) {    /\* handle error \*/  }  if (temp > MYTYPEDEF\_MAX) {    /\* handle error \*/  }  else {    x = temp;  } |