Integer Promotions in C++

Adam Graliński

C++ Friends, October 2022

Introduction

"Use `ints` unless you need something different. Then still use something signed, until you **really** need something different, at which point — resort to unsigned"

— Herb Sutter, Interactive Panel @ Microsoft's Going Native Conference 2013

Introduction

"Yeah, I was going to say something very similar. Use **`ints`** — until you have a reason not to. Don't use **`unsigned`** unless you're fiddling with bit-patterns, and never mix **`signed`** and **`unsigned`**.

— Bjarne Stroustrup, Interactive Panel @ Microsoft's Going Native Conference 2013 (12:30 in)

Pop quiz I

intro.cpp

```cpp
#include <iostream>
#include <vector>

template<typename T_LHS, typename T_RHS>
void print_comparison(T_LHS lhs, T_RHS rhs)
{
  std::cout << lhs << " < " << rhs << ": "
    << std::boolalpha << (lhs < rhs) << "\n";
}

int main()
{
  long negative{-123};
  unsigned short positive{123};
  std::size_t unsigned_positive{456};

  print_comparison(negative, positive);           // A
  print_comparison(negative, unsigned_positive);  // B
}
```

Pop quiz I

intro.cpp

```cpp
#include <iostream>
#include <vector>

template<typename T_LHS, typename T_RHS>
void print_comparison(T_LHS lhs, T_RHS rhs)
{
  std::cout << lhs << " < " << rhs << ": "
    << std::boolalpha << (lhs < rhs) << "\n";
}

int main()
{
  long negative{-123};
  unsigned short positive{123};
  std::size_t unsigned_positive{456};

  print_comparison(negative, positive);          // A
  print_comparison(negative, unsigned_positive);  // B
}
```

output

```
-123 < 123: true
-123 < 456: false
```

https://godbolt.org/z/ePaWrfexf

Pop quiz II

surprises.cpp

```
#include <limits>
#include <iostream>

int main()
{
  static_assert(sizeof(int) > sizeof(unsigned short));

  unsigned short one = 1;
  unsigned short maxshort = std::numeric_limits<unsigned short>::max();
  unsigned short sum = maxshort + one;

  std::cout << "one == " << one
            << ",\nmaxshort == " << maxshort
            << ",\nsum = maxshort+one = " << sum << "\n";
  if (sum == maxshort + one) {
    std::cout << "As expected!\n";
  }
  else {
    std::cout << "Oh no!\n";
  }
}
```

Pop quiz II

surprises.cpp

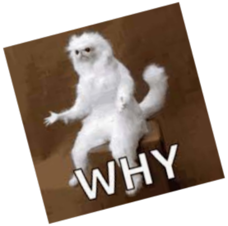
```
#include <limits>
#include <iostream>

int main()
{
  static_assert(sizeof(int) > sizeof(unsigned short));

  unsigned short one = 1;
  unsigned short maxshort = std::numeric_limits<unsigned short>::max();
  unsigned short sum = maxshort + one;

  std::cout << "one == " << one
            << ",\nmaxshort == " << maxshort
            << ",\nsum = maxshort+one = " << sum << "\n";
  if (sum == maxshort + one) {
    std::cout << "As expected!\n";
  }
  else {
    std::cout << "Oh no!\n";
  }
}
```

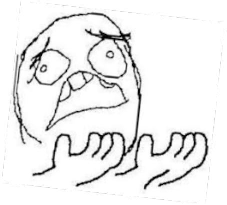
output

```
one == 1,
maxshort == 65535,
sum = maxshort+one = 0
Oh no!
```

https://godbolt.org/z/aKzEsaY8q

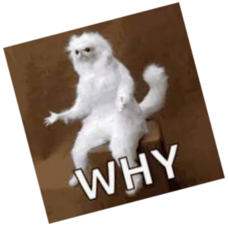
Why?

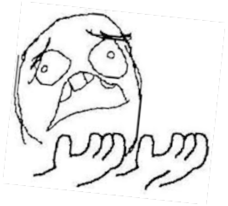




- undefined behavior?
- cosmic rays?
- ghost in the machine?

Why?

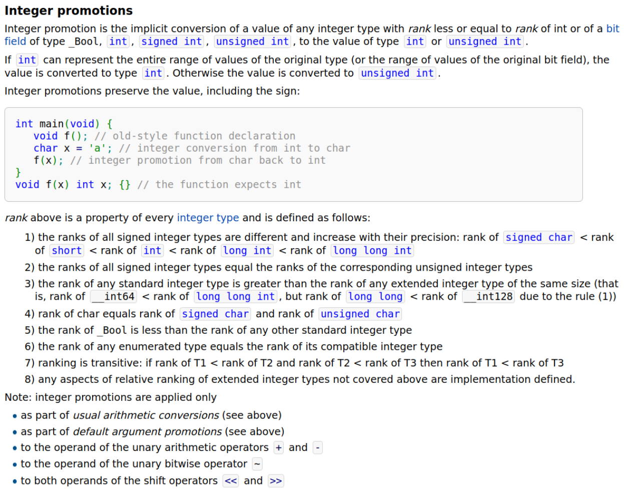




- undefined behavior?
- cosmic rays?
- ghost in the machine?

No, this is *well defined behavior*.
Both caused by *implicit promotion rules for numeric data types*.

cppreference sources



Integer promotions

Integer promotion is the implicit conversion of a value of any integer type with *rank* less or equal to *rank* of int or of a bit field of type _Bool, `int`, `signed int`, `unsigned int`, to the value of type `int` or `unsigned int`.

If `int` can represent the entire range of values of the original type (or the range of values of the original bit field), the value is converted to type `int`. Otherwise the value is converted to `unsigned int`.

Integer promotions preserve the value, including the sign:

```
int main(void) {
   void f(); // old-style function declaration
   char x = 'a'; // integer conversion from int to char
   f(x); // integer promotion from char back to int
}
void f(x) int x; {} // the function expects int
```

rank above is a property of every integer type and is defined as follows:

1) the ranks of all signed integer types are different and increase with their precision: rank of `signed char` < rank of `short` < rank of `int` < rank of `long int` < rank of `long long int`
2) the ranks of all signed integer types equal the ranks of the corresponding unsigned integer types
3) the rank of any standard integer type is greater than the rank of any extended integer type of the same size (that is, rank of `__int64` < rank of `long long int`, but rank of `long long` < rank of `__int128` due to the rule (1))
4) rank of char equals rank of `signed char` and rank of `unsigned char`
5) the rank of _Bool is less than the rank of any other standard integer type
6) the rank of any enumerated type equals the rank of its compatible integer type
7) ranking is transitive: if rank of T1 < rank of T2 and rank of T2 < rank of T3 then rank of T1 < rank of T3
8) any aspects of relative ranking of extended integer types not covered above are implementation defined.

Note: integer promotions are applied only

- as part of *usual arithmetic conversions* (see above)
- as part of *default argument promotions* (see above)
- to the operand of the unary arithmetic operators `+` and `-`
- to the operand of the unary bitwise operator `~`
- to both operands of the shift operators `<<` and `>>`

https://en.cppreference.com/w/cpp/language/implicit_conversion#Numeric_promotions

Pop quiz I — explanation

intro-explained.cpp

```
int main()
{
  long negative{-123};
  unsigned short positive{123};
  std::size_t unsigned_positive{456};

  print_comparison(negative, positive);           // A
  print_comparison(negative, unsigned_positive);  // B
  explain();
}

void explain() {
  std::cout << "-123(long) is "
    << static_cast<unsigned long>(-123)
    << "(unsigned long)\n";
}
```

Pop quiz I — explanation

intro-explained.cpp

```
int main()
{
  long negative{-123};
  unsigned short positive{123};
  std::size_t unsigned_positive{456};

  print_comparison(negative, positive);          // A
  print_comparison(negative, unsigned_positive);  // B
  explain();
}

void explain() {
  std::cout << "-123(long) is "
    << static_cast<unsigned long>(-123)
    << "(unsigned long)\n";
}
```

output

```
-123 < 123: true
-123 < 456: false
-123(long) is
18446744073709551493
 (unsigned long)
```

https://godbolt.org/z/1efG9WP1n

Pop quiz II — explanation

surprises-explained.cpp

```cpp
#include <limits>
#include <iostream>
int main()
{
  static_assert(sizeof(int) > sizeof(unsigned short));
  unsigned short one = 1;
  unsigned short maxshort =
      std::numeric_limits<unsigned short>::max();
  unsigned short sum = maxshort + one;
  std::cout << "one == " << one
            << ",\nmaxshort == " << maxshort
            << ",\nsum = maxshort+one = " << sum << "\n";
  if (sum == maxshort + one) {
    std::cout << "As expected!\n";
  } else {
    std::cout << "Oh no!\n";
  }
  // Explanation:
  std::cout << "\nReminder: sum = maxshort + one\n";
  std::cout << "sum is: " << sum << "\n";
  std::cout << "(maxshort + one) is: " << maxshort+one << "\n";
  // dear reviewer: note the promotion happening when adding unsigned numbers
}
```

Pop quiz II — explanation

surprises-explained.cpp

```cpp
#include <limits>
#include <iostream>
int main()
{
  static_assert(sizeof(int) > sizeof(unsigned short));
  unsigned short one = 1;
  unsigned short maxshort =
      std::numeric_limits<unsigned short>::max();
  unsigned short sum = maxshort + one;
  std::cout << "one == " << one
            << ",\nmaxshort == " << maxshort
            << ",\nsum = maxshort+one = " << sum << "\n";
  if (sum == maxshort + one) {
    std::cout << "As expected!\n";
  } else {
    std::cout << "Oh no!\n";
  }
  // Explanation:
  std::cout << "\nReminder: sum = maxshort + one\n";
  std::cout << "sum is: " << sum << "\n";
  std::cout << "(maxshort + one) is: " << maxshort+one << "\n";
  // dear reviewer: note the promotion happening when adding unsigned numbers
}
```

output

```
one == 1,
maxshort == 65535,
sum = maxshort+one = 0
Oh no!

Reminder:
sum = maxshort + one
sum is: 0
(maxshort+one) is: 65536
```

Key takeaways

- **never compare signed and unsigned integers!**
- enable *-Wsign-compare* (and *-Werror* to force this).
- remember that small unsigned ints might be promoted.
  - Especially that they **are** promoted when using mathematical operators (e.g. `operator+()`, `operator*()`).
- also, keep in mind that both `int32_t` and `int64_t` are valid candidates for integer promotion, too

  on platforms where int is of size 64-bits and 128-bits, respectively

  – we may be writing vulnerable code right now, for the future platforms.
  - although future compiler writers already know this, and will probably provide mitigations
- statically-check, lint, sanitize and fuzz your codebases.

Thank you!