

Extracted from:

C++ Brain Teasers

Exercise Your Mind

This PDF file contains pages extracted from *C++ Brain Teasers*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2023 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

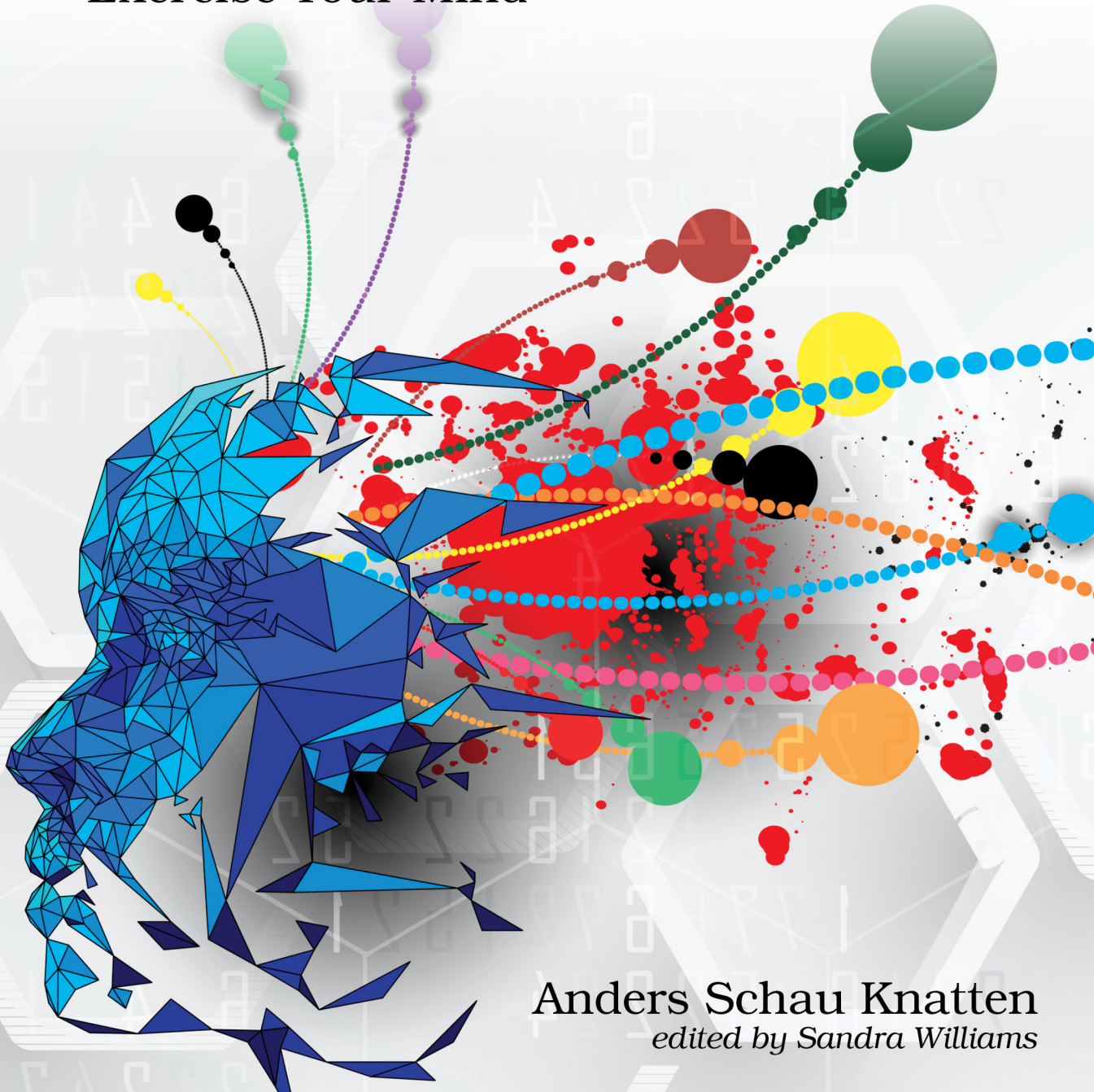
The Pragmatic Bookshelf

Dallas, Texas

The
Pragmatic
Programmers

C++ Brain Teasers

Exercise Your Mind



Anders Schau Knatten
edited by Sandra Williams

C++ Brain Teasers

Exercise Your Mind

Anders Schau Knatten

The Pragmatic Bookshelf

Dallas, Texas



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit <https://pragprog.com>.

For sales, volume licensing, and support, please contact support@pragprog.com.

For international rights, please contact rights@pragprog.com.

Copyright © 2023 The Pragmatic Programmers, LLC.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

ISBN-13: 979-8-88865-051-6

Encoded using the finest acid-free high-entropy binary digits.

Book version: B1.0—November 8, 2023

Aristotle's Sum of Parts

`aristotles-sum-of-parts.cpp`

```
#include <iostream>
#include <type_traits>

int main()
{
    char char1 = 1;
    char char2 = 2;

    // True if the type of char1 + char2 is char
    std::cout << std::is_same_v<decltype(char1 + char2), char>;
}
```

Guess the Output



Try to guess what the output is before moving to the next page.

The program displays the following output:

0

Discussion

I know, I know. This is supposed to be a fun book. But sometimes we need to talk about sad topics like this.

As you've figured out by now, the type of `char + char` is unfortunately not `char`. So what is it, then?

For many binary operations, such as `+`, `-`, `*`, `/`, `>`, `==`, `^` and more, the operands must be of the same type. You don't have to ensure that yourself, though; C++ will take care of it. C++ is in fact very eager to implicitly convert between types when you're not looking, and this is one of the more confusing cases. Meet the usual arithmetic conversions.

The usual arithmetic conversions are performed on the operands to these operators to ensure both operands are of the same type. This common type is also the type of the result. For instance, if you add a float and an int, the int is first converted to a float, and the result is a float. And if you add a float and a double, the float is first converted to a double, and the result is a double. Makes sense so far.

The confusion starts when both operands are of integer types, especially if they are of the *same* type. For instance, if you add a `char` and an `int`, the `char` is first converted to an `int`, and the result is an `int`. It still makes sense. But what if you add a `char` and a `short`? Rather than converting the `char` to a `short`, both operands are converted to an `int` (usually, see below). And it gets worse: If you add two `chars`, which are already of the same type, they are still converted to a different type!

Let's see what the usual arithmetic conversions do to our expression `char1 + char2`. Since both operands are of integer types, a process known as integral promotions is performed on both of them. The idea is to convert smaller integer types to `int` or `unsigned int` before performing the arithmetic expression. After converting both operands, we can now add them using regular `int` or `unsigned int` addition and get an `int` or `unsigned int` as a result.

The rule for promoting a smaller integer type is that it can be converted to an `int` if `int` can represent all the values of the smaller type; otherwise, it can

be converted to an unsigned int. Does a char fit in an int? Usually, it does, but that actually depends on the implementation. (I told you this is a sad story.)

A complicating factor here is that the sizes of fundamental types in C++ are not fixed. An implementation can decide that int is 16 bits or 64 bits, and even that char and long are of the same size. The only rules are:

- Each of signed char, short, int, long, and long long is at least as large as the preceding type in the list.
- For each of the types in that list, an unsigned version exists, which has the same size as the signed version.

There are also some minimum sizes defined:

| Type | Minimum width |
|-------------|---------------|
| signed char | 8 |
| short | 16 |
| int | 16 |
| long | 32 |
| long long | 64 |

On a typical 64 bit Linux, Windows or macOS system, int is 32 bits, whereas long is 64 bits on Linux/macOS and 32 bits on Windows. But a conforming implementation could equally well decide that all integer types are 64 bits.

In this puzzle, we're interested in char in particular, but char is curiously missing from all the lists above. What's going on? char is a bit special in that the implementation can choose whether to back it by signed char or unsigned char. In either case, char is a distinct type. So char, signed char, and unsigned char are actually three different types, and whether or not a plain char is signed is implementation-defined!

Remember, we're interested in whether all the values in char fit in an int, to figure out whether our chars would be converted to int or unsigned int. Normally they all fit, so both operands of `char1 + char2` would be converted to int, and the result would also be an int.

But let's say that the implementation has chosen char and int to both be 16 bits. Let's also say that the implementation has chosen that char is unsigned. The range of int is then -2^{16-1} to $2^{16-1}-1$, or -32768 to 32767, inclusive. The range of char is 0 to $2^{16}-1$, or 0 to 65535, inclusive. That is, half the chars don't fit in an int! On these systems, the operands of `char1 + char2` would instead be promoted to unsigned int and the result would also be an unsigned int.

So the type of `char1 + char2` can either be `int` or `unsigned int`. All we know for sure is that it's not `char`.

Further Reading

The Usual Arithmetic Confusions

https://shafik.github.io/c++/2021/12/30/usual_arithmetic_confusions.html

64-Bit Data Models

https://en.wikipedia.org/wiki/64-bit_computing#64-bit_data_models

Fundamental Types

<https://timsong-cpp.github.io/cppwp/std20/basic.fundamental>

Usual Arithmetic Conversions

<https://timsong-cpp.github.io/cppwp/std20/expr.arith.conv>

Integral Promotions

<https://timsong-cpp.github.io/cppwp/std20/conv.prom>