

Exercism: Isogram in C

Eric Bailey

February 22, 2018 ¹

¹ Last updated February 23, 2018

An *isogram* is a word or phrase without a repeating letter.

Contents

| | |
|---|---|
| <i>The is_isogram function</i> | 1 |
| <i>NULL is not an isogram</i> | 2 |
| <i>Determining whether a word is an isogram</i> | 2 |
| <i>Double negation</i> | 3 |
| <i>Include headers</i> | 3 |
| <i>Full Listing</i> | 4 |
| <i>Chunks</i> | 5 |
| <i>Index</i> | 5 |

1a `<* 1a>≡
#include "isogram.h"
<Include headers. 3d>`

`<Define the is_isogram function. 1b>
Root chunk (not used in this document).`

The is_isogram function

To determine a phrase is an isogram, there are two failing conditions we must disprove.

1b `<Define the is_isogram function. 1b>≡
bool is_isogram(const char phrase[])
{
 <If phrase is NULL, return false. 2a>

 <If any letter in phrase appears more than once, return false. 2b>

 <Otherwise, phrase is an isogram. 3c>
}`

This code is used in chunk **1a**.

Defines:

seen, used in chunks **2** and **3**.

NULL is not an isogram

The implementation of the first condition is self-explanatory.

2a *⟨If phrase is NULL, return false. 2a⟩*≡
 if (phrase == NULL)
 return false;

This code is used in chunk 1b.
 Uses NULL 3e.

Determining whether a word is an isogram

To determine whether phrase is an isogram, we need to keep track of which letters we've already **seen**. For that, use an unsigned 32-bit integer.

We'll bitmask this later.

2b *⟨If any letter in phrase appears more than once, return false. 2b⟩*≡
 uint32_t seen = 0;

This definition is continued in chunk 2c.
 This code is used in chunk 1b.
 Uses seen 1b and uint32_t 3f.

Next, iterate through each letter in phrase until NUL, short-circuiting if we've **seen** one before.

2c *⟨If any letter in phrase appears more than once, return false. 2b⟩*+≡
 for (size_t i = 0; phrase[i] != '\0'; ++i) {
⟨Ignore nonalphabetic characters. 2d⟩

⟨If we've already seen the current letter, return false. 2e⟩

⟨Otherwise, mask the bit for the current letter on in seen. 3b⟩
 }

This code is used in chunk 1b.

Since we don't care about nonalphabetic characters, ignore them.

2d *⟨Ignore nonalphabetic characters. 2d⟩*≡
 if (!isalpha(phrase[i]))
 continue;

This code is used in chunk 2c.
 Uses isalpha 3d.

To determine, case-insensitively, if we've seen a letter already, convert it to uppercase and subtract 'A', e.g 'a' - 'A' == 0 and 'e' - 'A' == 4. **Mask the corresponding bit on** and store the result.

2e *⟨If we've already seen the current letter, return false. 2e⟩*≡
 uint32_t bit_mask = 1 << (toupper(phrase[i]) - 'A');

This definition is continued in chunk 3a.
 This code is used in chunk 2c.
 Defines:
 bit_mask, used in chunk 3.
 Uses toupper 3d and uint32_t 3f.

If the result of `seen` bitwise AND `bit_mask` is nonzero, we've seen this letter before and phrase is not an isogram.

3a $\langle \text{If we've already seen the current letter, return false. 2e} \rangle + \equiv$
`if (seen & bit_mask)`
`return false;`

This code is used in chunk 2c.
 Uses `bit_mask 2e` and `seen 1b`.

If this is a new letter, mask its bit on in `seen` and proceed to the next letter.

3b $\langle \text{Otherwise, mask the bit for the current letter on in seen. 3b} \rangle \equiv$
`seen |= bit_mask;`

This code is used in chunk 2c.
 Uses `bit_mask 2e` and `seen 1b`.

Double negation

If phrase is not not an isogram, then it is an isogram.

3c $\langle \text{Otherwise, phrase is an isogram. 3c} \rangle \equiv$
`return true;`

This code is used in chunk 1b.

***2.14.** $\vdash \sim(\sim p) \supset p$
 (Whitehead and Russell)

Include headers

From `ctype.h` import the functions `isalpha`, to determine whether a character is alphabetic, and `toupper`, to convert a letter to uppercase.

3d $\langle \text{Include headers. 3d} \rangle \equiv$
`#include <ctype.h>`

This definition is continued in chunk 3.
 This code is used in chunk 1a.
 Defines:
`isalpha`, used in chunk 2d.
`toupper`, used in chunk 2e.

From `stddef.h` import the `NULL` macro.

3e $\langle \text{Include headers. 3d} \rangle + \equiv$
`#include <stddef.h>`

This code is used in chunk 1a.
 Defines:
`NULL`, used in chunk 2a.

From `stdint.h` import the 32-bit unsigned integer type, `uint32_t`.

3f $\langle \text{Include headers. 3d} \rangle + \equiv$
`#include <stdint.h>`

This code is used in chunk 1a.
 Defines:
`uint32_t`, used in chunk 2.

Full Listing

Listing 1: `isogram.h`

```
1  #ifndef ISOGRAM_H
2  #define ISOGRAM_H
3
4  #include <stdbool.h>
5
6  bool is_isogram(const char phrase[]);
7
8  #endif
```

Listing 2: `isogram.c`

```
1  #include "isogram.h"
2  #include <ctype.h>
3  #include <stddef.h>
4  #include <stdint.h>
5
6
7  bool is_isogram(const char phrase[])
8  {
9      if (phrase == NULL)
10         return false;
11
12     uint32_t seen = 0;
13
14     for (size_t i = 0; phrase[i] != '\0'; ++i) {
15         if (!isalpha(phrase[i]))
16             continue;
17
18         uint32_t bit_mask = 1 << (toupper(phrase[i]) - 'A');
19         if (seen & bit_mask)
20             return false;
21
22         seen |= bit_mask;
23     }
24
25     return true;
26 }
```

Chunks

⟨* 1a⟩ [1a](#)
 ⟨Define the `is_isogram` function. 1b⟩ [1a](#), [1b](#)
 ⟨If `phrase` is `NULL`, return false. 2a⟩ [1b](#), [2a](#)
 ⟨If any letter in `phrase` appears more than once, return false. 2b⟩ [1b](#), [2b](#), [2c](#)
 ⟨If we've already seen the current letter, return false. 2e⟩ [2c](#), [2e](#), [3a](#)
 ⟨Ignore nonalphabetic characters. 2d⟩ [2c](#), [2d](#)
 ⟨Include headers. 3d⟩ [1a](#), [3d](#), [3e](#), [3f](#)
 ⟨Otherwise, `phrase` is an isogram. 3c⟩ [1b](#), [3c](#)
 ⟨Otherwise, mask the bit for the current letter on in `seen`. 3b⟩ [2c](#), [3b](#)

Index

`bit_mask`: [2e](#), [3a](#), [3b](#)
`isalpha`: [2d](#), [3d](#)
`NULL`: [2a](#), [3e](#)
`seen`: [1b](#), [2b](#), [3a](#), [3b](#)
`toupper`: [2e](#), [3d](#)
`uint32_t`: [2b](#), [2e](#), [3f](#)

References

Alfred North Whitehead and Bertrand Russell. *Principia mathematica*.
 Cambridge University Press, 1910-. URL <http://name.umd1.umich.edu/AAT3201.0001.001>.