Exercises — Variant

version **#7be580532266ed398481e31366afcc24b1950c2a**

The way is lit. The path is clear.
We require only the strength to follow it.

Assistants C/Unix 2023 <assistants@tickets.assistants.epita.fr>

Contents

*https://intra.assistants.epita.fr

File Tree

```
variant/
├─ variant.c   (to submit)
└─ variant.h
```

Authorized functions : You are only allowed to use the following functions

- printf(3)
- strcmp(3)

Authorized headers : You are only allowed to use the functions defined in the following headers

- err.h
- errno.h
- assert.h
- stddef.h
- stdbool.h

Compilation : Your code must compile with the following flags

- -std=c99 -pedantic -Werror -Wall -Wextra -Wvla

Main function : None

1 Goal

In this exercise you will implement a variant using a tagged union. To do so, you need to write a structure named `variant` containing:

- An union, named `type_any`
- An enum representing the field currently held by the union, named `type`

The variant must be able to store the following types:

- `int`
- `float`
- `char`
- `const char *`

2 Display

Write a function that will print the content of a variant on the standard output, followed by a line feed (`\n`).

For instance, if the variant contains the following integer value: `12`, it should display `12\n`.

```
void variant_display(const struct variant *e);
```

3 Equal

Write a function that returns true if two variants have the same type **and** the same content.

```
bool variant_equal(const struct variant *left, const struct variant *right);
```

Note that you must include `stdbool.h` to have booleans.

4 Find

Write a function that will look for an element in a variant array. The function will return the index of the first matched element if found, otherwise it returns -1.

```
int variant_find(const struct variant *array, size_t len, enum type type,
                 union type_any value);
```

5 Sum

Write a function that returns the sum of all numeric elements in a variant array (`int` and `float`)

```
float variant_sum(const struct variant *array, size_t len);
```

The way is lit. The path is clear. We require only the strength to follow it.