ABC by Example: Data Structure Alignment …

… and how it affects the size of a struct

Data Structure Alignment

- **For Primitive Types**
  - Values need to be stored at addresses that are multiples of their size.

Data Structure Alignment

- **For Primitive Types**
  - Values need to be stored at addresses that are multiples of their size.
- **For Arrays**
  - The alignment requirement of an array is the same as the alignment requirement of its elements.

Data Structure Alignment

- **For Primitive Types**
  - Values need to be stored at addresses that are multiples of their size.
- **For Arrays**
  - The alignment requirement of an array is the same as the alignment requirement of its elements.
- **For Structs**
  - Each member has to be aligned according to its type’s alignment requirement (might require "padding").
  - The total size of the struct has to be "padded" to ensure proper alignment when used in arrays or other data structures.

```
@ <stdio.hdr>

struct Foo
{
    a: char;
    b: int;
};

fn main()
{
    printf("sizeof(char) = %zu\n", sizeof(char));
    printf("sizeof(int) = %zu\n", sizeof(int));
    printf("sizeof(Foo) = %zu\n", sizeof(Foo));

    local foo: array[3] of Foo;
    printf("sizeof(foo) = %zu\n", sizeof(foo));
    printf("sizeof(foo) / sizeof(Foo) = %zu\n", sizeof(foo) / sizeof(Foo));
}
```

```
@ <stdio.hdr>

struct Foo
{
    b: int;
    a: char;
};

fn main()
{
    printf("sizeof(char) = %zu\n", sizeof(char));
    printf("sizeof(int) = %zu\n", sizeof(int));
    printf("sizeof(Foo) = %zu\n", sizeof(Foo));

    local foo: array[3] of Foo;
    printf("sizeof(foo) = %zu\n", sizeof(foo));
    printf("sizeof(foo) / sizeof(Foo) = %zu\n", sizeof(foo) / sizeof(Foo));
}
```

```
@ <stdio.hdr>

struct Foo
{
    a: char;
    c: char;
    b: int;
};

fn main()
{
    printf("sizeof(char) = %zu\n", sizeof(char));
    printf("sizeof(int) = %zu\n", sizeof(int));
    printf("sizeof(Foo) = %zu\n", sizeof(Foo));

    local foo: array[3] of Foo;
    printf("sizeof(foo) = %zu\n", sizeof(foo));
    printf("sizeof(foo) / sizeof(Foo) = %zu\n", sizeof(foo) / sizeof(Foo));
}
```

```
@ <stdio.hdr>

struct Foo
{
    a: char;
    b: int;
    c: char;
};

fn main()
{
    printf("sizeof(char) = %zu\n", sizeof(char));
    printf("sizeof(int) = %zu\n", sizeof(int));
    printf("sizeof(Foo) = %zu\n", sizeof(Foo));

    local foo: array[3] of Foo;
    printf("sizeof(foo) = %zu\n", sizeof(foo));
    printf("sizeof(foo) / sizeof(Foo) = %zu\n", sizeof(foo) / sizeof(Foo));
}
```

```
@ <stdio.hdr>

struct Foo
{
    a: u32;
    b: u64;
    c: u16;
};

fn main()
{
    printf("sizeof(u32) = %zu\n", sizeof(u32));
    printf("sizeof(u64) = %zu\n", sizeof(u64));
    printf("sizeof(u16) = %zu\n", sizeof(u16));

    local foo: array[3] of Foo;
    printf("sizeof(foo) = %zu\n", sizeof(foo));
    printf("sizeof(foo) / sizeof(Foo) = %zu\n", sizeof(foo) / sizeof(Foo));
}
```

```
@ <stdio.hdr>

struct Foo
{
    a: array[2] of u32;
    b: u16;
};

fn main()
{
    printf("sizeof(char) = %zu\n", sizeof(u16));
    printf("sizeof(int) = %zu\n", sizeof(u32));
    printf("sizeof(Foo) = %zu\n", sizeof(Foo));

    local foo: array[3] of Foo;
    printf("sizeof(foo) = %zu\n", sizeof(foo));
    printf("sizeof(foo) / sizeof(Foo) = %zu\n", sizeof(foo) / sizeof(Foo));
}
```