Johannes Kepler University Linz
LIT – Secure and Correct Systems Lab
S. Rass · S. Ahmad · M. Schwaighofer ·
K. Feichtinger · D. Zaitsev

Excercise Sheet 5

Warmup (optional): Determine and correct the errors in the following definitions.

a)
```
struct Date {
    short month, day, year;
    int print(void);
}
```

b)
```
struct Item {
    float x = 0;
    long cap = 50000L;
    Item *next;
};
```

c)
```
struct Fields {
    unsigned a : 8;
    float x : 8;
    long n : 24;
};
```

d)
```
struct Member {
    char name[64];
    char *info;
    struct Member base;
};
```

Exercise 5.1. Write the declarations for the following, in a file `source.c`:

a) a global variable `var` of type `long`, which is defined in its own file, distinct from `source.c`

b) a global function `gfunc()` declared in its own file, distinct from `source.c`

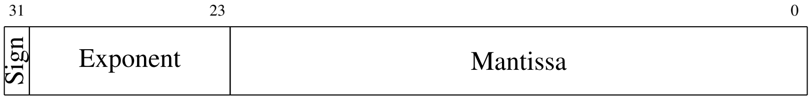
c) a static function `sfunc()` declared in the same source file `source.c`

Both functions have no parameters and a return value of type `double*`.

Exercise 5.2. The memory allocation of a `float` variable is to be analyzed. The representation of a floating point number x is always based on a decomposition into a sign v, a mantissa m and an exponent exp to the base 2:

$$x = v \cdot m \cdot 2^{exp}$$

In the common IEEE format, the mantissa has a value greater than or equal to 1 and less than 2. Only for $x = 0$ the mantissa has the value 0. The 32 bits of a `float` variable are usually divided as follows:



a) Define a `struct` type with name `bits` that has three unsigned variables for the `mantissa`, the `exponent` and the `sign`. This is possible if the type `int` has at least the size of `float`.

The number of bits for the mantissa can be taken from the header file `float.h`. For this the constant `FLT_MANT_DIG` is defined there. The first bit of the mantissa is always 1 and is not stored. The width of the bit field is therefore only `FLT_MANT_DIG - 1`. The `sign` always occupies one bit. The remaining bits are used for the exponent. For a better overview it is useful to define symbolic constants for the widths of the bit fields, and use these constants to tell the compiler about the number of bytes to use for each field (see slide 3-13).

b) Then define a `union` with three elements: a `float` element, an `unsigned` element and a `struct` element that will represent the bit fields (in this order). This allows direct access to the exponent and mantissa, for example. Directly assign the floating point number to your variable of this type union, and try accessing the struct

A sample output of the program:

```
** Sign, exponent and mantissa of a float-variable ***

Number of bits for the mantissa: 23
number of bits for the exponent: 8

Enter a floating point number:5.2
bit pattern: 0100 0000 1010 0110 0110 0110 0110 0110
sign: +
exponent: 2
mantissa: 1.300000
```

c) Declare two variables of your self-defined types and try to assign a float represented by such a structure to the other variable. So, suppose that you have some floating point number stored in a `union` variable `f1` (of type `FloatLong` in the template; see below), and another such `union` variable named `f2`. Does the assignment `f1=f2` work like it would do for variables of type `float`? Does the assignment work if you assign the `struct` members of the variables only?

Hints:

- You should use the provided code template `float-bitpattern-template.c` to fill in your definitions of the struct and union types. Please note that the naming used in the template is consistent with the naming as above, so if you use different names for the fields, be sure to have the naming consistent everywhere.
- The exponent `exp` is stored with a shift (bias) of 127, namely as a non-negative number `exp+127`.
- The value in the bit field of the mantissa (see the figure above) represents the decimal places. Therefore, the corresponding value results by division with `2M = (1<<M)`, if `M` is the width of the bit field.