Bit fields and inline functions

Microcontrollers

Stefan Huber
www.sthu.org

Dept. for Information Technologies and Digitalisation
FH Salzburg

Winter 2024

Bit fields

Bit fields in C

Programming networking protocols or close to hardware often requires us to interpret data bit-wise rather than byte-wise.

- For instance, the IPv4 header starts with 4 bits for the version, 4 bits of the header length, 6 bits ToS, and so on. Another example is the `SREG` register of the ATmega32.
- The C and C++ programming languages defines so-called bit fields to model such data layouts as a sequence of bits. [cppref-c-bitfields]
- Syntactically, a bit field is a struct[1] member of integer type plus a declaration of how many bits are spent.

```c
typedef struct {
    unsigned int version : 4;
    unsigned int ihl : 4;
    unsigned int tos : 6;
    /* and more */
} ipv4_header;
```

1 Actually, also union members can be defined as bit fields.

Bit fields in C

- Adjacent bit fields are *usually* packaged together, but it is implementation-defined.
- If field size is larger than type then value is limited to type's size or a compiler error is raised.

```c
typedef struct {
    unsigned int a : 4;
    unsigned int   : 1;    /* Nameless field for padding. */
    unsigned int b : 2;
    unsigned int   : 0;    /* Padding to next allocation unit boundary. */
} mybitfield;
```

Bit fields in C

- Adjacent bit fields are *usually* packaged together, but it is implementation-defined.
- If field size is larger than type then value is limited to type's size or a compiler error is raised.

```c
typedef struct {
    unsigned int a : 4;
    unsigned int   : 1;    /* Nameless field for padding. */
    unsigned int b : 2;
    unsigned int   : 0;    /* Padding to next allocation unit boundary. */
} mybitfield;
```

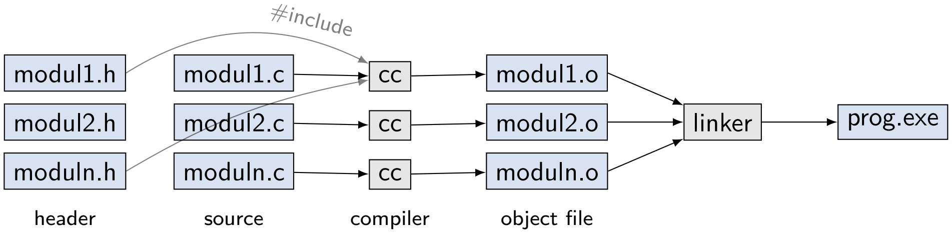
Technical details:

- A signed field means two's complement. A signed field of size 1 can only have values 0 and -1.
- The standard defines `int`, `signed int`, `unsigned int`, `_Bool` as base types, but implementations may support additional types, such as `char`, `short`, `long` and its signed and unsigned counterparts.
- It is implementation-defined whether `int` is signed or unsigned. For bit fields `int` has different meaning than `signed int`!

Translation units and linking

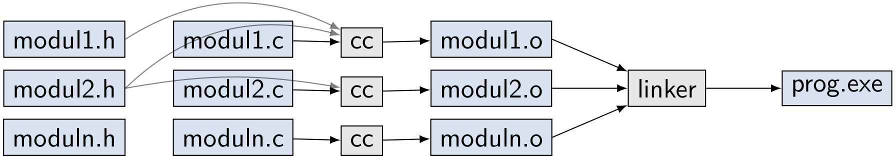
Translation units and linking

- A translation unit is (typically) a `.c` file that is compiled to an object file.
- The object files are then linked to a final binary (executable, library, ATmega32 program, . . .)



Functions and symbols

- A function definition exports a symbol.
  - If other translation units call this function then the linker looks for this symbol.
  - The function declaration in the header file determines the symbol name.
  - No two translation units can export the same symbol!



Example: Implement a function `f()`

- Say, `modul2.h` declares it and `modul2.c` defines it.
- Only `modul2.o` has the symbol for `f()`.
- But we could call it in `modul1.c`, by including `module2.h` and therefore knowing its symbol name.

Inline functions

Inline functions

With C99 we can give the compiler a *hint* to inline a function [cppref-c-inline]:

- When calling a function, instead placing a function call the function's body is placed.
- Inline functions are like macros, but with types.
- For small functions this is an optimization technique to save the function call costs. However, it is only a hint to the compiler, we cannot force it.

Dilemma

- Where to define the inline function?
- Definition (now only declaration) must be known for all translation units for inlining.
  $\rightarrow$ Definition in header file.
- But only one translation unit must export a symbol.

Answer: Declare the inline function with extern keyword in one translation unit (.c file).

Complete inline function demo

The header file `geom.h`:

```c
#ifndef geom_h_Epai3ohkaevei0ea
#define geom_h_Epai3ohkaevei0ea

#include <math.h>

inline double sq(double x) {
    return x * x;
}

inline double norm(double x, double y) {
    return sqrt(sq(x) + sq(y));
}

#endif
```

The implementation file `geom.c`:

```c
#include "geom.h"

extern inline double sq(double x);
extern inline double norm(double x, double y);
```

Static and inline functions

- If you declare a function as `static` then it is only local to this translation unit.
  - No symbol is exported. The namespace is not polluted.
  - The function cannot be called from another translation unit.
- The definition of an inline function does not automatically export a symbol.
  - So `inline` is a bit like `static`, but in header files.
  - However, an external definition *must* exist. Hence, we have to explicitly add an export definition of the function in *one* translation unit using the `extern` keyword.

References I

[cppref-c-bitfields] *cppreference.com: bit fields*. URL: https://en.cppreference.com/w/c/language/bit_field.

[cppref-c-inline] *cppreference.com: inline function specifier*. URL: https://en.cppreference.com/w/c/language/inline.