【C】Day3

Course	Advanced C
Study Date	@April 4, 2022

【Ch6】Structures

6.9 Bit-fields

When storage space is at a premium, it may be necessary to pack several objects into a single machine word; one common use is a set of single-bit flags in applications like compiler symbol tables.

The usual way this is done is to define a set of "masks" corresponding to the relevant bit positions, as in

```
#define KEYWORD 01
#define EXTERNAL 02
#define STATIC 04
```

or

```
enum { KEYWORD = 01, EXTERNAL = 02, STATIC = 04 };
```

The numbers must be powers of two.

Certain idioms appear frequently:

```
flags |= EXTERNAL | STATIC;
```

turns on the `EXTERNAL` and `STATIC` bits in flags, while

```
flags &= ~(EXTERNAL | STATIC);
```

turns them off, and

```
if((flags & (EXTERNAL | STATIC)) == 0)
```

is true if both `EXTERNAL` and `STATIC` are off.

As an alternative, C offers the capability of defining and accessing fields within a word directly rather than by bitwise logical operators.

A bit-field, or field for short, is a set of adjacent bits within a single implementation-defined storage unit that we call a “word.”

```
struct {
  unsigned int is_keyword : 1;
  unsigned int is_extern : 1;
  unsigned int is_static : 1;
} flags;
```

This defines a variable called `flags` that contains three 1-bit fields. The number following the colon represents the field width in bits.

The fields are declared `unsigned int` to ensure that they are unsigned quantities.

Individual fields are referenced in the same way as other structure members:

```
flags.is_keyword = 1;
```

to turn on the bits.

Fields need not to be named.