C Programming
Structs

Adam Sampson (based on material by Greg Michaelson)
School of Mathematical and Computer Sciences
Heriot-Watt University

Structures

- A finite sequence of elements
  - Like an object, but without methods
- Elements (**fields**) have potentially different types
- Each element is identified by a field name
- e.g. Person contains:
  - name (String)
  - height (Float)
  - age (Integer)

Structure type definition

`struct` *name* `{` *type*$_1$ *name*$_1$`;` … *type*$_N$ *name*$_N$`; };`

- This defines a **new type**, `struct` *name*
- It does **not** allocate space for an object

`struct` *name1* *name2*`;`

- Allocates space for an instance of `struct` *name1* called *name2*

typedef

`typedef oldtype newtype;`

- Defines an additional name for an existing type
- A **type alias** – not a new, different type

```
typedef float length;

typedef struct person person_t;

typedef struct {
    const char *name;
    int age;
} person_t;
```

You can include a `struct` definition inside a `typedef` or variable declaration

Structure layout

`struct {` *`type1 name1`*`; … `*`typeN nameN`*`; } `*`name`*`;`

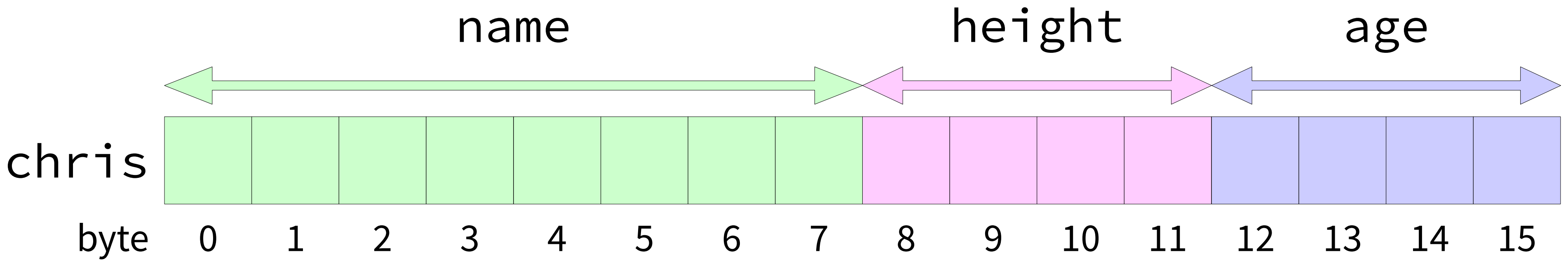
- Size: at least `sizeof(`*`type1`*`) + … + sizeof(`*`typeN`*`)`
- May be bigger because of **padding** – extra space inserted by the compiler to align fields efficiently in memory
- Fields stored in memory from left to right
- *`name`* is not the same as *`&name`*
  - *`&name`* is a pointer – the **address** of the first byte
  - *`name`* is the **value** of the structure

`person.c`

Person structure

Structure declaration

```
struct person {
    const char *name;
    float height;
    int age;
};
struct person chris;
```



`freq.c`

Frequency count

Frequency count

- Count how often each distinct character appears in a file
- Array of `structs` to hold character and count
- For each character in file:
  - If there is already a `struct` in the array for the next character: increment count within that existing `struct`
  - Else (we haven't seen this character before): create a new `struct` for that character with the count `1`, and add it to the array

Frequency count

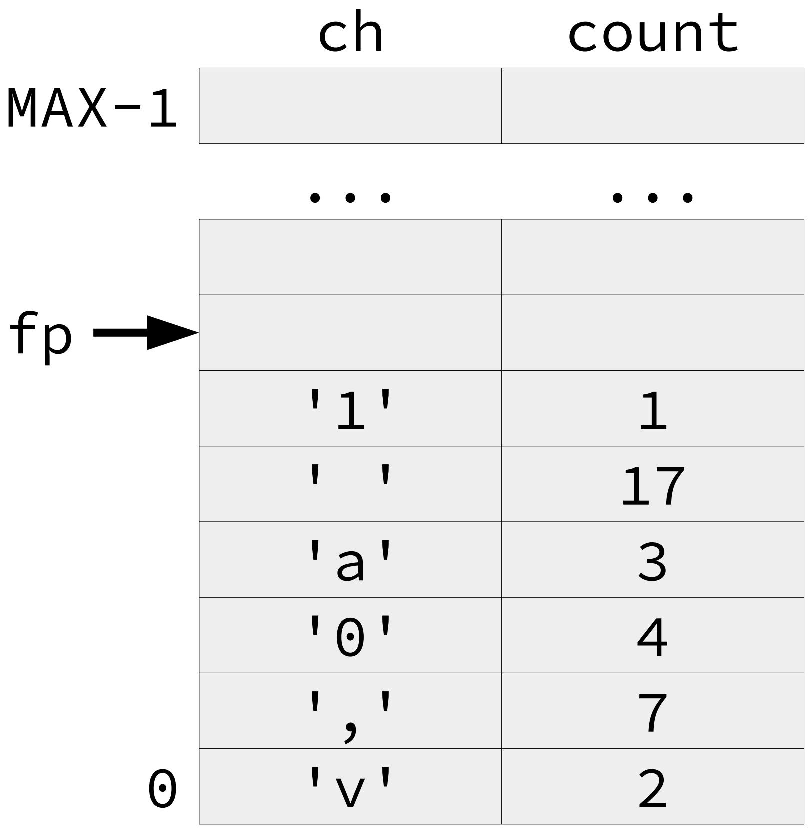
```
#include <stdio.h>
#include <stdlib.h>

#define MAX 256

struct freq {
    int ch;
    int count;
};

struct freq f[MAX];
int fp;
```

- `fp` is index of next free entry in `f`

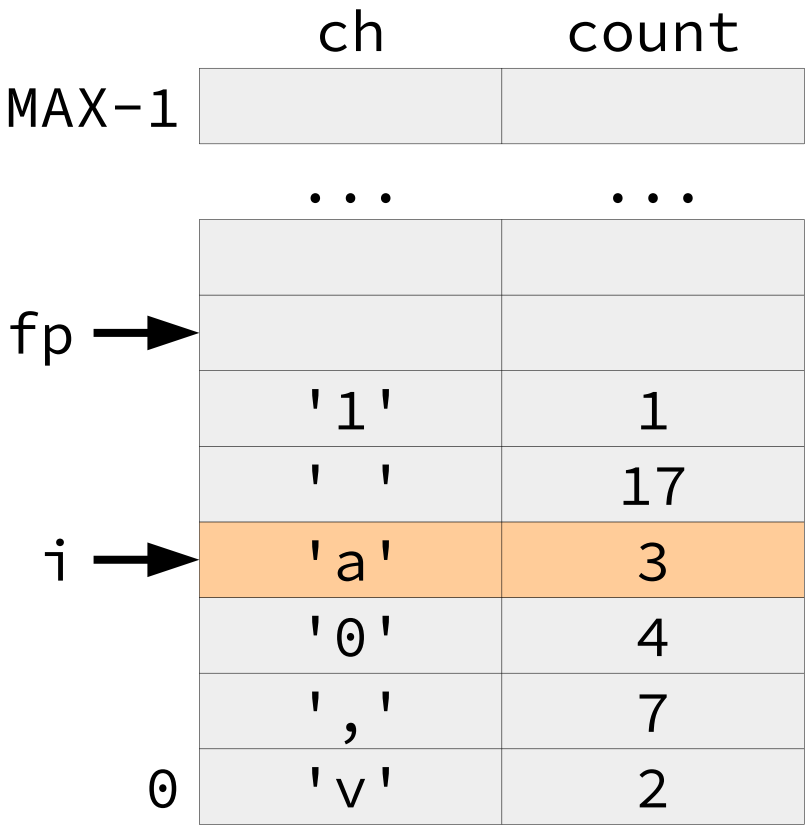


Frequency count

```
void incFreq(int ch)
{
    for (int i = 0; i < fp; i++) {
        if (f[i].ch == ch) {
            f[i].count++;
            return;
        }
    }
```

- Search `f` for entry for `ch`
- If found, increment `count` and return

- If `ch` was `'a'`...

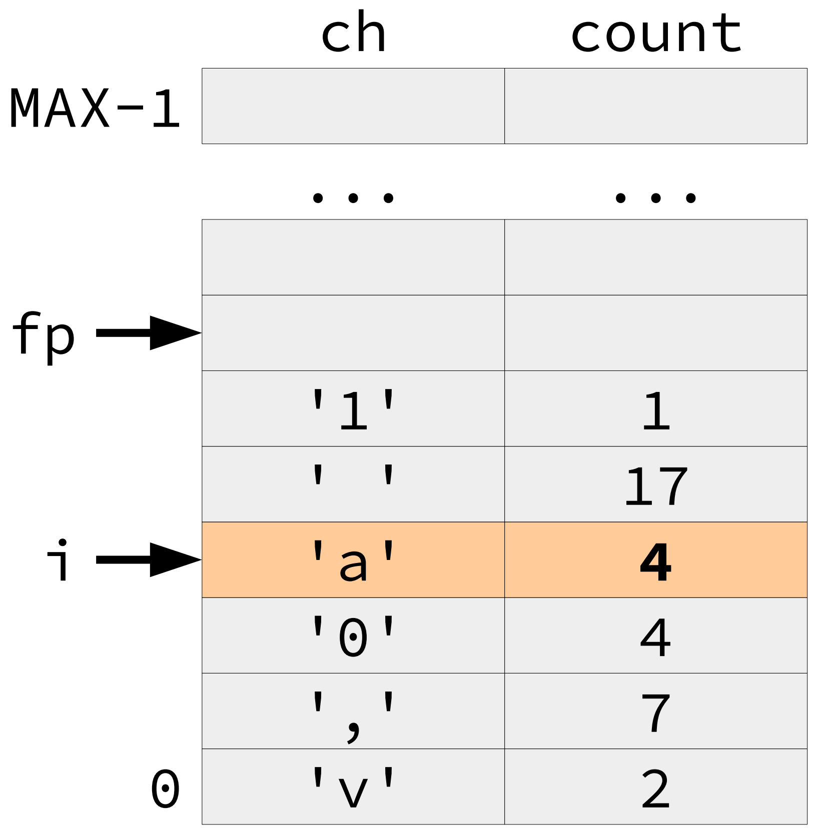


Frequency count

```
void incFreq(int ch)
{
    for (int i = 0; i < fp; i++) {
        if (f[i].ch == ch) {
            f[i].count++;
            return;
        }
    }
```

- Search `f` for entry for `ch`
- If found, increment `count` and return

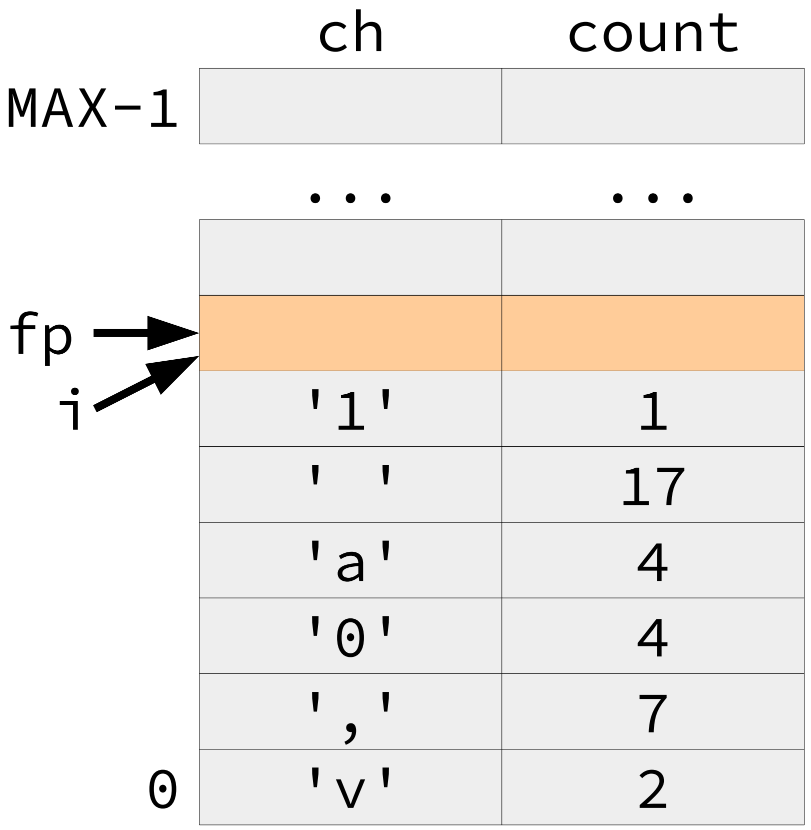
- If `ch` was `'a'`…



Frequency count

```
    if (fp == MAX) {
        printf("error\n");
        exit(1);
    }
    f[fp].ch = ch;
    f[fp].count = 1;
    fp++;
}
```

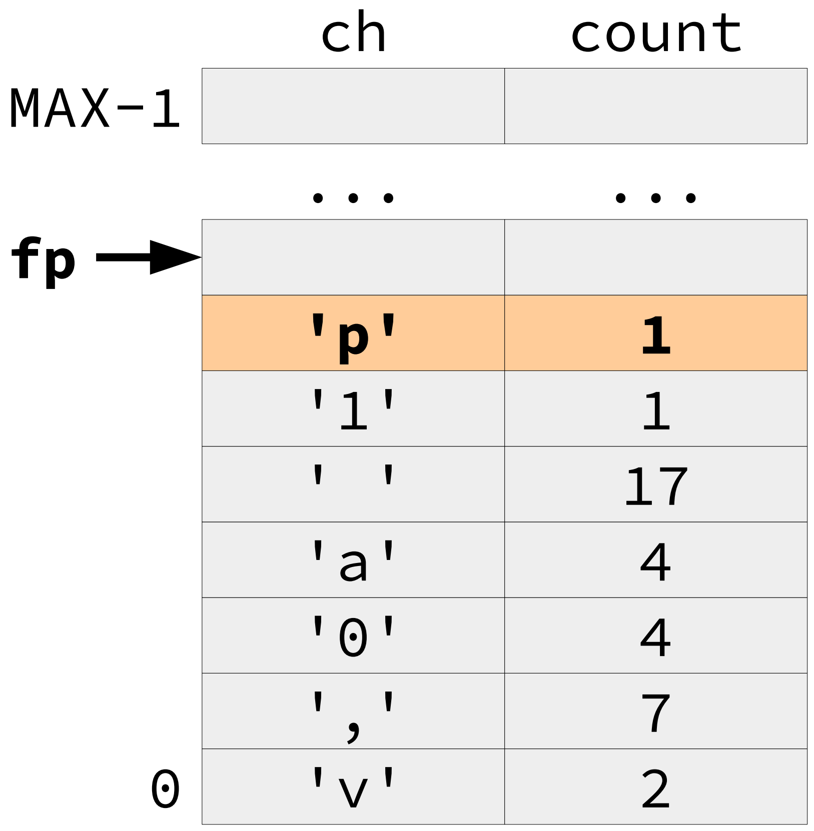
- If the loop doesn't find ch...
- Make sure f isn't full
- Fill in f[fp] with ch and 1
- Increment fp



Frequency count

```
    if (fp == MAX) {
        printf("error\n");
        exit(1);
    }
    f[fp].ch = ch;
    f[fp].count = 1;
    fp++;
}
```

- If the loop doesn't find ch...
- Make sure f isn't full
- Fill in f[fp] with ch and 1
- Increment fp



Frequency count

```
void showFreq(void)
{
    for (int i = 0; i < fp; i++) {
        printf("%c : %d\n", f[i].ch, f[i].count);
    }
}
```

Frequency count

```c
int main(int argc, char *argv[])
{
    FILE *fin;
    if ((fin = fopen(argv[1], "r")) == NULL) {
        printf("can't open %s\n", argv[1]);
        return 1;
    }
    fp = 0;
    while (1) {
        int ch = getc(fin);
        if (ch == EOF)
            break;
        incFreq(ch);
    }
```

```c
    fclose(fin);
    showFreq();
    return 0;
}
```

Frequency count

```
$ ./freq freq.c
# : 3
i : 48
n : 36
c : 32
l : 8
u : 10
d : 8
e : 27
  : 185
< : 4
s : 10
t : 31
o : 12
. : 9
h : 22
> : 2

 : 54
b : 1
f : 35
M : 4
A : 4
X : 4
2 : 1
5 : 2
r : 23
q : 6
{ : 9
; : 29
} : 9
[ : 10
] : 10
p : 14
F : 6
( : 21
) : 21
= : 19
0 : 5
w : 5
+ : 4
1 : 7
" : 8
m : 2
a : 10
% : 4
\ : 3
, : 6
x : 2
: : 1
g : 6
* : 3
v : 3
I : 1
L : 3
E : 2
N : 1
U : 1
' : 1
! : 1
O : 1
```

Newline
'\n'