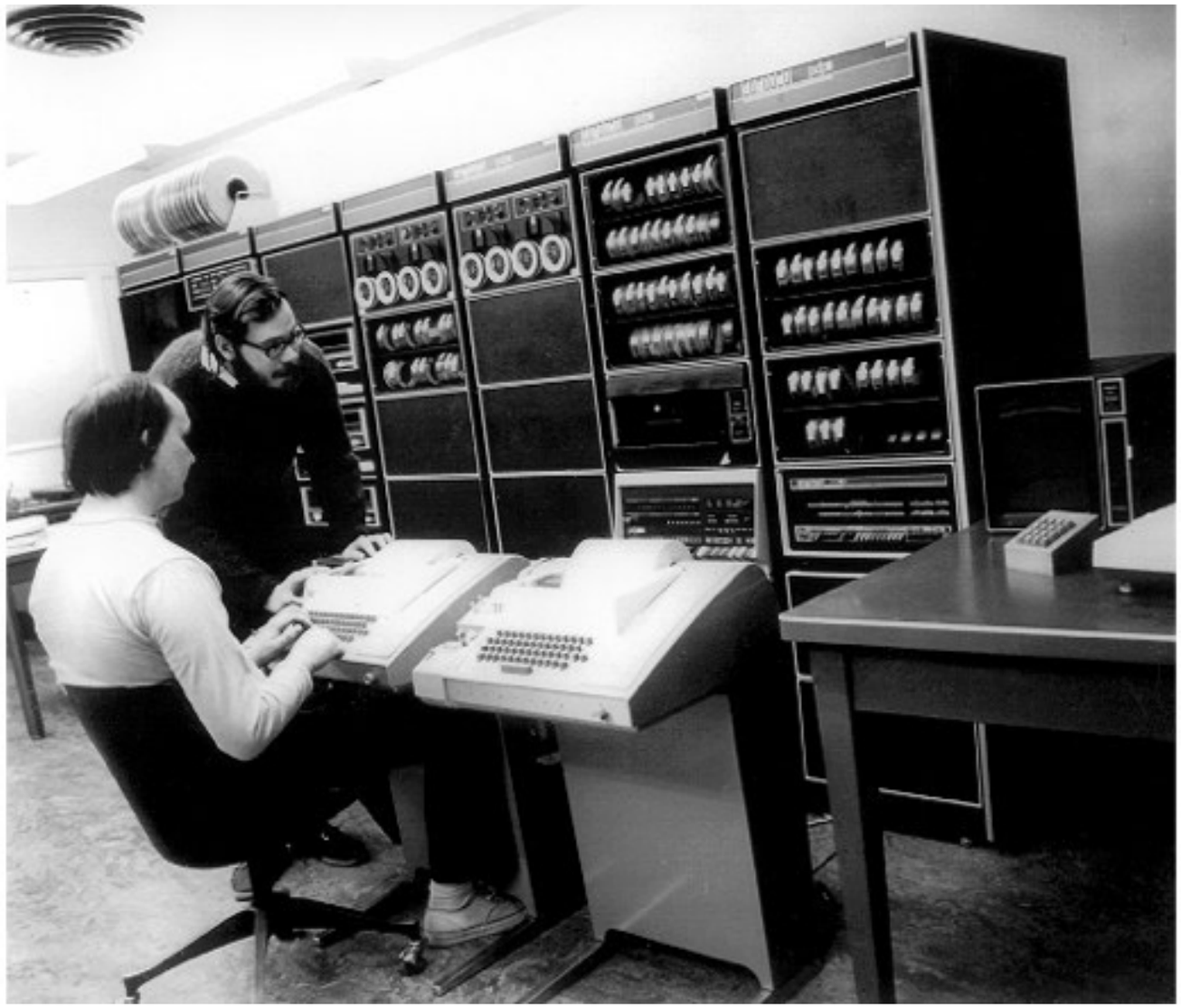
Type Misc

"to get a deeper understanding of the language"



Deep C - a 3 day course
Jon Jagger & Olve Maudal

typedef

- defines a new name (an alias) for an *existing* type
- does *not* create a new type

```
typedef unsigned int size_t;
```

```
void f(unsigned int variable);
void f(size_t variable);
```

equivalent declarations

```
void f(unsigned int variable);
...
void f(size_t variable)
{
    ...
}
```

declared without typedef
defined with typedef

C was designed so that the syntax of use mirrors the syntax of declaration

```
        int identifier;
            ↕      ↕
typedef int identifier;
```

```
       int *pointer = &variable;
           ↕  ↕
int copy = *pointer;
           ↕  ↕
           *pointer = 42;
```

```
int days_in_month[12];
          ↕       ↕
 ...days_in_month[at]...
```

```
void func(int a, int b);
       ↕      ↕      ↕
     func(    4,     2);
```

typedef

- aids portability and expresses intention

```
void eg(void)
{
    struct wibble buffer[4096];
    unsigned int count = sizeof(buffer);
    ...
}
```

```
#include <stddef.h>

void f(void)
{
    struct wibble buffer[4096];
    size_t count = sizeof(buffer);
    ...
}
```

typedef

c11

- c99 does not allow duplicate typedefs
- c11 does

```
typedef struct date date;
typedef struct date date;
```

```
typedef struct date date;
typedef struct date date;
```

c11

gcc: -std=c11

clang: -x c -std=c11

enum

- an enum definition introduces a new type
- and a sequence of enumerators
- each enumerator is *not* scoped to its enum

```
enum suit
{
    spades, hearts, diamonds, clubs
};
```

```
enum stones
{
    emeraldes, diamonds, sapphires, ...
};
```

enum typedef?

bad alternative

don't use a different tag name

```
enum suit_tag { ... };
typedef enum suit_tag suit;
suit trumps = clubs;
```

```
enum suit { ... };
typedef enum suit suit;
suit trumps = clubs;
```

```
enum suit { clubs, diamonds, hearts, spades };
enum suit trumps = clubs;
```

kernel style

enum conversions

- an enum is a thinly wrapped integer – not type safe
- *any* int value can be converted to an enum
- an enum can be converted to an int

```
const char * suit_name(suit s)
{
    switch (s)
    {
    case clubs    : return "clubs";
    case diamonds : return "diamonds";
    case hearts   : return "hearts";
    case spades   : return "spades";
    default       : return NULL; // can happen
    }
}
```

```
int main(void)
{
    suit trumps = (suit)42;
    int value = (int)trumps;
    printf("%s\n", suit_name(trumps));
}
```

neither cast is required

?

anonymous enums

- useful for [designators]

```
enum {
  january,
  february,
  ...
  november,
  december
};
```

```
const int days_in_month[] =
    {
        [january] = 31,
        [february] = 28,
        ...
        [november] = 30,
        [december] = 31
    };
```

c99

anonymous enums

- alternative to #define for array sizes

```
#define MAX_SIZE (1024)

char buffer[MAX_SIZE];
```

```
enum { MAX_SIZE = 1024 };

char buffer[MAX_SIZE];
```

```
enum { max_size = 1024 };

char buffer[max_size];
```

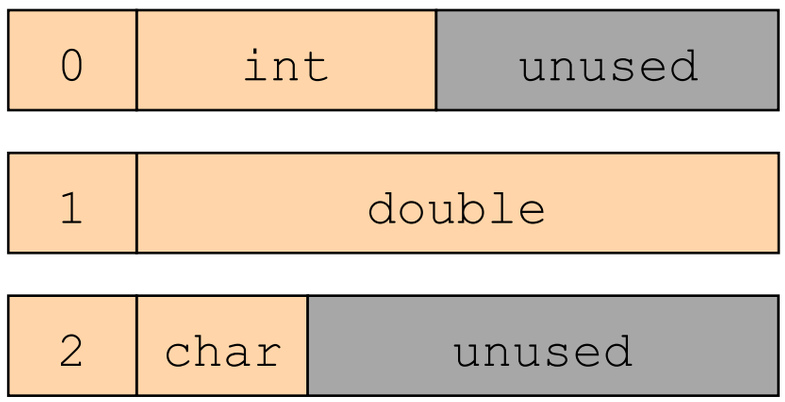
unions

- the members of a union overlay one another
- typically used to save memory
- often accompanied by an enum discriminator

```
enum descrim { int_u=0, double_u=1, char_u=2 };
```

```
union jack
{
    int i;
    double d;
    char c;
};

struct ural
{
    enum descrim is_a;
    union jack value;
};
```



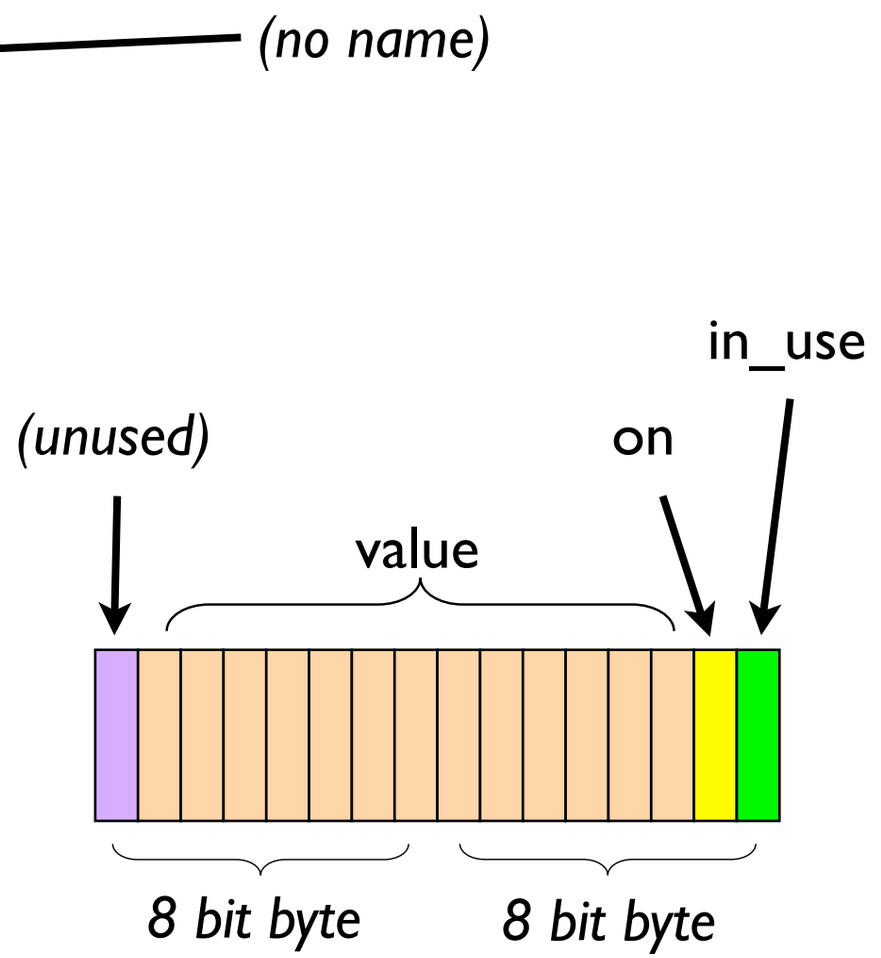
bit fields

- you can use bit fields to control memory allocation
- right down to the bit level
- compiler dependent; not portable
- you cannot take the address of a bit field

```
struct fields
{
    unsigned int : 1;
    unsigned int value : 13;
    unsigned int on : 1;
    unsigned int in_use : 1;
};
```

```
struct fields widget;
...
if (widget.in_use)
    ...

widget.value = 37;
```



summary

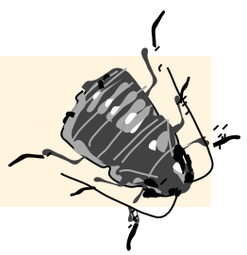
- typedef does not create a new type
- typedefs help portability and help express intention
- enumerators are not scoped
- enums are thinly wrapped integers
- anonymous enums can be useful!
- unions occasionally useful
- bitfields are not portable
- _Alignas in c11
- _Alignof in c11

_Alignas

c11

- a new declaration alignment-specifier
- _Alignas(type-name)
- _Alignas(constant-expression)
- #include <stdalign.h> provides alignas macro
- #include <stddef.h> provides max_align_t
- #include <stdlib.h> provides aligned_alloc()

```
char buffer[sizeof(double)];
double * ptr = (double*)buffer;
```



```
#include <stdalign.h>

alignas(double) char buffer[sizeof(double)];
double * ptr = (double*)buffer;
```

```
#include <stdalign.h>
#include <stddef.h>

alignas(max_align_t) char buffer[sizeof(double)];
double * ptr = (double*)buffer;
```

_Alignof

c11

- _Alignof(type-name) is the size_t alignment of type-name
- valid alignment values are always integral powers of 2
- char has the weakest alignment requirement
- #include <stdalign.h> provides alignof macro

```c
#include <stdalign.h>
#include <stddef.h>
#include <stdio.h>

int main(void)
{
    printf("%zu\n", alignof(char));
    printf("%zu\n", alignof(short));
    printf("%zu\n", alignof(int));
    printf("%zu\n", alignof(long));
    printf("%zu\n", alignof(void*));
    printf("%zu\n", alignof(float));
    printf("%zu\n", alignof(double));
    printf("%zu\n", alignof(max_align_t));
}
```

```
1
2
4
8
8
4
8
16
```

eg