Pointer+,
struct, union, enum

字符串回顾

- `char * array[N];`

- selectsort排序字符串数组
  - strcmp
  - `int __cdecl strcmp(const char *_Str1,const char *_Str2);`
  - 思考：每一轮排序swap的是什么?
  - pointer-array-sort.c

比较指针数组和字符二维数组

- `char * array[N];`

- `char array[N][M];`

指针数组的另外用途

- 传递命令行参数
  - echo.c
  - `./echo.c hello world`

```
int
main(int argc, char *argv[]){

}
```

  - `./echo.c hello -t world –e aaa`

字符串的动态分配内存

- Concat
  - concat.c

高级指针

- 这又是什么意思?
  - `int (*f[])();`
  - `int *(*f[])();`
  - `int (*f)(int, double);`
  - `int *(*g[])(int, double);`

函数指针

- 指针表示对某个特定内存地址的指向
  - 内存地址不仅能够表示变量
  - 还可能代表某个函数的实现
    - 函数占用内存单元，每一个函数都有自己的地址

    - point.c
    - strcpy-p-func.c

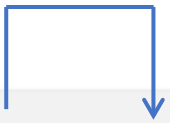
一个现实中可能遇到的地狱难度例子

- 人类不可读版本（STFW: clockwise/spiral rule）
  - 终极使用顺时针螺旋法则的案例

```
void (*signal (int sig, void (*func)(int)))(int);
```

一个现实中可能遇到的例子

- 人类不可读版本（STFW: clockwise/spiral rule）
  - 终极使用顺时针螺旋法则的案例

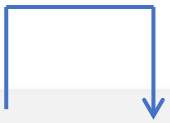


```
void (*signal (int sig, void (*func)(int)))(int);
```

  - signal是一个函数
    - 参数为……
    - 返回值为……

一个现实中可能遇到的例子

- 人类不可读版本（STFW: clockwise/spiral rule）
  - 终极使用顺时针螺旋法则的案例



```
void (*signal (int sig, void (*func)(int)))(int);
```

  - signal是一个函数
    - 参数为int和一个函数指针（参数为int，返回值为void）
    - 返回值为……

一个现实中可能遇到的例子

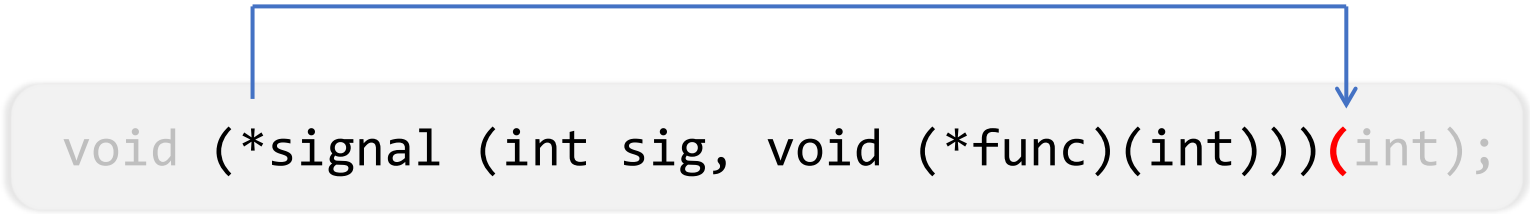
- 人类不可读版本（STFW: clockwise/spiral rule）
  - 终极使用顺时针螺旋法则的案例

```
void (*signal (int sig, void (*func)(int)))(int);
```

  - signal是一个函数
    - 参数为int和一个函数指针（参数为int，返回值为void）
    - 返回值为一个……指针

一个现实中可能遇到的例子

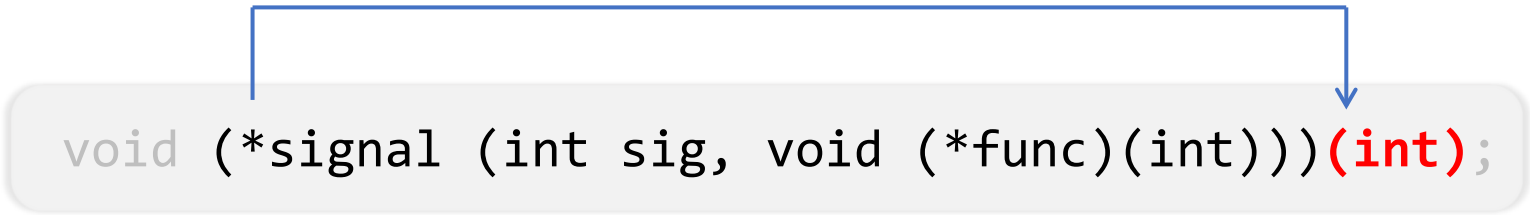
- 人类不可读版本（STFW: clockwise/spiral rule）
  - 终极使用顺时针螺旋法则的案例



  - signal是一个函数
    - 参数为int和一个指向函数的指针（函数参数为int，返回值为void）
    - 返回值为一个指向函数的指针（参数为…，返回值为…）

一个现实中可能遇到的例子

- 人类不可读版本（STFW: clockwise/spiral rule）
  - 终极使用顺时针螺旋法则的案例

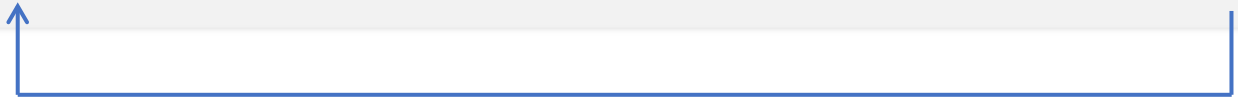


  - signal是一个函数
    - 参数为int和一个指向函数的指针（函数参数为int，返回值为void）
    - 返回值为一个指向函数的指针（参数为int，返回值为…）

一个现实中可能遇到的例子

- 人类不可读版本（STFW: clockwise/spiral rule）
  - 终极使用顺时针螺旋法则的案例

```
void (*signal (int sig, void (*func)(int)))(int);
```



  - signal是一个函数
    - 参数为int和一个指向函数的指针（函数参数为int，返回值为void）
    - 返回值为一个指向函数的指针（参数为int，返回值为void）

- 太复杂了>_<，有没有更简单的办法

一个现实中可能遇到的例子

- 人类不可读版本（STFW: clockwise/spiral rule）
  - 终极使用顺时针螺旋法则

```
void (*signal (int sig, void (*func)(int)))(int);
```

  - signal是一个函数
    - 参数为int和一个指向函数的指针（函数参数为int，返回值为void）
    - 返回值为一个指向函数的指针（参数为int，返回值为void）

```
void (*signal (int sig, void (*func)(int)))(int);
```

```
void (*signal (int sig, func))(int);
```

```
void (*)(int);
```

一个现实中可能遇到的例子

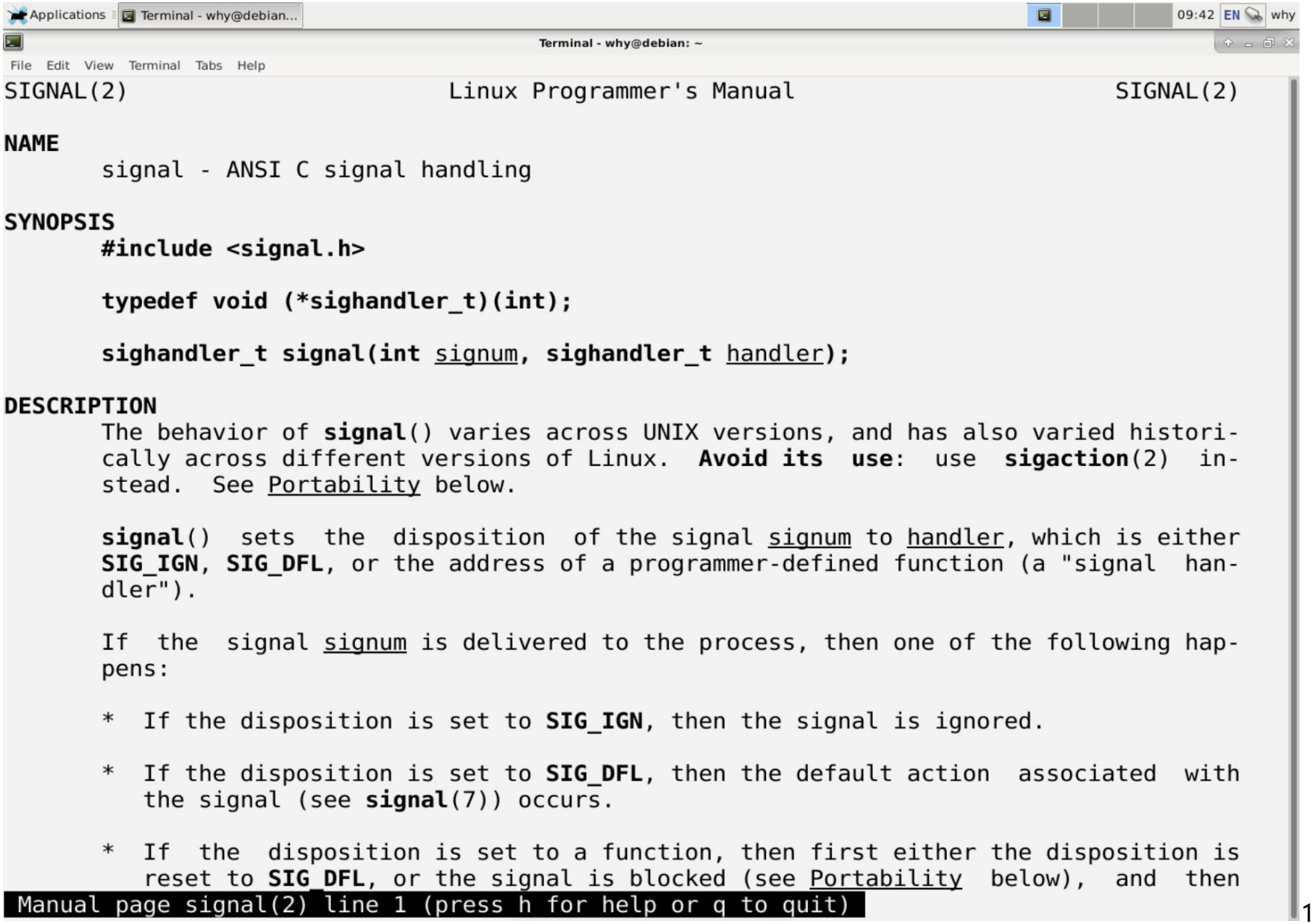
- 人类不可读版本（STFW: clockwise/spiral rule）
  - 终极使用顺时针螺旋法则

```
void (*signal (int sig, void (*func)(int)))(int);
```

- 人类可读版本

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int, sighandler_t);
```

man 2 signal


Applications
Terminal - why@debian...
09:42
EN
why
Terminal - why@debian: ~
File Edit View Terminal Tabs Help
SIGNAL(2) Linux Programmer's Manual SIGNAL(2)
NAME
signal - ANSI C signal handling
SYNOPSIS
#include <signal.h>
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
DESCRIPTION
The behavior of signal() varies across UNIX versions, and has also varied historically across different versions of Linux. Avoid its use: use sigaction(2) instead. See Portability below.
signal() sets the disposition of the signal signum to handler, which is either SIG_IGN, SIG_DFL, or the address of a programmer-defined function (a "signal handler").
If the signal signum is delivered to the process, then one of the following happens:
* If the disposition is set to SIG_IGN, then the signal is ignored.
* If the disposition is set to SIG_DFL, then the default action associated with the signal (see signal(7)) occurs.
* If the disposition is set to a function, then first either the disposition is reset to SIG_DFL, or the signal is blocked (see Portability below), and then
Manual page signal(2) line 1 (press h for help or q to quit)

struct, union, enum

struct

- 结构变量
  - 可能具有不同类型的值（成员）的集合

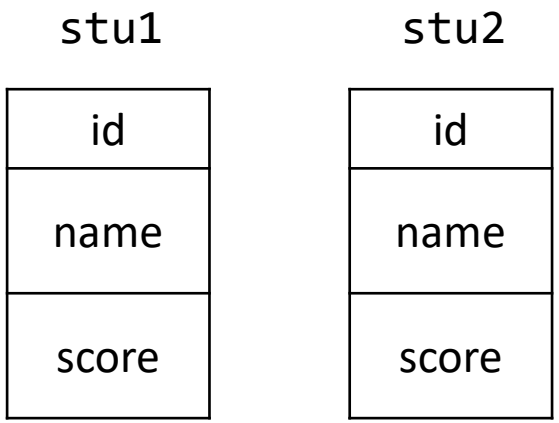
```
struct {
    int id;
    char name[N];
    double score;
}stu1, stu2;
```

  - struct.c
  - 初始化
    - {1, “Allen”, 98.5};
    - {.name = “Su”, .score = 88.0, .id = 2};
    - {.name = “Su”, 88.0, .id = 2};
    - {.name = “Su”, .score = 88.0};

struct成员

- 结构体成员作用域仅在当前结构体，具有独立的name space
  - 多结构体成员重名，互不冲突

- 成员访问通过“.”操作访问
  - `stu1.name`
  - `stu2.score`
  - `&stu1.name`



结构类型的名字

- 结构标记

```
struct record {
    int id;
    char name[N];
    double score;
};
struct record stu1, stu2;
```

- typedef类型定义

```
typedef struct {
            int num;
            long score;
            char id;
        } record2;
record2 stu2
```

结构类型作为参数和返回值

- 结构体变量赋值
  - `struct record stu1, stu2;`
  - `stu1 = stu2;`
  - 结构变量的名字不是一个地址
- rectangle.c
- `void Print(struct record stu);`
  - 值传递：结构体会复制

> “If a larger structure is to be passed to a function, it is generally more efficient to pass a pointer than to copy the whole structure.”
>
> —— K & R (p.131)

事情开始变得有趣了

- 结构类型给我们定义一种C语言能够理解的自定义的类型
- struct可以嵌套
  - 匿名结构：有成员无标记，未命名的结构类型

```
struct tag
{
    int i;
    struct m{
        int j, k;
    };
    struct
    {
        char c;
        float f;
    };
    struct
    {
        double d;
    }s;
} t;
```

```
t.i = 2000;
t.j = 5;
t.k = 6;
t.c = 'A';
t.f = 0.2;
t.s.d = 20.2;
```

struct内存对齐原则

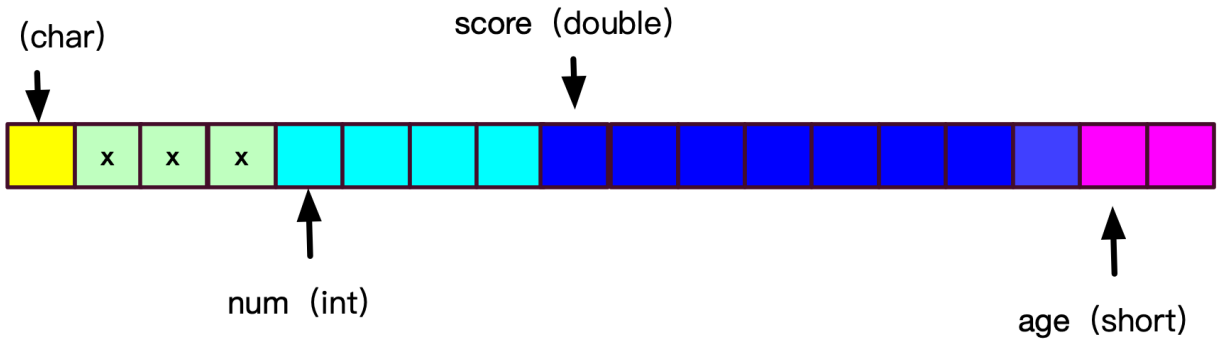
- 每个结构体的成员:
  - 第一个成员都位于偏移为0的位置
  - 以后每个数据成员的偏移量min(#pragma pack()指定的数，这个数据成员的自身长度) 的倍数**
    - 一般情况下不指定#pragma pack()默认长度32位系统是4字节，64位系统是8字节
- 在数据完成各自对齐之后，结构体本身也要对齐
  - 结构体的大小要和该结构体内最大元素大小的倍数，不够补齐
- 结构体内套用结构体
  - 遵循两个原则，首先第一点就是要将嵌套结构体内要内存对齐，然后就是嵌套结构体的起始位的偏移量必须是嵌套结构体内的占用最大内存属性的倍数就好，其他都一样

struct内存对齐原则

在没有指定#pragma pack()情况下

char只占用第一个字节所以
直接存到偏移为0的位置

同样的偏移量要是自身的倍数，但是
偏移量刚好是自身的倍数所以不用补齐



因为偏移量要是自身长度的倍数
所以要冲第五号位置开始存储
（int前面三个位置就是需要补齐的内存），
然后int自身长度有4字节

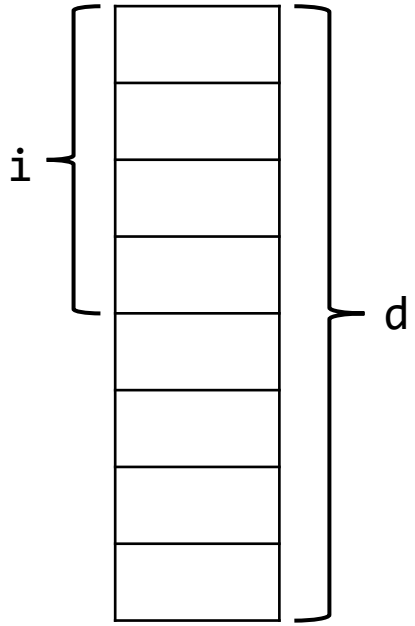
同样的偏移量要是自身的倍数，但是
偏移量刚好是自身的倍数所以不用补齐

最后还有一条规则是：结构的的长度要是结构体内最大元素尺寸的倍数，不难看出最大的是double 8字节，而我们成员对齐之后发现长度只有18字节，所以要补齐再补6个字节结果就是24字节刚好是8的倍数

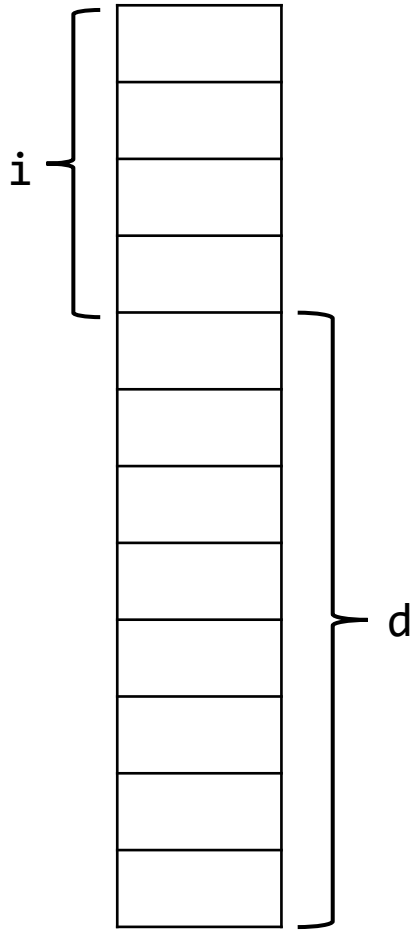
union

- 和struct的不同点：
  - 成员是否共用同样的内存空间
  - struct-union.c

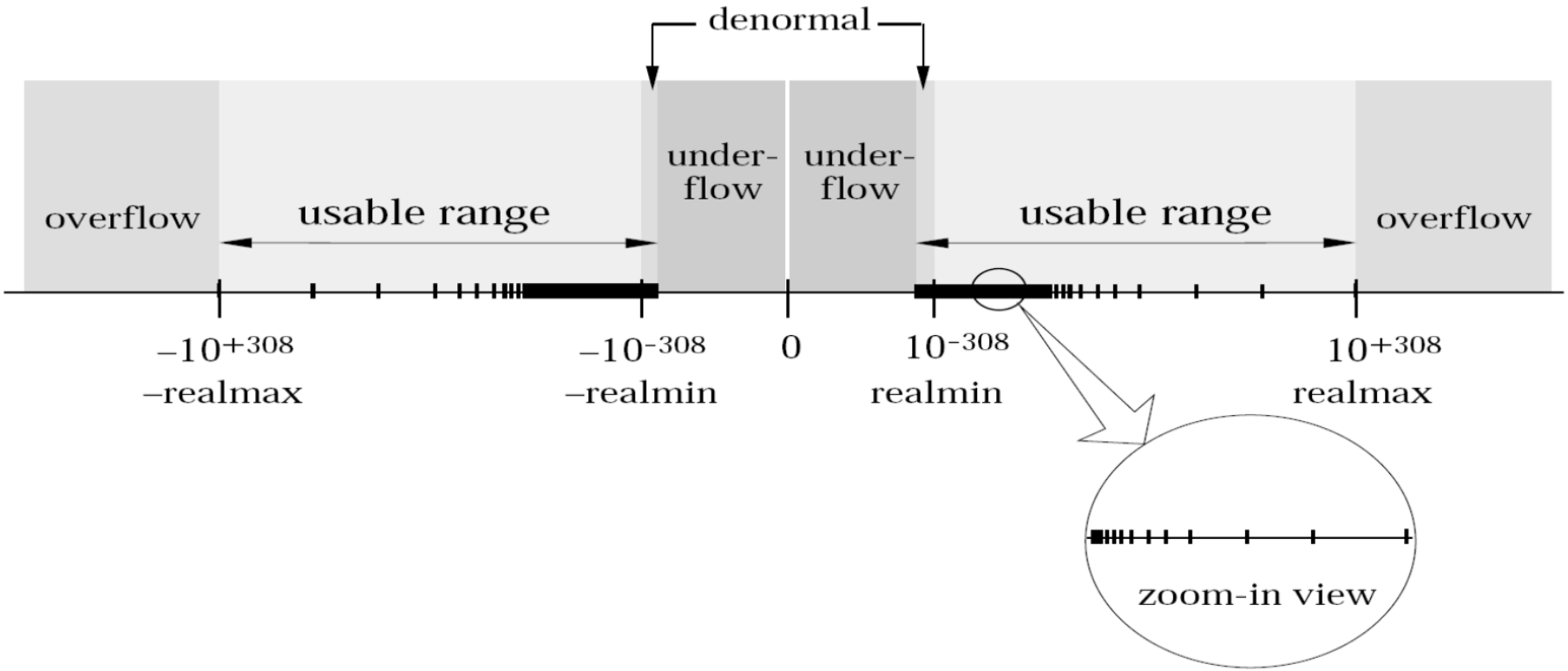
```
union{int i; double d};
```

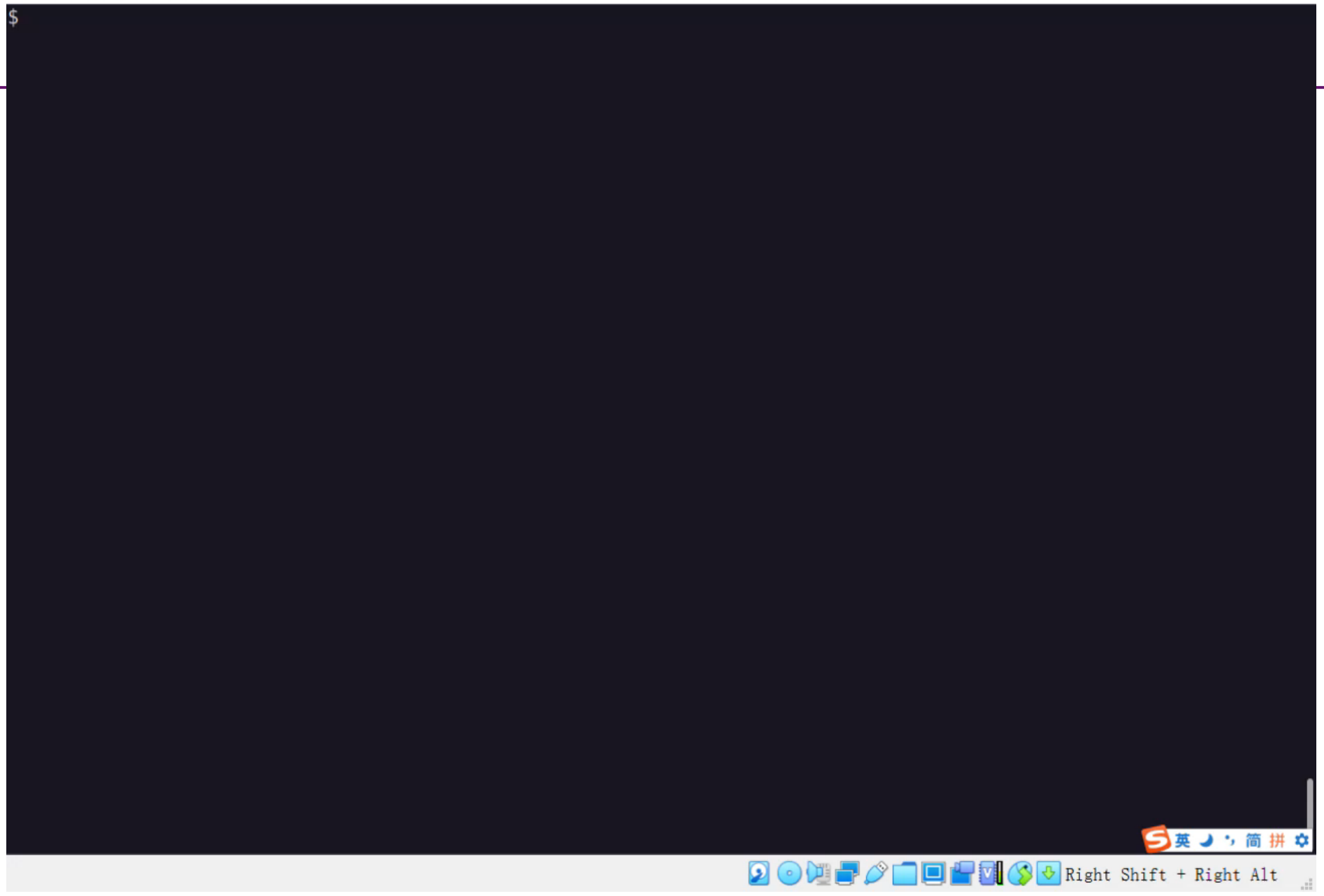



```
struct{int i; double d};
```



Floating Point Number Line




管理 控制 视图 热键 设备 帮助
$
Right Shift + Right Alt

enum

- 枚举类型
  - 由程序员列出的，具有常量的值
  - 常量符号化

  - `enum {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday};`

  - `typedef enum {False, True} Bool;`

  - `enum {RED, YELLOW, GREEN, NumOfColors};`

  - enum.c
  - month-enum.c

End