期末复习

王慧妍

why@nju.edu.cn

南京大学

软件学院

计算机软件研究所

回顾：这个学期学了些什么？有陷阱？

- 数据类型
  - 整型：`int`，`short`，`char`，`bool`
  - 浮点型：`float`，`double`，`long double`
  - 指针类型：`*`，`&`
  - 自定义类型：struct，union，enum
- 表达式语句
  - 操作符：优先级
- 流程控制
  - 条件分支：if，switch-case
  - 循环：for，while，do-while
  - 跳转：continue，break，goto
  - 函数调用：迭代和递归

数据类型

整型类型

类型	字节数	位数	取值范围	格式匹配符
char	1	8	$-2^7 \sim 2^7-1$	%c, %d
signed char	1	8	$-2^7 \sim 2^7-1$	%c, %d
unsigned char	1	8	$0 \sim 2^8-1$	%c, %d
signed short int	2	16	$-2^{15} \sim 2^{15}-1$	%hd
unsigned short int	2	16	$0 \sim 2^{16}-1$	%hu
signed int	4	32	$-2^{31} \sim 2^{31}-1$	%d
unsigned int	4	32	$0 \sim 2^{32}-1$	%u
signed long int	4	32	$-2^{31} \sim 2^{31}-1$	%ld
unsigned long int	4	32	$0 \sim 2^{32}-1$	%lu
signed long long int	8	64	$-2^{63} \sim 2^{63}-1$	%lld
unsigned long long int	8	64	$0 \sim 2^{64}-1$	%llu

short<=int<=long

类型所占机器位数与特定编译器平台相关

sizeof（静态运算符，编译时决定，不要在括号内做运算）

stdint.h

Fixed width integer types (since C99)

Types

Defined in header <stdint.h>

int8_t **int16_t** **int32_t** **int64_t**	signed integer type with width of exactly 8, 16, 32 and 64 bits respectively with no padding bits and using 2's complement for negative values (provided only if the implementation directly supports the type)
int_fast8_t **int_fast16_t** **int_fast32_t** **int_fast64_t**	fastest signed integer type with width of at least 8, 16, 32 and 64 bits respectively
int_least8_t **int_least16_t** **int_least32_t** **int_least64_t**	smallest signed integer type with width of at least 8, 16, 32 and 64 bits respectively
intmax_t	maximum width integer type
intptr_t	integer type capable of holding a pointer
uint8_t **uint16_t** **uint32_t** **uint64_t**	unsigned integer type with width of exactly 8, 16, 32 and 64 bits respectively (provided only if the implementation directly supports the type)
uint_fast8_t **uint_fast16_t** **uint_fast32_t** **uint_fast64_t**	fastest unsigned integer type with width of at least 8, 16, 32 and 64 bits respectively
uint_least8_t **uint_least16_t** **uint_least32_t** **uint_least64_t**	smallest unsigned integer type with width of at least 8, 16, 32 and 64 bits respectively
uintmax_t	maximum width unsigned integer type
uintptr_t	unsigned integer type capable of holding a pointer

The implementation may define typedef names `int`*N*`_t`, `int_fast`*N*`_t`, `int_least`*N*`_t`, `uint`*N*`_t`, `uint_fast`*N*`_t`, and `uint_least`*N*`_t` when *N* is not 8, 16, 32 or 64. Typedef names of the form `int`*N*`_t` may only be defined if the implementation supports an integer type of that width with no padding. Thus, `uint24_t` denotes an unsigned integer type with a width of exactly 24 bits.

Each of the macros listed in below is defined if and only if the implementation defines the corresponding typedef name. The macros `INT`*N*`_C` and `UINT`*N*`_C` correspond to the typedef names `int_least`*N*`_t` and `uint_least`*N*`_t`, respectively.

Macro constants

Defined in header <stdint.h>

Signed integers : width

INT8_WIDTH **INT16_WIDTH** (C23)(optional) **INT32_WIDTH** **INT64_WIDTH**	bit width of an object of type `int8_t`, `int16_t`, `int32_t`, `int64_t` (exactly 8, 16, 32, 64) (macro constant)
INT_FAST8_WIDTH **INT_FAST16_WIDTH** (C23) **INT_FAST32_WIDTH** **INT_FAST64_WIDTH**	bit width of an object of type `int_fast8_t`, `int_fast16_t`, `int_fast32_t`, `int_fast64_t` (macro constant)
INT_LEAST8_WIDTH **INT_LEAST16_WIDTH** (C23) **INT_LEAST32_WIDTH** **INT_LEAST64_WIDTH**	bit width of an object of type `int_least8_t`, `int_least16_t`, `int_least32_t`, `int_least64_t` (macro constant)
INTPTR_WIDTH (C23)(optional)	bit width of an object of type `intptr_t` (macro constant)
INTMAX_WIDTH (C23)	bit width of an object of type `intmax_t` (macro constant)

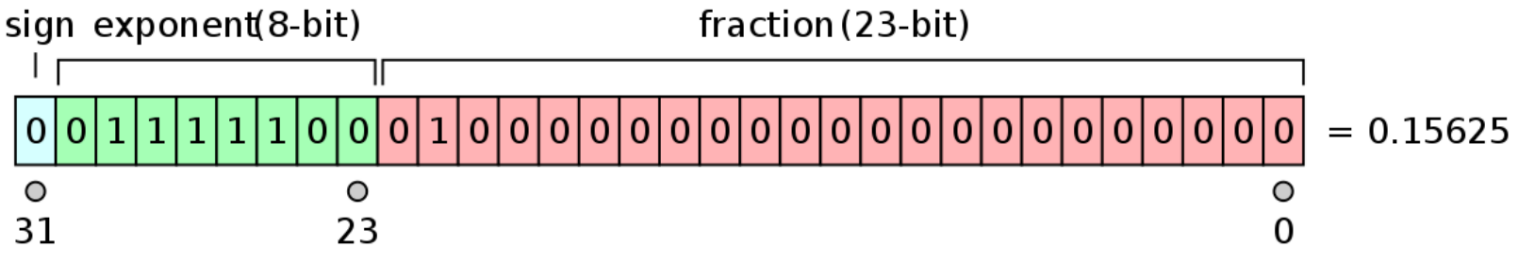
Undefined Behavior: 警惕整数溢出

表达式	值
`UINT_MAX+1`	0
`INT_MAX+1; LONG_MAX+1`	undefined
`char c = CHAR_MAX; c++;`	varies (???)
`1 << -1`	undefined
`1 << 0`	1
`1 << 31`	undefined
`1 << 32`	undefined
`1 / 0`	undefined
`INT_MAX % -1`	undefined

- W. Dietz, et al. Understanding integer overflow in C/C++. In *Proceedings of ICSE*, 2012.

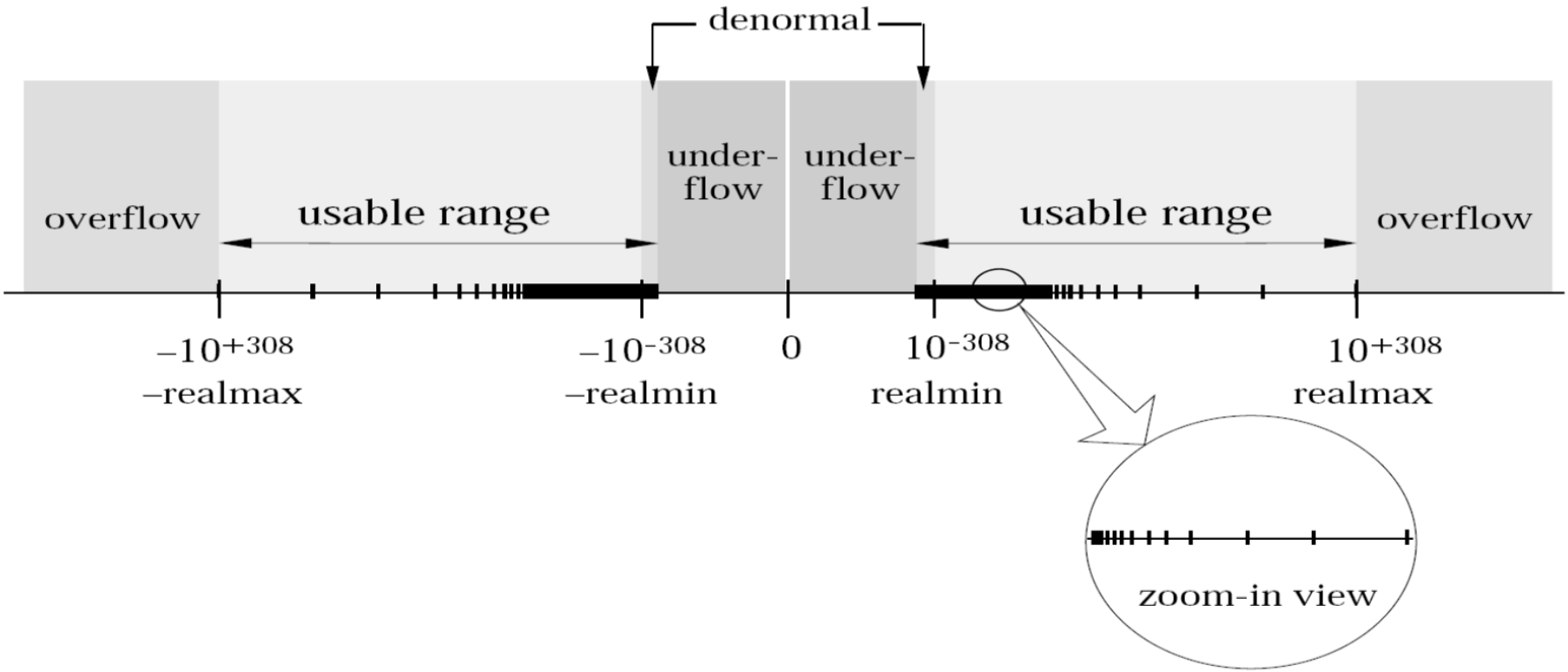
浮点类型

$$x = (-1)^S \times (1.F) \times 2^{E-B}$$



类型	字节数	位数	规范	取值范围	输入符	输出符
float	4	32	S1|E8|F23	$\pm 1.2E-38 \sim \pm 3.4E+38$	%f	%f, %e
double	8	64	S1|E11|F52	$\pm 2.2E-308 \sim \pm 1.8E+308$	%lf	%f, %e
long double	10/12/16	80/96/128				

Floating Point Number Line



浮点运算的精度

- `float`：大约6-7位有效数字
- `double`：大约15-16位有效数字

```
// Absolute tolerance comparison of x and y
if (Abs(x - y) <= EPSILON) …
```

```
// Relative tolerance comparison of x and y
if (Abs(x - y) <= EPSILON * Max(Abs(x), Abs(y)) …
```

```
if (Abs(x - y) <= Max(absTol, relTol * Max(Abs(x), Abs(y)))) …
```

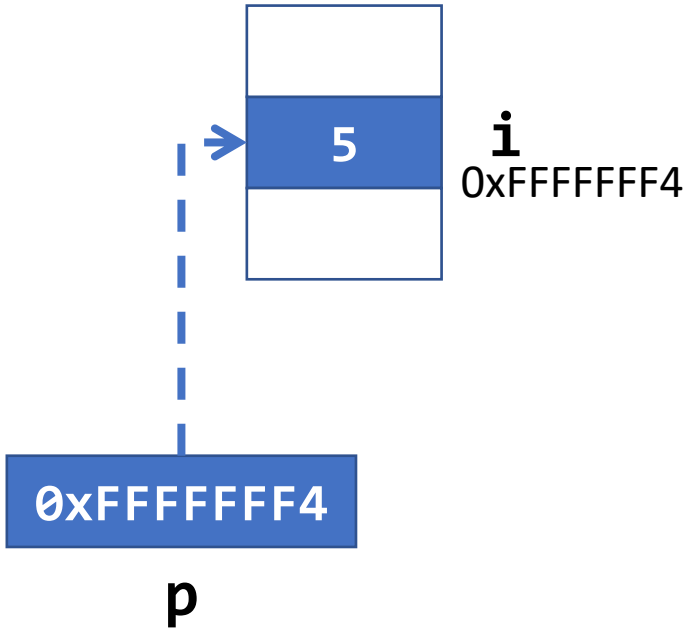
Floating-point tolerances revisited – realtimecollisiondetection.net – the blog

指针类型

- “A *pointer* is a *variable* that contains the *address* of a variable.”
  - 一个保存内存地址的变量，代表指向某个具体的内存地址
  - `int i = 5;`
  - `int* p;`
  - `p = &i;`

  “p是指向i的指针”

  → “p存储了i的内存地址”



- 指针使用
  - `*p = 2` 等价于 `i = 2`
  - `*p`：可以看作i的别名，代表使用*运算符访问存储在指向对象中的内容

  - 指针变量的值，是具有实际值的变量的地址，而普通变量的值是实际值

指针类型

- 可以作为参数
  - 数组作为函数参数
    - `void func(int (*mat)[10])`
    - `void func(int mat[][10])`

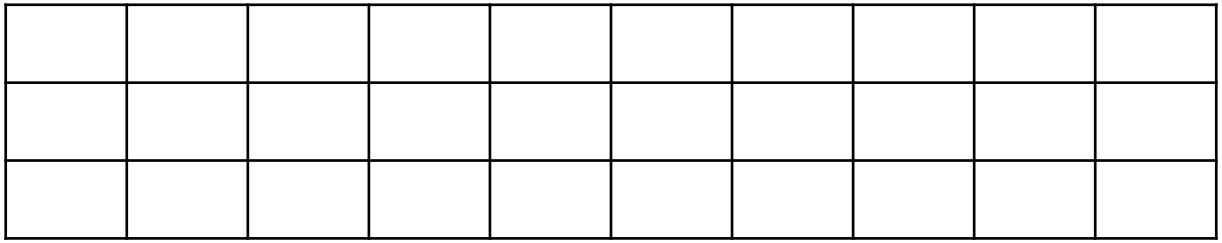
- 可以作为返回值

```
warning: function returns address of local variable [-Wreturn-local-addr]
```

- 可以做算术操作，也可以类似数组做下标运算[ ]
  - `int a[10], *p;`
  - `p+1`：加其指向类型的`sizeof`大小
    - 如果指针指向的不是连续内存，没有意义
    - 一般和数组关系密切

多维数组与指针

- int matrix[3][10];
  - matrix
  - matrix+1
  - *(matrix +1)
  - *(matrix+1)+5
  - *(*(matrix+1)+5)
  - p = matrix , p = matrix[1] p = matrix[0][0]
  - p = &matrix , p = &matrix[1] p = &matrix[0][0]



string.h

- 常见的字符串函数
  - 不受限制的字符串函数
    - `size_t strlen (char const *string);`
    - `char *strcpy (char *dst, char const *src);`
    - `char *strcat (char *dst, char const *src);`
    - `int strcmp (char const *s1, char const *s2);`
  - 长度受限的字符串函数
    - `char *strncpy (char *dst, char const *src, size_t len);`
    - `char *strncat (char *dst, char const *src, size_t len);`
    - `int strncmp (char const *s1, char const *s2, size_t len);`

string.h

- 常见的字符串函数
  - 查找字符或子串函数
    - `char *strchr(char const *str, int ch);`
    - `char *strrchr(char const *str, int ch);`
    - `char *strpbrk(char const *str, char const *group);`
    - `char *strstr(char const *s1, char const *s2);`
  - 查找计数
    - `size_t *strspn(char const *str, char const *group);`
    - `size_t *strcspn(char const *str, char const *group);`
  - 查找标记
    - `char *strtok(char *str, char const *sep);`
      - strtok.c

Functions

Character classification

Defined in header <ctype.h>

isalnum	checks if a character is alphanumeric (function)
isalpha	checks if a character is alphabetic (function)
islower	checks if a character is lowercase (function)
isupper	checks if a character is an uppercase character (function)
isdigit	checks if a character is a digit (function)
isxdigit	checks if a character is a hexadecimal character (function)
iscntrl	checks if a character is a control character (function)
isgraph	checks if a character is a graphical character (function)
isspace	checks if a character is a space character (function)
isblank (C99)	checks if a character is a blank character (function)
isprint	checks if a character is a printing character (function)
ispunct	checks if a character is a punctuation character (function)

Character manipulation

tolower	converts a character to lowercase (function)
toupper	converts a character to uppercase (function)

Conversions to and from numeric formats

Defined in header <stdlib.h>

atof	converts a byte string to a floating-point value (function)
atoi atol atoll (C99)	converts a byte string to an integer value (function)
strtol strtoll (C99)	converts a byte string to an integer value (function)
strtoul strtoull (C99)	converts a byte string to an unsigned integer value (function)
strtof (C99) strtod strtold (C99)	converts a byte string to a floating point value (function)
strfromf (C23) strfromd (C23) strfromld (C23)	converts a floating point value to a byte string (function)

Defined in header <inttypes.h>

strtoimax (C99) strtoumax (C99)	converts a byte string to intmax_t or uintmax_t (function)

String manipulation

Defined in header <string.h>

strcpy strcpy_s (C11)	copies one string to another (function)
strncpy strncpy_s (C11)	copies a certain amount of characters from one string to another (function)
strcat strcat_s (C11)	concatenates two strings (function)
strncat strncat_s (C11)	concatenates a certain amount of characters of two strings (function)
strxfrm	transform a string so that strcmp would produce the same result as strcoll (function)
strdup (C23)	allocates a copy of a string (function)
strndup (C23)	allocates a copy of a string of specified size (function)

String examination

Defined in header <string.h>

strlen strnlen_s (C11)	returns the length of a given string (function)
strcmp	compares two strings (function)
strncmp	compares a certain amount of characters of two strings (function)
strcoll	compares two strings in accordance to the current locale (function)
strchr	finds the first occurrence of a character (function)
strrchr	finds the last occurrence of a character (function)
strspn	returns the length of the maximum initial segment that consists of only the characters found in another byte string (function)
strcspn	returns the length of the maximum initial segment that consists of only the characters not found in another byte string (function)
strpbrk	finds the first location of any character in one string, in another string (function)
strstr	finds the first occurrence of a substring of characters (function)
strtok strtok_s (C11)	finds the next token in a byte string (function)

qsort和bsearch

- https://en.cppreference.com/w/c/algorithm
  - Pointer type matters！

```
int cmp(const void *a, const void *b);
```

qsort, qsort_s

Defined in header <stdlib.h>

```
void qsort( void* ptr, size_t count, size_t size,
            int (*comp)(const void*, const void*) );                    (1)

errno_t qsort_s( void* ptr, rsize_t count, rsize_t size,
                 int (*comp)(const void*, const void*, void*),          (2)  (since C11)
                 void* context );
```

bsearch, bsearch_s

Defined in header <stdlib.h>

```
void* bsearch( const void *key, const void *ptr, size_t count, size_t size,
               int (*comp)(const void*, const void*) );                 (1)

void* bsearch_s( const void *key, const void *ptr, rsize_t count, rsize_t size,
                 int (*comp)(const void *, const void *, void *),       (2)  (since C11)
                 void *context );
```

qsort和bsearch

- 关于 qsort 与 bsearch 的用法，可以参考：
  - 课堂录屏
    - https://www.bilibili.com/video/BV1jrUYB5EC9?spm_id_from=333.788.videopod.sections&vd_source=49dd5159129c5cf96b663a53b83768bd
    - https://www.bilibili.com/video/BV1mu2xBNEpq/?spm_id_from=333.1387.collection.video_card.click&vd_source=49dd5159129c5cf96b663a53b83768bd
  - 飞书文档
    - https://njusecourse.feishu.cn/wiki/ZlxPwL2W2i1Xuek2CXgcxau3nLd?from=from_copylink
  - 务必熟练掌握！

指针和const

- 指针是const
  - `int * const p = &a;`
  - `*p = 100;//ok`
  - `p = &b; //ERROR`
  - `p++; //ERROR`

- 指针所指是const
  - `const int *p = &a;`
  - `*p = 100; //ERROR`
  - `a = 100; //ok`
  - `p = &b; //ok`

表示不能通过该指针修改此变量
（并不能使变量变成const）

  - 数组名称天然是const，不可改变其值，常量地址

指针指向动态分配内存

- 回顾VLA：可变长数组int array[n]
  - 不推荐

- C函数库提供malloc和free，用于执行动态内存的分配与释放
  - void *malloc(size_t _Size);
    - typedef unsigned long/int size_t
  - void free(void *pointer);
  - int *p = NULL;
  - p = (int *)malloc(n*sizeof(int));
  - free(p);
  - 警惕：分配失败返回NULL，p不可随意移动

自定义类型：struct

- 结构体
  - 可能具有不同类型的值（成员）的集合
  - 初始化
    - {1, “Allen”, 98.5};
    - {.name = “Su”, .score = 88.0};
    - {.name = “Su”, 88.0, .id = 2};

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

- 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

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

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

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)

- `void Print(struct record * stu);//stu->x, (*stu).x`

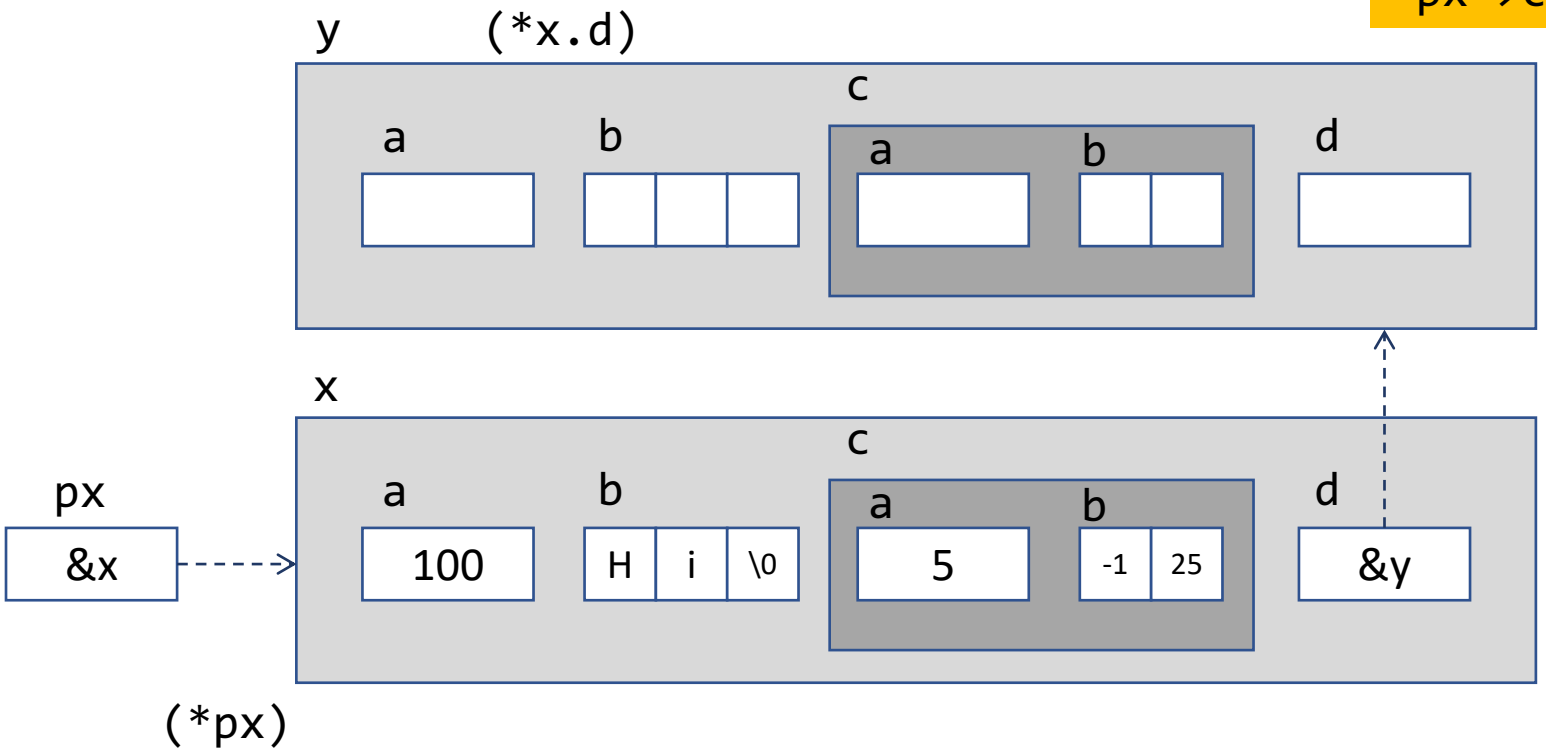
```c
typedef struct {
    int a;
    short b[2];
} EX2;
```

```c
typedef struct EX{
    int a;
    char b[3];
    EX2 c;
    struct EX *d;
} EX;
```

```c
EX x;
EX *px = &x;
```

```c
EX y;
x.d = &y;
```

*px->c.b




```c
EX x = {100, {'H', 'i', '\0'}, {5, {-1, 25}}, 0};
```

struct自引用

```
struct SELF{
    int a;
    struct SELF b;
    float c;
};
```

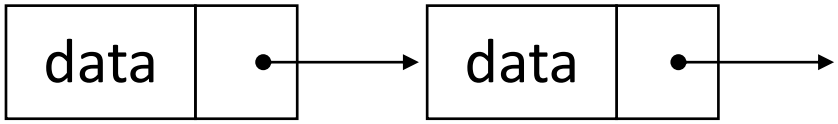
```
struct SELF{
    int a;
    struct SELF *b;
    float c;
};
```

```
typedef struct {
    int a;
    SELF *b;
    float c;
} SELF;
```

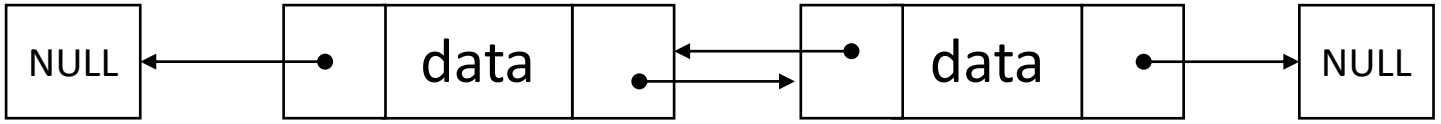
```
typedef struct SELF_TAG {
    int a;
    struct SELF_TAG *b;
    float c;
} SELF;
```

Linked List

- 链表：内存非连续的线性结构
  - 单链表节点
    - 存储有价值的数据，与指向下一个链表节点的指针
    - `struct node {int data; struct node *next;};`

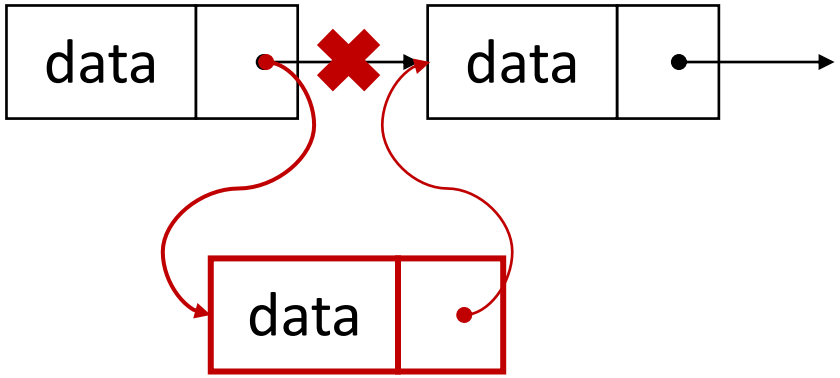


  - 双向链表
    -
```
struct node {
        struct node * prev;
        int data;
        struct node *next;
};
```

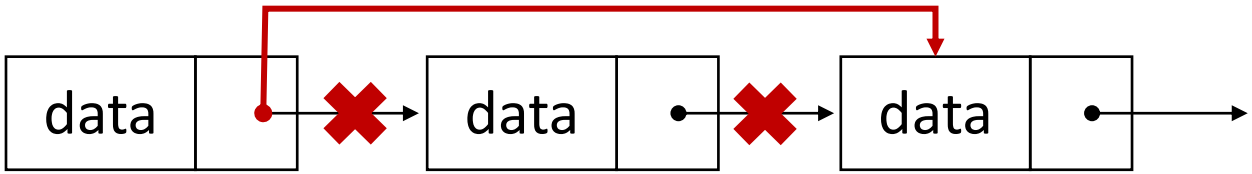


链表的节点插入和删除

- 新链表节点的插入



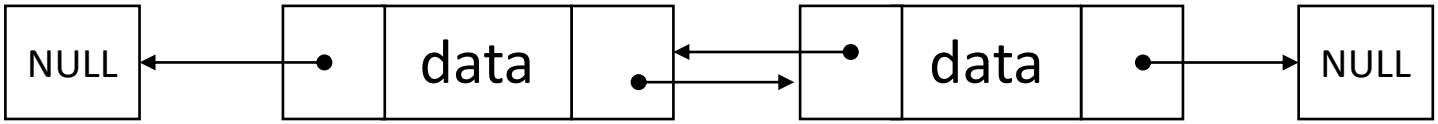
- 已有链表节点的删除



单向链表和双向链表

- 单向链表
  - `struct node {int data; struct node *next;};`

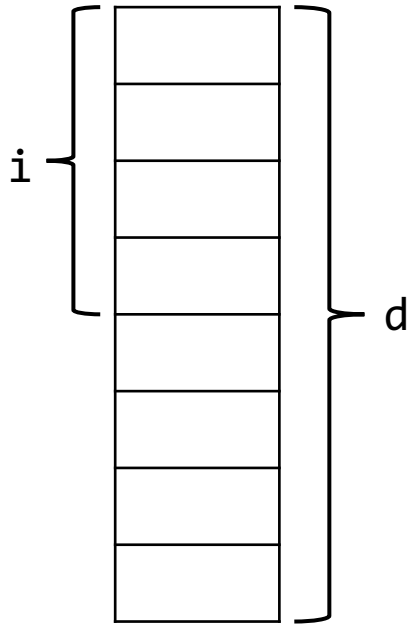
- 双向链表
  -
```
struct node {
  struct node * prev;
  int data;
  struct node *next;
};
```



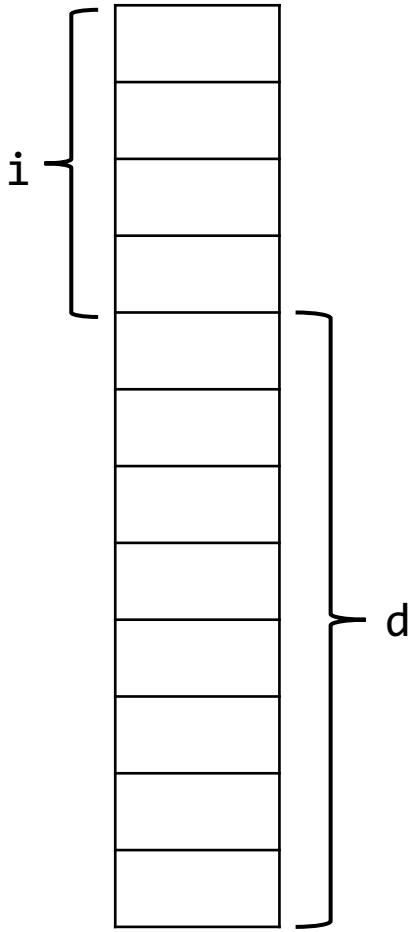
union

- 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};
```



表达式语句

优先级

- C Operator Precedence - cppreference.com
  - `int *p, q;`
  - `*p++;`
  - `a & b != 0`
  - `a << 4 + x`

表 2-1　C 语言运算符优先级表（由上至下，优先级依次递减）

运 算 符	结 合 性
() [] -> .	自左向右
! ~ ++ -- - (type) * & sizeof	自右向左
* / %	自左向右
+ -	自左向右
<< >>	自左向右
< <= > >=	自左向右
== !=	自左向右
&	自左向右
^	自左向右
|	自左向右
&&	自左向右
||	自左向右
?:	自右向左
assignments	自右向左
,	自左向右

优先级

- C Operator Precedence - cppreference.com
  - `&stu1.name`
  - `*stu1.name`
  - `int *p[5];`
  - `int *(p[5]);`
  - `int (*p)[5];`
  - `int (*f)(int);`
  - `int *f(int);`
  - `*f();`
  - `(*f)();`

表 2-1　C 语言运算符优先级表（由上至下，优先级依次递减）

运 算 符	结 合 性
() [] -> .	自左向右
! ~ ++ -- - (type) * & sizeof	自右向左
* / %	自左向右
+ -	自左向右
<< >>	自左向右
< <= > >=	自左向右
== !=	自左向右
&	自左向右
^	自左向右
|	自左向右
&&	自左向右
||	自左向右
?:	自右向左
assignments	自右向左
,	自左向右

流程控制

if

- 警惕悬挂的else
  - else总是匹配前面最近的if
    - 无大括号隔开

```
if(…) {
    …
}
else {
    …
}
```

- 运算符
  - 关系运算符：<，>，<=，>=（优先级低于算术运算符）
    - i > j > k
    - i + j < j * k
  - 判等运算符：==，!=（优先级低于关系运算符）
    - i < j == j < k
  - 逻辑运算符：&&(与)，||(或)，!(非)
    - (i != 0) && (j / i > 0)

注意逻辑运算符与位运算的区别

- & (与), | (或), ~ (非)
- ^ (异或)
- << (左移位), >> (右移位)

switch-case

```
switch (expression)
{
case /* constant-expression */:
    /* code */
    break;
case /* constant-expression */:
    /* code */
    break;

default:
    break;
}
```

For循环

- Given a set *A* of integers, to compute their minimum.


```
for ( <expression> ; <expression> ; <expression> )
    <statement>
```

循环开始前的准备　　循环结束条件　　每轮循环的最后一个执行

惯用法i++

while循环

```
while (表达式){
    语句
}
```

- “当”
  - 当表达式条件满足时，执行循环体内语句

do-while循环

```
do{
    语句
}while (表达式);
```

- 进入循环时不做检查，执行完一轮循环后，检查条件是否满足，满足则进入下一轮，否则结束循环
- “一直做，直到表达式不满足了”

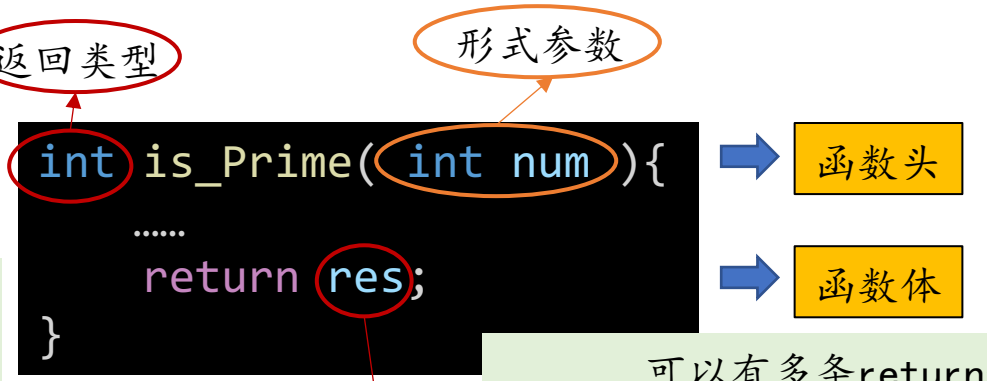
- while(1){……}

- break
  - 跳出最近的循环
- continue
  - 跳出当前这一次循环

函数定义与使用

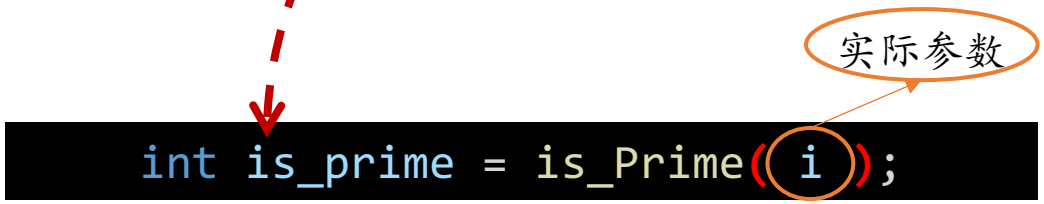
每个形式参数必须有类型
如：int a, int b
可以没有形式参数（有不建议过多）

返回类型不能是数组



不可省略返回值（C99）
除非函数返回类型为void
void f(…){…}

可以有多条return语句，
但每次只可能执行一条，
标志函数执行结束



有返回值可按需赋值
is_Prime(i);

带参函数需要传入实际参数
f();//不可省略“()”

函数声明

- 在定义前需要使用函数时，提前写上函数声明
  - 在调用一个函数之前，必须对其进行声明或定义

```
int is_Prime();
```

```
int is_Prime( int );
```

```
int is_Prime( int num );
```

```
int is_Prime( int num ){
    ……
    return res;
}
```

函数运行细节

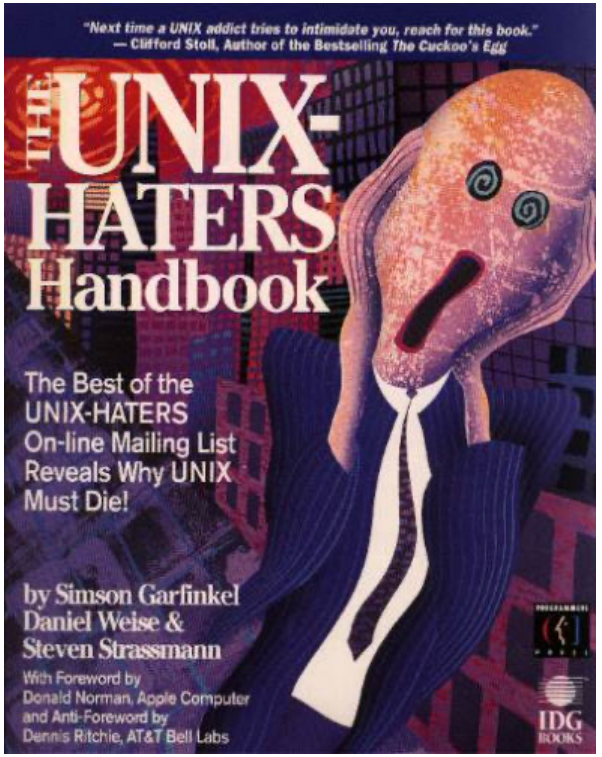
- 利用堆栈实现函数的调用与返回
  - Programs run in memory (内存; 記憶體).
  - Memory = Stack (栈区) + Heap (堆区) + …
  - Each function call has its own stack frame (栈帧).
  - Stack grows/shrinks with function calls and returns.

- Visualization of Function Calls @ C Tutor

还有一些啰啰嗦嗦的故事

The UNIX Hater's Handbook (and Beyond)

- 写于1994年
  - Simson Garfinkel 的主页有电子版
  - 说有道理也有道理说没道理也没道理



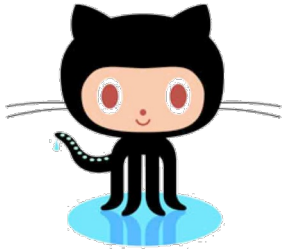
- 至少指出了 UNIX 的一些缺陷
  - user friendly
  - 命令行/系统工具的缺陷
  - 手册的冗长

- 但今天 UNIX/Linux 已经成熟多了！
  - `man`
  - `tldr`

The Community Way

开源社区 (百度百科)：根据相应的开源软件许可证协议公布软件源代码的网络平台，同时也为网络成员提供一个自由学习交流的空间。

- 从GitHub获取代码
  - 传统工具链 + “现代”编程体验
    - ~~有的时候网络不太靠谱~~



github
SOCIAL CODING

```
git clone -b 2024 git@github.com:NJU-ProjectN/ics-pa.git ics2024
git clone https://github.com/NJU-ProjectN/ics-workbench
```

GitLab

gitee

https://git.nju.edu.cn/

The Community Way (cont'd)

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 50 million developers.

- 无所不能的代码聚集地
  - 有整个计算机系统世界的代码
    - 硬件、操作系统、分布式系统、库函数、应用程序……
- 学习各种技术的最佳平台
  - 海量的文档、学习资料、博客（新世界的大门）
    - 提供友好的搜索（例子：**awesome C**）

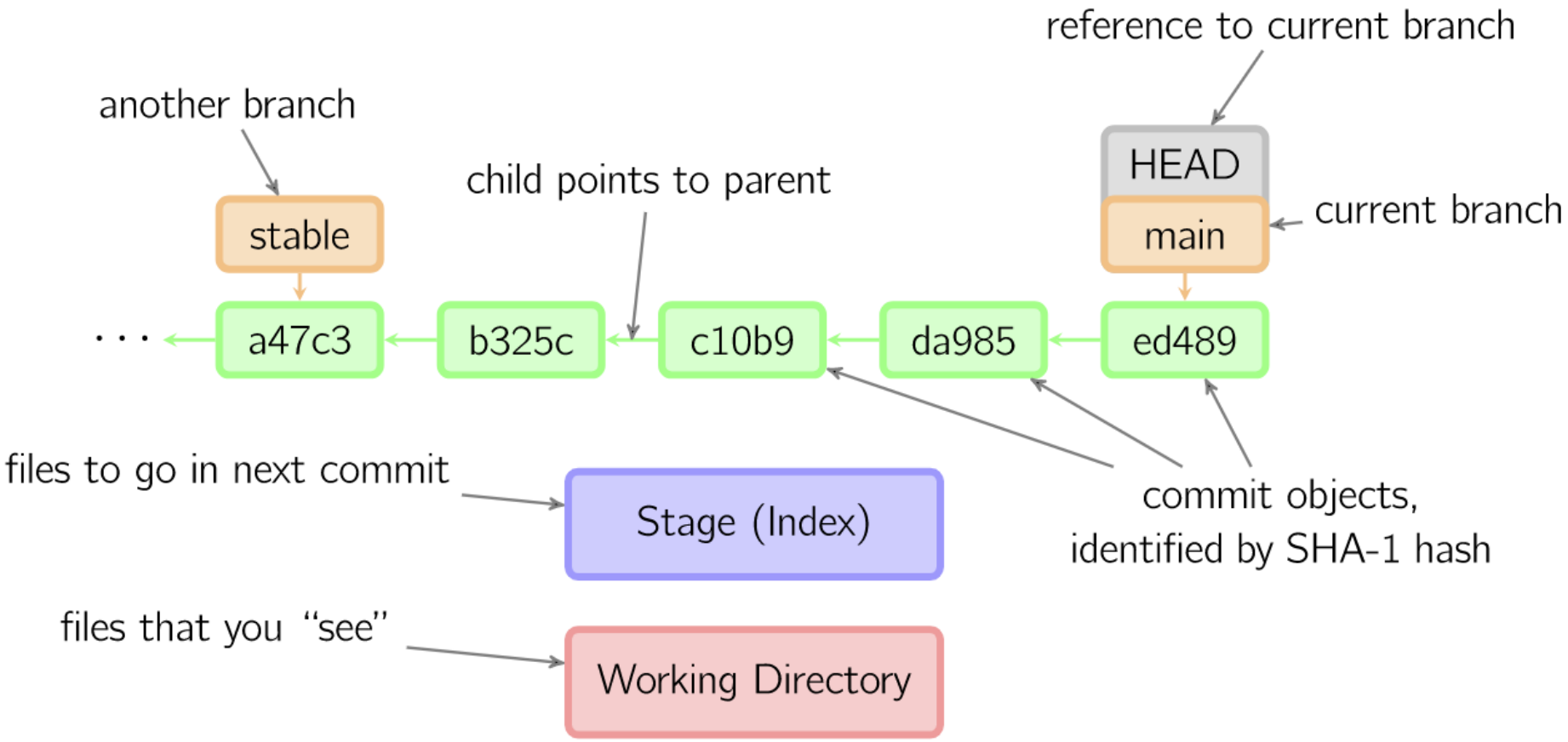
学习Git？

```
git clone -b 2024 https://github.com/NJU-ProjectN/ics-pa ics2024
```

内心独白：又要学新东西，我拒绝= =

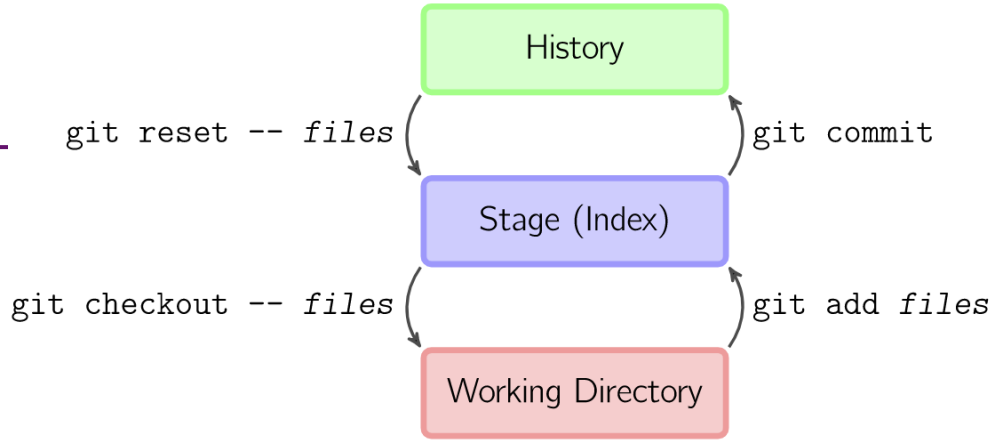
- RTFM?STFW！
  - 百度：得到一堆不太靠谱的教程
  - 大家已经见识过开源社区的力量了
    - A Visual Git Reference
      - 英文、中文、日文……
    - 好的文档是存在的
      - 还记得 tldr 吗？

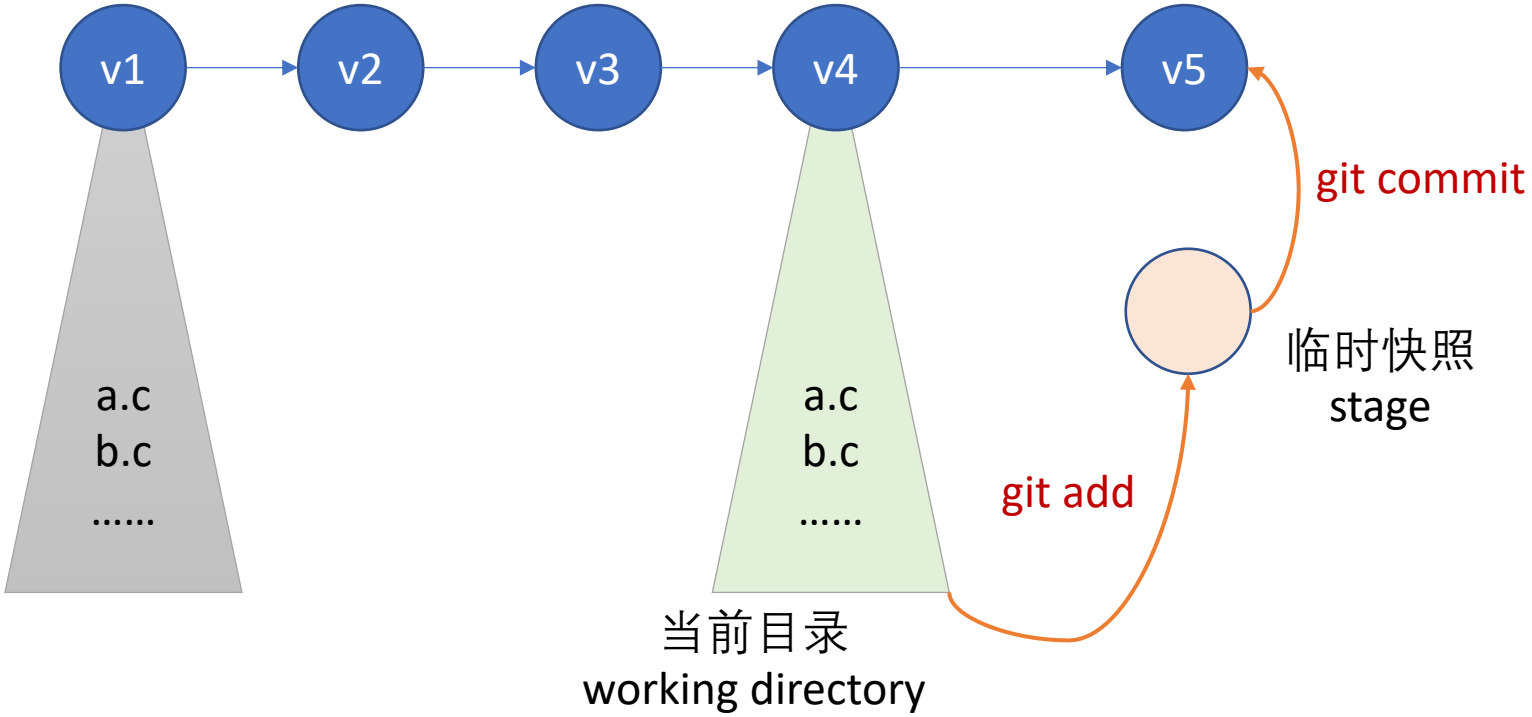
A Visual Git Reference



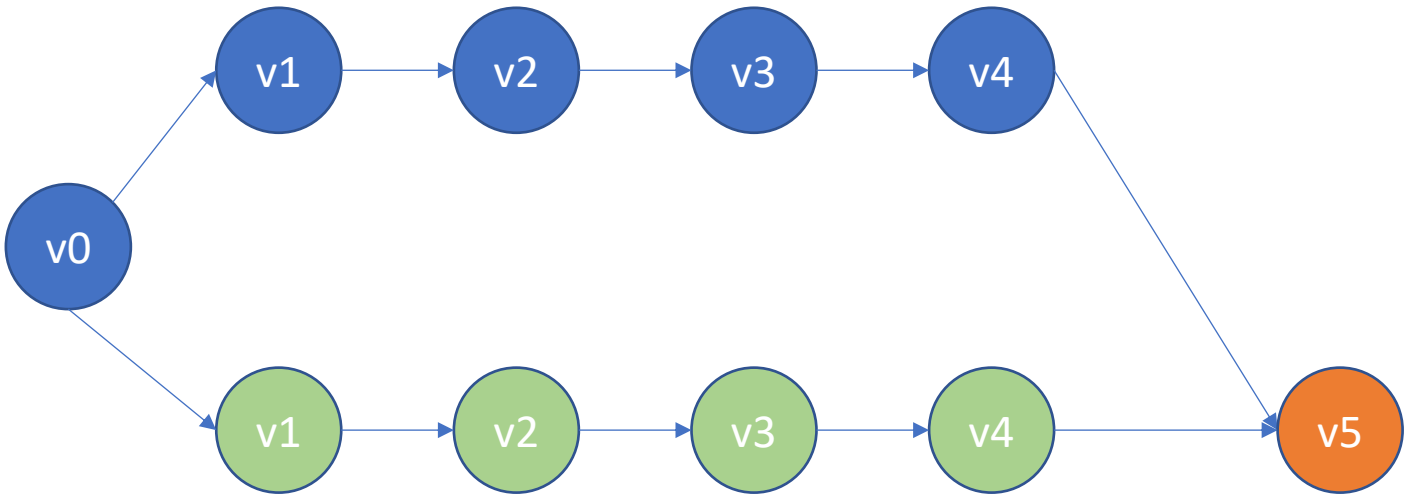

```
commit 8738b7b32e71a78f5b73376dc664780dd16800eb (HEAD -> pa1)
Author: tracer-ics2020 <tracer@njuics.org>
Date:   Thu Aug 19 18:27:15 2021 +0800
```

Git


History
git reset -- files
git commit
Stage (Index)
git checkout -- files
git add files
Working Directory


v1
v2
v3
v4
v5
a.c
b.c
.....
a.c
b.c
.....
git commit
临时快照
stage
git add
当前目录
working directory

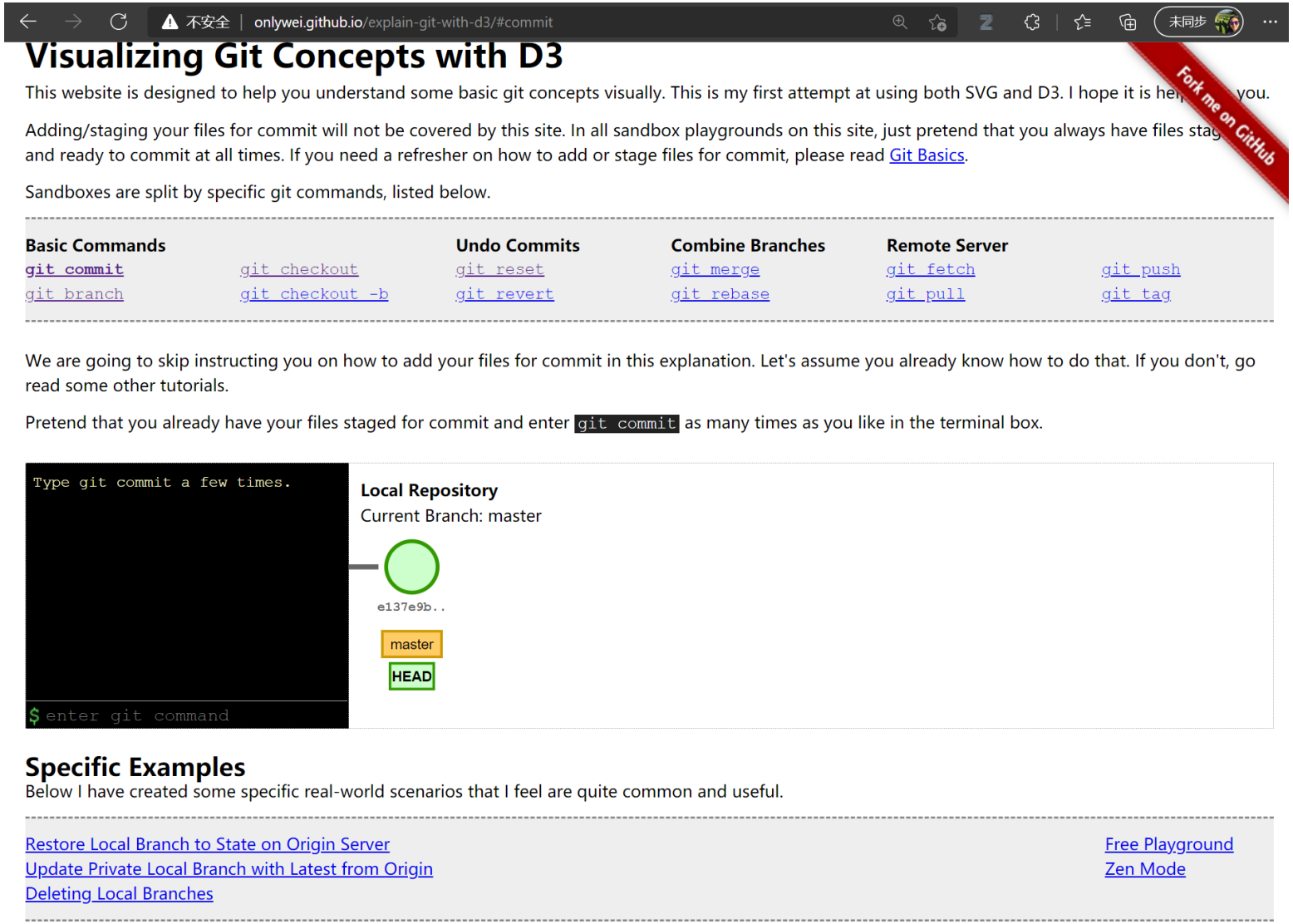
Git：分布式版本控制系统



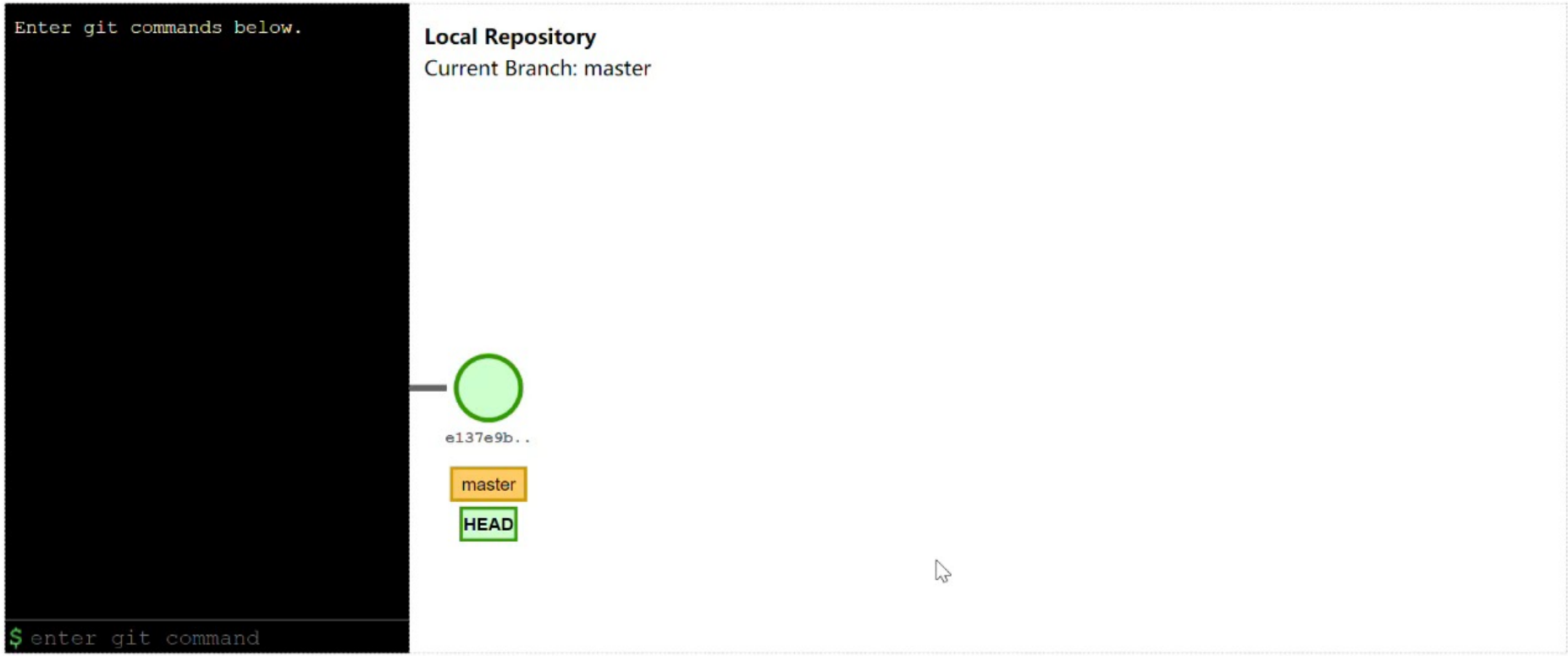
一些Comments

- 有趣的 “--”
  - UNIX 的设计缺陷 (UGH 中点名批评)
    - 虽然是编程语言，但 Shell 更贴近自然语言
  - 也有很多 corner cases
    - 如果有一个文件叫 “-rf”……怎么删除它？？？
    - best practice: 文件名不以 “-” 开头、不含空格/符号……
- 体验 Git
  - 创建一个新的 repo，自由探索
    - 为什么 “预计完成时间 XX 小时” 是骗人的？
      - 预计完成时间是假设你在大一开始就用 Git 的
  - Visualizing Git Concepts with D3

Visualizing Git Concepts with D3



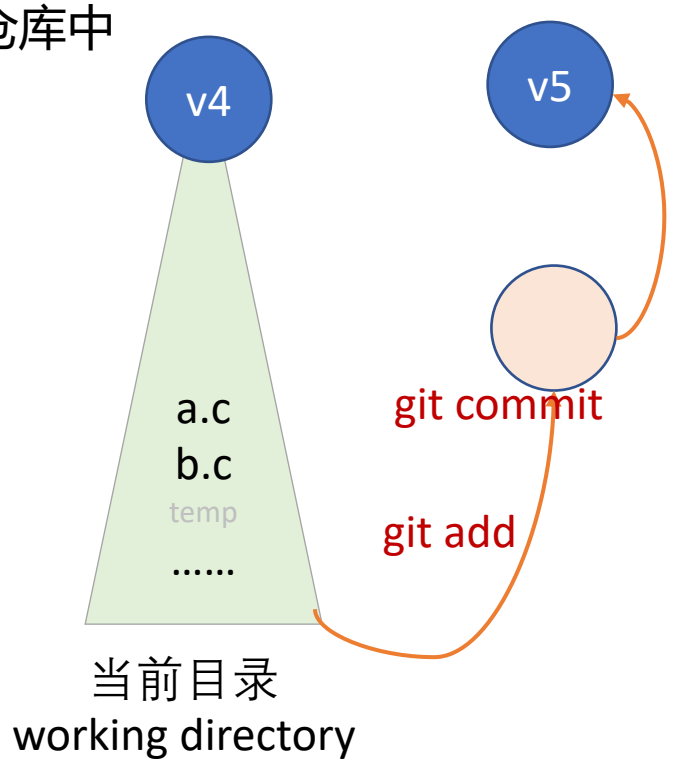
Visualizing Git Concepts with D3



一些Comments (cont'd)

- 我们使用了“白名单” .gitignore文件
  - 只在Git repo里管理.c，.h和Makefile
    - 基本原则：一切生成的文件都不放在Git仓库中

```
*           # 忽略一切文件
!*/         # 除了目录
!*.c        # .c
!*.h        # ...
!Makefile*
!.gitignore
```



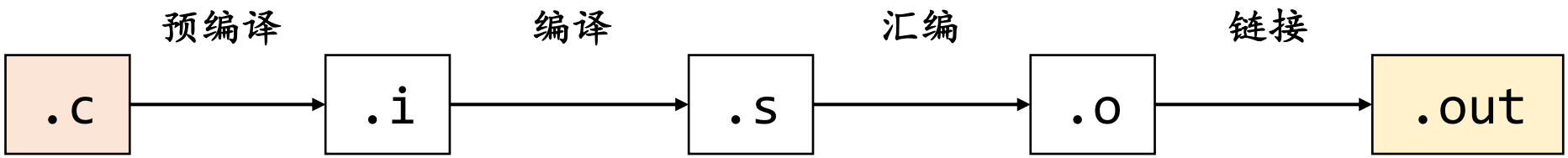
- 为什么ls看不到这个文件？
  - 怎么还有一个.git

回顾一下

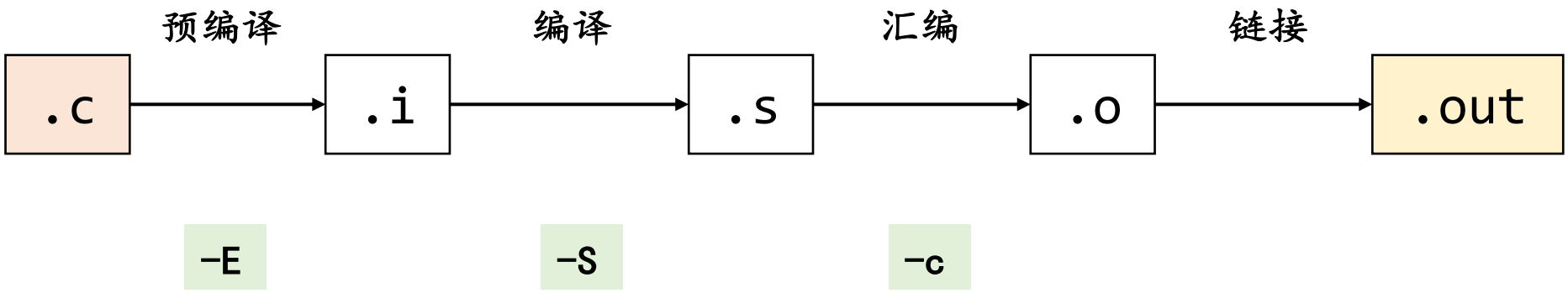
- 在IDE里，为什么按一个键，就能够编译运行？
  - 编译、链接
    - `.c` → 预编译 → `.i` → 编译 → `.s` → 汇编 → `.o` → 链接 → `a.out`
  - 加载执行
    - `./a.out`

- 背后是通过调用命令行工具完成的
  - RTFM： `man gcc; gcc –help; tldr gcc`
    - 控制行为的三个选项：`-E`，`-S`，`-c`

IDE的一个键到底发生了什么？



从源代码到可执行文件



编写大型程序

- 把程序划分成多个文件
  - 头文件
    - 一般包括宏定义，变量声明，函数原型
    - 惯例扩展名为.h
    - 全局变量：static, extern的区别
  - 源文件
    - 每个源文件包含程序的部分内容，主要是函数定义和变量定义
    - 某个源文件必须包含名为main的函数，作为程序的起始点

```c
int max(int x, int y);
```

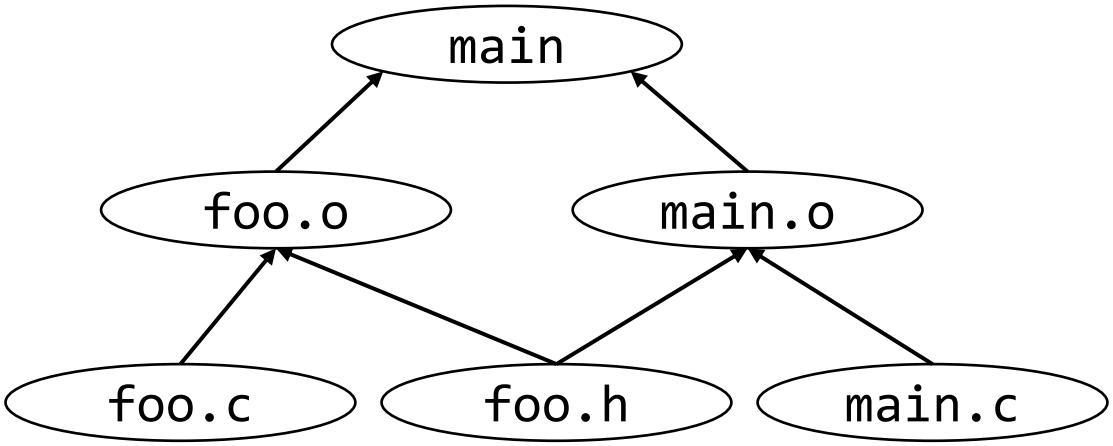
```c
#include"max.h"

int max(int x, int y){
    return x>y?x:y;
}
```

```c
#include<stdio.h>
#include"max.h"

int main(){
    int a = 5;
    int b = 6;
    printf("%d\n", max(a,b));
}

```

构建多文件程序

lec > make > M makefile

```
1  main: foo.o main.o
2      gcc -o main foo.o main.o
3  foo.o: foo.c foo.h
4      gcc -c -o foo.o foo.c
5  main.o: main.c foo.h
6      gcc -c -o main.o main.c
7  clean:
8      rm -f *.o
9  run: main
10     ./main
```



关于debug的一点福利

开始调试之前

- 摆正心态 (编程哲♂学)

机器永远是对的

不管是 crash 了，图形显示不正常了，还是 HIT BAD TRAP 了，最后都是你自己背锅

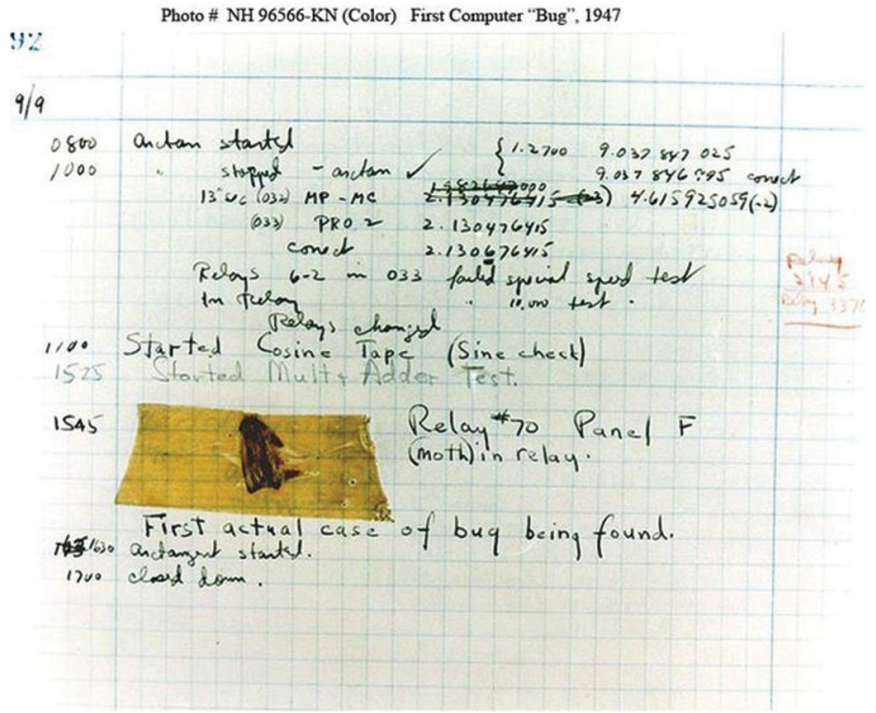
未测代码永远是错的

你以为最不可能出 bug 的地方，往往 bug 就在那躺着

调试理论

- 程序的两个功能
  - 人类世界需求的载体
    - 理解错需求 → bug
  - 计算过程的精确描述
    - 实现错误 → bug

- 调试 (debugging)
  - 已知程序有 bug，如何找到？

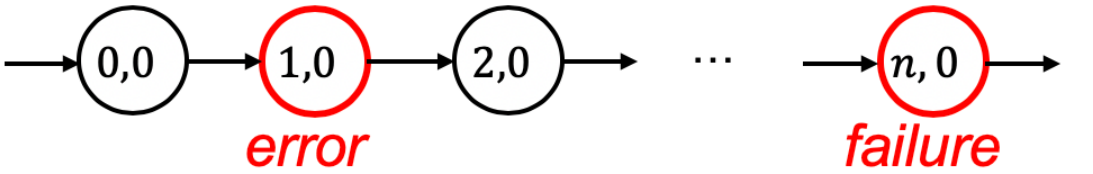


为什么debug那么困难

- 因为 bug 的触发经历了漫长的过程
  - 需求 → 设计 → 代码 → Fault (bug) → Error (程序状态错) → Failure
    - 我们只能观测到 failure (可观测的结果错)
    - 我们可以检查状态的正确性 (但非常费时)
    - 无法预知 bug 在哪里 (每一行 “看起来” 都挺对的)

```
for (int i = 0; i < n; i++)
  for (int j = 0; j < n; i++)
    ...
```

fault 程序bug



状态违反spec

可观测的timeout

调试理论

调试理论：如果我们能判定任意程序状态的正确性，那么给定一个 failure，我们可以通过二分查找定位到第一个 error 的状态，此时的代码就是 fault (bug)。

- 调试理论：推论
  - 为什么我们喜欢 “单步调试” ？
    - 从一个假定正确的状态出发
    - 每个语句的行为有限，容易判定是否是 error
  - 为什么调试理论看起来没用？
    - 因为判定程序状态的正确性非常困难
      - (是否在调试 DP 题/图论算法时陷入时间黑洞？)

调试理论（cont'd）

- 实际中的调试：通过观察程序执行的轨迹 (trace)
  - 缩小错误状态 (error) 可能产生的位置
  - 作出适当的假设
  - 再进行细粒度的定位和诊断

- 最重要的两个工具
  - printf → 自定义log的trace
    - + 灵活可控、能快速定位问题大概位置、适用于大型软件
    - - 无法精确定位、大量的logs管理起来比较
  - gdb → 指令/语句级trace
  - + 精确、指令级定位、任意查看程序内部状态
  - - 耗费大量时间

本学期课到此结束！

答疑环节

欢迎给我提各种意见和建议
我会持续改进！

1班

Scan the QR or use link to join

https://forms.office.com/r/46d-cft9Ud3

Copy link

2班

Scan the QR or use link to join

https://forms.office.com/r/zx1aEY1cuF

Copy link

3班

Scan the QR or use link to join

https://forms.office.com/r/J64i-TanMXK

Copy link