

clang



shell

```
1 | clang -o hello hello.c
2 |
```

clang——编译器

-o 输出

hello 输出文件名

hello.c输入文件

当用到现成的库的时候



shell

```
1 | clang -o hello hello.c -lcrypt
```

告诉计算机现成的库在哪里

最终这些被自动化（封装..? ）为make

```
~/ $ make hello
clang -ggdb3 -O0 -std=c11 -Wall -Werror -Wextra -Wno-sign-compare -Wno-unused-parameter -Wno-unused-variable -Wshadow hello.c -lcrypt -lcrypt -lm -o hello
```

如何编译代码

1预处理，寻找含有“#”的行（头文件），然后将写好的函数导入该地方

```
string get_string(string prompt);
int printf(string format, ...);

int main(void)
{
    string name = get_string("What's your name? ");
    printf("hello, %s\n", name);
}
```

2含义：将源代码变成汇编代码，汇编代码更接近机器底层，几十年前人们写的就是这种代码；再然后转化为机器码（01）

```
...
main:
    .cfi_startproc
# BB#0:
    pushq    %rbp
.Ltmp0:
    .cfi_def_cfa_offset 16
.Ltmp1:
    .cfi_offset %rbp, -16
    movq     %rsp, %rbp
.Ltmp2:
    .cfi_def_cfa_register %rbp
    subq     $16, %rsp
    xorl     %eax, %eax
    movl     %eax, %edi
    movabsq   $.L.str, %rsi
    movb     $0, %al
    callq    get_string
    movabsq   $.L.str.1, %rdi
    movq     %rax, -8(%rbp)
    movq     -8(%rbp), %rsi
    movb     $0, %al
    callq    printf
```

头文件已经有了，有点类似python的包...？

库文件已经被预编译了

debug

好家伙出处是计算机物理状态里的一个虫子吗

printf()是一个很好的debug函数！

```

int main(void)
{
    // Print 10 hashes
    for (int i = 0; i <= 10; i++)
    {
        printf("i is now %i\n")
        printf("#\n");
    }
}

```

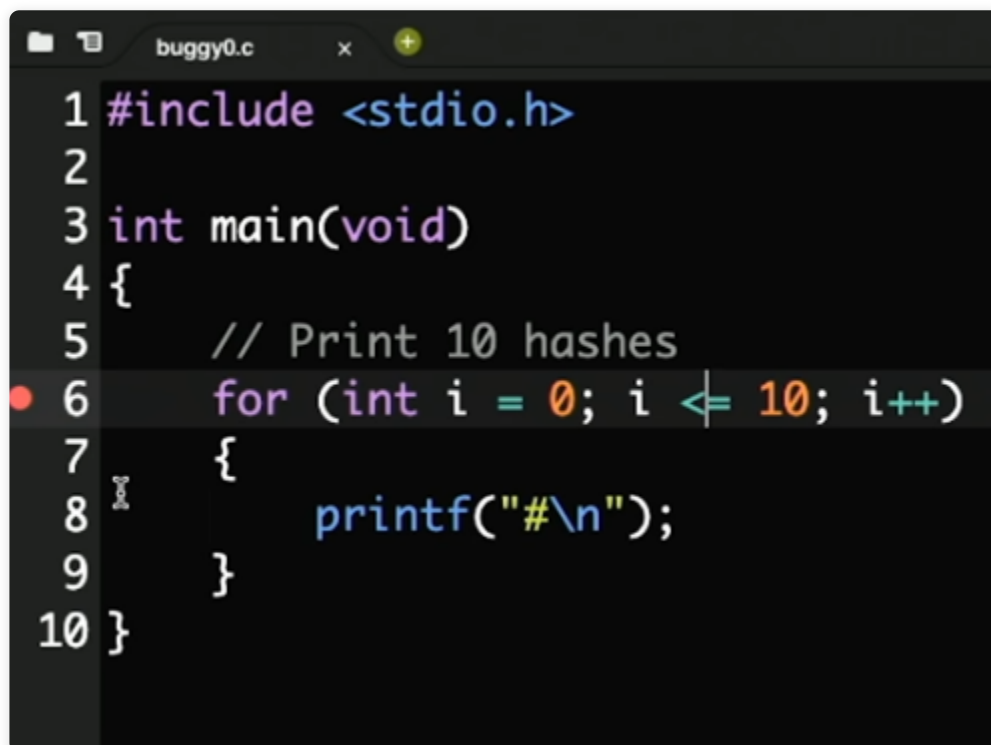
本来要打印10hashes，但是打印了11个，通过printf（）追踪i的值找到问题

debug50

```
~/ $ debug50 ./buggy0
```

上下箭头可以调出历史输入命令

打断点：单击行数

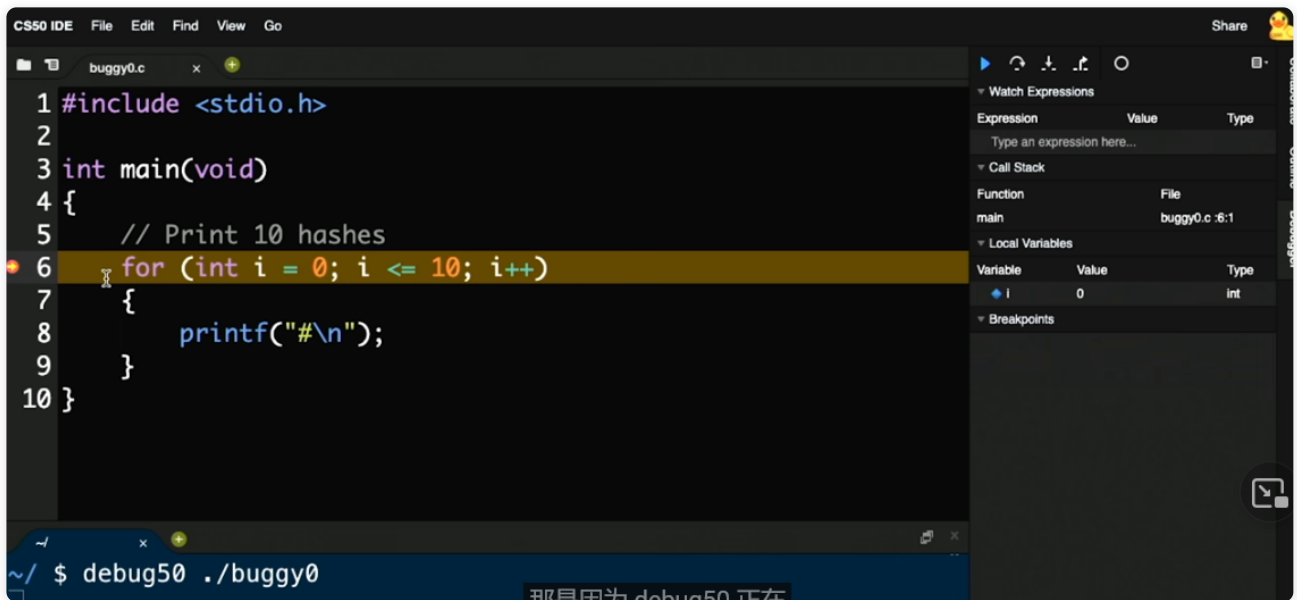


```

buggy0.c
1 #include <stdio.h>
2
3 int main(void)
4 {
5     // Print 10 hashes
6     for (int i = 0; i <= 10; i++)
7     {
8         printf("#\n");
9     }
10 }

```

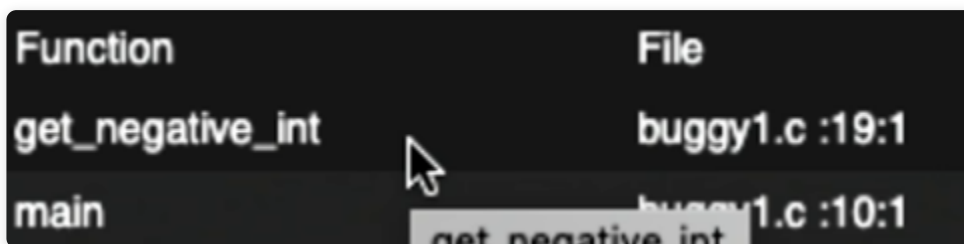
右边面板打开



step over: 逐行执行代码，黄色行就是要执行的

step into: 进入一个有意义的行...?

call stack (调用堆栈): 看哪个函数被调用



橡皮鸭debug

对橡皮鸭说你的代码...?

内存

不同的数据类型占据的byte

```
bool    1 byte
char    1 byte
double  8 bytes
float   4 bytes
int     4 bytes
long    8 bytes
string  ? bytes
...
```

RAM

arrays数组

```
int scores[3];

scores[0] = 72;
scores[1] = 73;
scores[2] = 33;
```

*常数功能

```
const int TOTAL = 3;
```

变量类型更改*int除int，需要将其中的一个变为float



C

```
1 | int a=114
2 | ...
3 | (float) a
```

一个应用



C

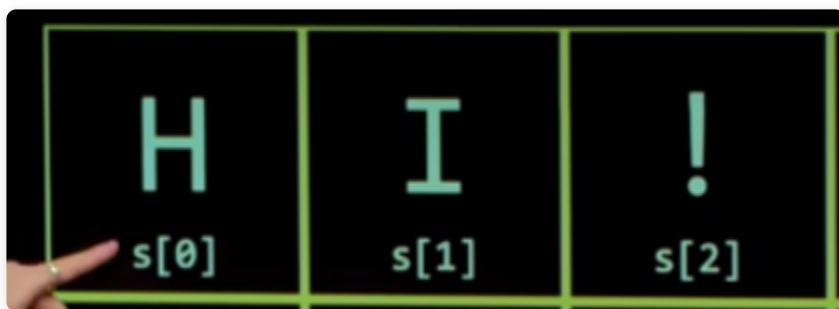
```
1 | #include
2 | #
3 | const int TOTAL=4;
4 | int main(void)
5 | {
6 |     int score[TOTAL]; // 创建一个数组，并限定其长度
7 |     for(int i=0; i<TOTAL; i++)
8 |     {
9 |         score[i]= get_int("score:");
10 |    }
11 |    printf("average:%f\n", average(TOTAL, score))
12 | }
13 | float average(int length, int array[]) // 但在自定义函数输入的数组中并不需要给出其长度
14 | {
15 |     int sum=0;
16 |     for (int i=0; i<length; i++)
17 |     {
18 |         sum=sum+array[i];
19 |     }
20 |     return sum/(float)length;
21 | }
```

不仅仅是int

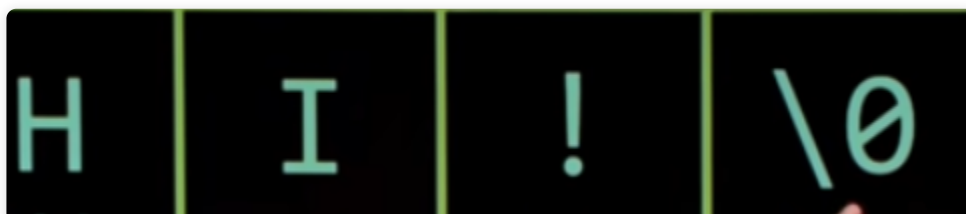
char c——> (int) c，即可看到对应字符的ASCII码，甚至不用谢那个 (int)

string

string的实现，数组



但如何知道这个数组在哪里终止？用n+1个byte,



\0表示空

但，我们甚至也可以达到第五个，即s[4]，这是一个危险的特性

```

1 #include <cs50.h>
2 #include <stdio.h>
3
4 int main(void)
5 {
6     string s = "HI!";
7
8     printf("%i %i %i %i %i\n", s[0], s[1], s[2], s[3], s[4]);
9 }

```

```

~/ $ ./hi
72 73 33 0 37

```

可以更改任意内存，甚至不一定是属于我们的，或许会导致程序崩溃...



C

1 | s[i] != '\0' // 条件式，代表只要数组中的i项不为0，即返回true

or已经做好的函数



C

```

1 | #include <string.h>
2 | strlen (s)

```

数组套娃

words[0][1], 意思是word中第一项的string的第二个字母

大小写转换

大小写判定 $c > a \ \&\& \ c \leq z$, 即说明是小写字母

s[i]-32, 即将小写转换为大写

or 用ctype.h, islower () 函数判断是否小写, toupper () 转换

[一些c语言里常见的函数](#)

自己从命令行里获得输出

```
1  #include <stdio.h>
2  #include <cs50.h>
3  int main(int argc, string argv[])
4  {
5      if (argc==2)
6      {
7          printf("hello,%s\n", argv[1]);
8      }
9      else
10     {
11         printf("hello,world\n");
12     }
13 }
```

```
~/ $ ./input
hello,world
~/ $ ./input sjiang
hello,sjiang
```

argv技术上是一切从命令行上输入的东西, 比如改成argv[0],就是./input、

为什么main总是返回一个int?

0表示没有出错, 其他的都是错误码

eg 404不存在


```
#include <cs50.h>
#include <stdio.h>

int main(int argc, string argv[])
{
    if (argc != 2)
    {
        printf("missing command-line argument\n");
        return 1;
    }
    printf("hello, %s\n", argv[1]);
}
```

正确输入一个词，则可以hello+名字（补一个return 0），但如果没有，返回报错

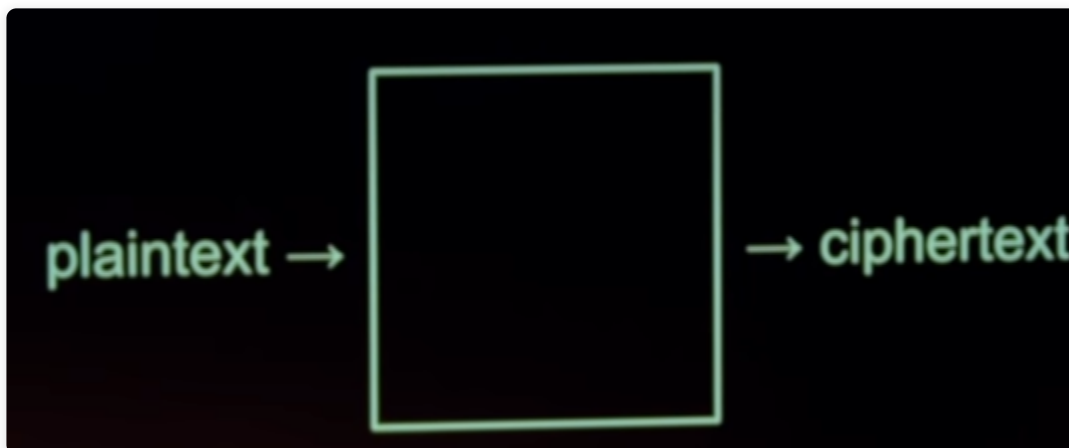


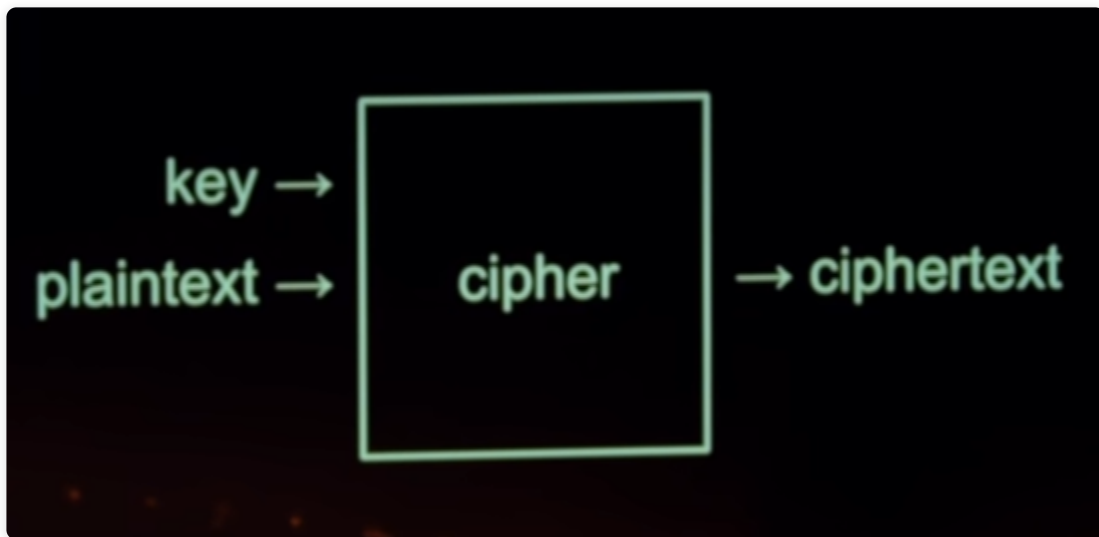
shell

1 | echo \$?

是看到返回数字的指令

密码学





就像之前改变string一样，不过密码学可能更复杂

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