

# 预处理命令与结构体

胡船长

初航我带你，远航靠自己

# 本期内容

1. 第一节：预处理命令
2. 第二节：宏定义-课后实战题
3. 第三节：结构体、联合体与枚举类型
4. 第四节：【附加内容】链表
5. 第五节：实现一种有趣的链表结构

# 一、预处理命令

- 1. 重新认识程序的『编译』过程
- 2. 认识：预处理命令家族
- 3. 必知必会：宏定义
- 4. 必知必会：条件编译

## 二、宏定义-课后实战题

1. 带等级的日志打印功能
2. 统计函数运行时间
3. 让 C 函数支持默认参数
4. 【搞笑】写出像摩斯密码一样的程序

# 三、结构体、联合体与枚举类型

- 1. 结构体
- 2. 联合体
- 3. 枚举类型
- 4. 位域的概念与使用

# 四、附加内容-链表

- 1. 链表：结构讲解
- 2. 链表：代码演示

# 五、实现一种有趣的链表结构

1. 设计一根『绳子』，将数据绑在上面
2. 从『字段地址』到结构体『首地址』
3. 梦想照进现实，实现有趣的链表结构

# 一、预处理命令

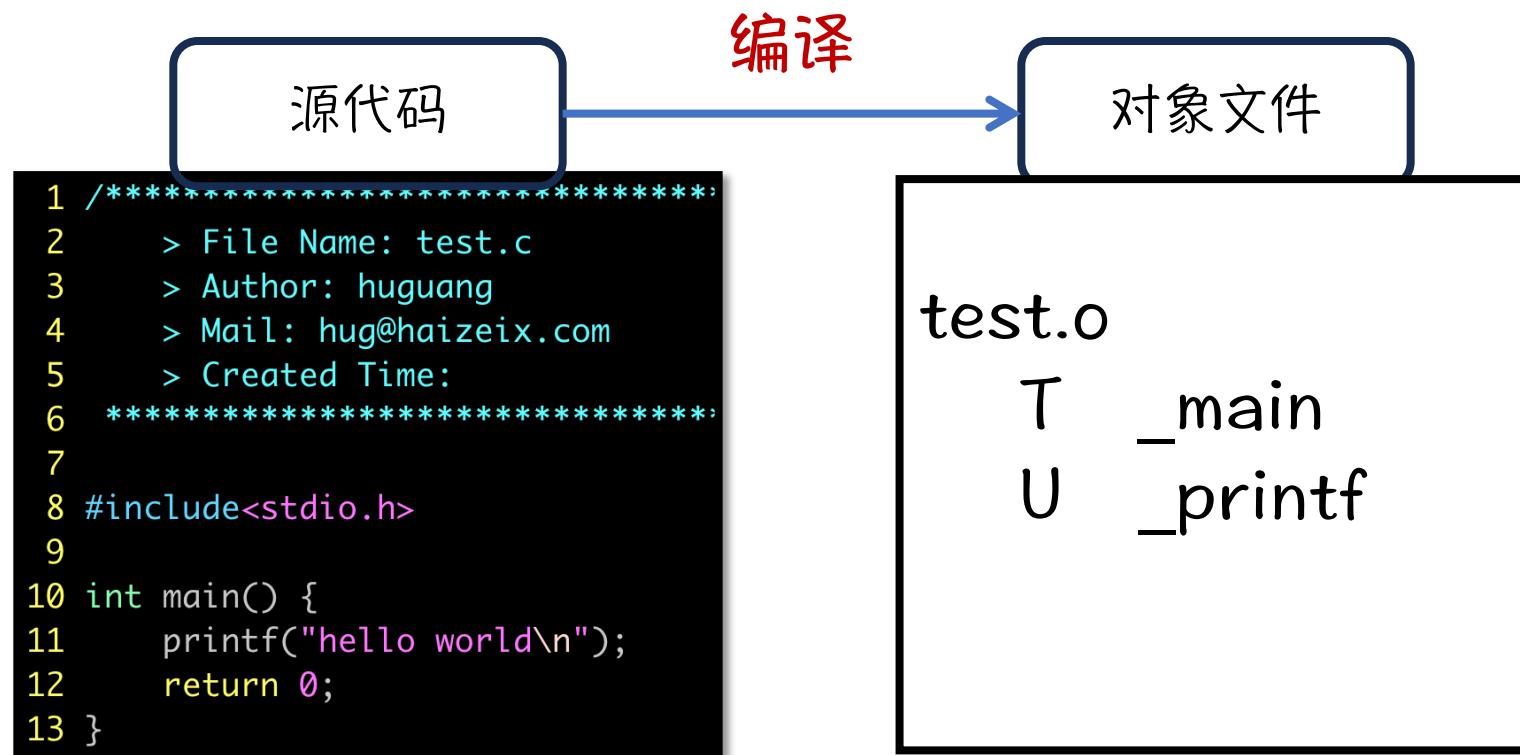
1. 重新认识程序的『编译』过程
2. 认识：预处理命令家族
3. 必知必会：宏定义
4. 必知必会：条件编译

# 重新认识程序的『编译』过程

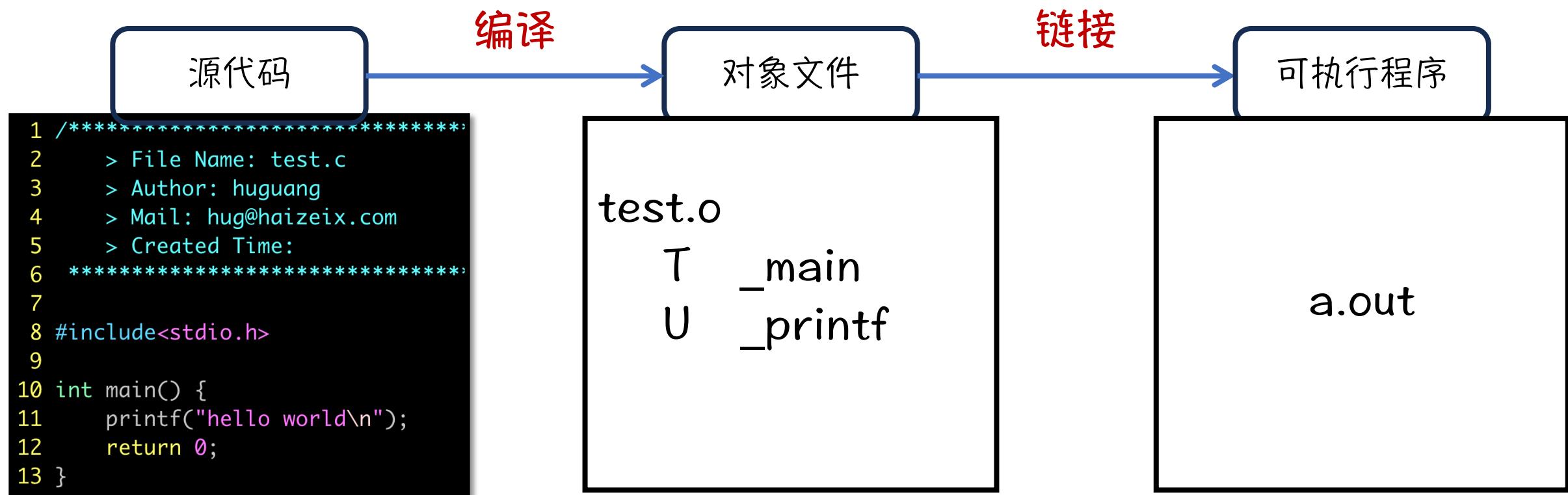
源代码

```
1 /*****  
2      > File Name: test.c  
3      > Author: huguang  
4      > Mail: hug@haizeix.com  
5      > Created Time:  
6 *****  
7  
8 #include<stdio.h>  
9  
10 int main() {  
11     printf("hello world\n");  
12     return 0;  
13 }
```

# 重新认识程序的『编译』过程



# 重新认识程序的『编译』过程



# 重新认识程序的『编译』过程



**编译** 检查是否有语法错误

**链接** 检查是否有定义缺失或冲突

# 聊聊：声明和定义

# 一、预处理命令

1. 重新认识程序的『编译』过程
2. 认识：预处理命令家族
3. 必知必会：宏定义
4. 必知必会：条件编译

# 认识：预处理命令家族

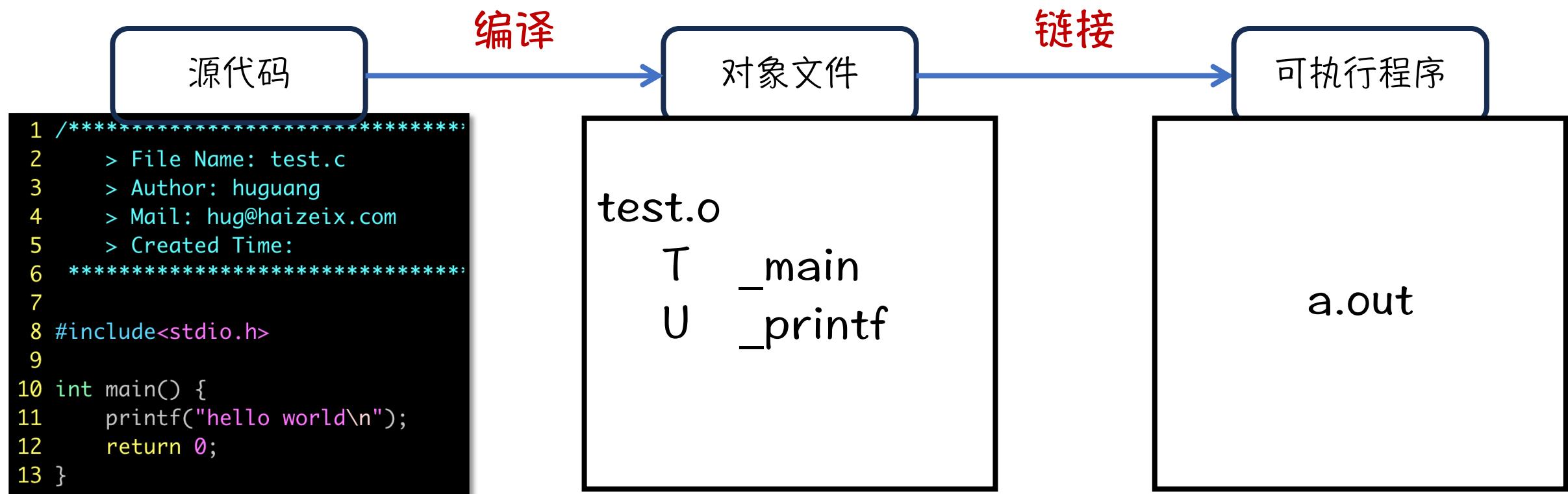
预处理命令家族的姓氏：**#**

# 认识：预处理命令家族

预处理命令家族的姓氏：**#**

**#include**

# 认识：预处理命令家族



# 认识：预处理命令家族



# 重新认识：#include



# 重新认识：#include

#include 做的事情就是『粘贴』



# 一、预处理命令

1. 重新认识程序的『编译』过程
2. 认识：预处理命令家族
3. **必知必会：宏定义**
4. 必知必会：条件编译

# 必知必会：宏定义

#include 做的事情就是『粘贴』

# 必知必会：宏定义

**#include** 做的事情就是『粘贴』

**#define** 做的事情就是『替换』

# 必知必会：宏定义

定义符号常量：

```
#define PI 3.1415926  
#define MAX_N 10000
```

定义傻瓜表达式：

```
#define MAX(a, b) (a) > (b) ? (a) : (b)  
#define S(a, b) a * b
```

定义代码段：

```
#define P(a) { \  
    printf("%d\n", a); \  
}
```

# 必知必会：宏定义

宏	说明
<code>__DATE__</code>	日期: Mmm dd yyyy
<code>__TIME__</code>	时间: hh:mm:ss
<code>__LINE__</code>	行号
<code>__FILE__</code>	文件名
<code>__func__</code>	函数名/ <b>非标准</b>
<code>__FUNC__</code>	函数名/ <b>非标准</b>
<code>__PRETTY_FUNCTION__</code>	更详细的函数信息/ <b>非标准</b>

# 必知必会：宏定义

宏定义中 # 和 ## 的作用

# 随堂练习题

请实现一个没有 BUG 的 MAX 宏，需要通过如下测试：

- 1、 $\text{MAX}(2, 3)$
- 2、 $5 + \text{MAX}(2, 3)$
- 3、 $\text{MAX}(2, \text{MAX}(3, 4))$
- 4、 $\text{MAX}(2, 3 > 4 ? 3 : 4)$
- 5、 $\text{MAX}(a++, 6)$  a 的初值为 7，返回值为 7，a 的值变为 8

# 一、预处理命令

1. 重新认识程序的『编译』过程
2. 认识：预处理命令家族
3. 必知必会：宏定义
4. 必知必会：条件编译

# 必知必会：条件编译

**#include** 做的事情就是『粘贴』

**#define** 做的事情就是『替换』

# 必知必会：条件编译

**#include** 做的事情就是『粘贴』

**#define** 做的事情就是『替换』

**#if** 做的事情就是『选择』

# 必知必会：条件编译

函数	说明
#ifdef DEBUG	是否定义了 DEBUG 宏
#ifndef DEBUG	是否没定义 DEBUG 宏
#if MAX_N == 5	宏 MAX_N 是否等于5
#elif MAX_N == 4	否则宏 MAX_N 是否等于4
#else	
#endif	

# 必知必会：条件编译

```
419 static void SetSysClock(void)
420 {
421 #ifdef SYSCLK_FREQ_HSE
422     SetSysClockToHSE();
423 #elif defined SYSCLK_FREQ_24MHz
424     SetSysClockTo24();
425 #elif defined SYSCLK_FREQ_36MHz
426     SetSysClockTo36();
427 #elif defined SYSCLK_FREQ_48MHz
428     SetSysClockTo48();
429 #elif defined SYSCLK_FREQ_56MHz
430     SetSysClockTo56();
431 #elif defined SYSCLK_FREQ_72MHz
432     SetSysClockTo72();
433 #endif
```

# 必知必会：条件编译

```
43 enum engine_id {  
44     E_RANDOM,  
45     E_REPLAY,  
46     E_PATTERNSCAN,  
47     E_PATTERNPLAY,  
48     E_MONTECARLO,  
49     E_UCT,  
50 #ifdef DISTRIBUTED  
51     E_DISTRIBUTED,  
52 #endif  
53 #ifdef JOSEKI  
54     E_JOSEKI,  
55 #endif  
56 #ifdef DCNN  
57     E_DCNN,  
58 #endif  
59     E_MAX,  
60 };
```

## 二、宏定义-课后实战题

1. 带等级的日志打印功能
2. 统计函数运行时间
3. 让 C 函数支持默认参数
4. 【搞笑】写出像摩斯密码一样的程序

# 带等级的日志打印功能

实现一个打印日志信息的 LOG 宏，需要输出所在函数及行号等信息。

注：

- 宏 `_FILE` 以字符串形式返回所在文件名称
- 宏 `_func` 以字符串形式返回所在函数名称
- 宏 `_LINE` 以整数形式返回代码行号

## 二、宏定义-课后实战题

1. 带等级的日志打印功能
2. 统计函数运行时间
3. 让 C 函数支持默认参数
4. 【搞笑】写出像摩斯密码一样的程序

# 统计函数运行时间

实现一个输出函数运行时间的 TIME 宏。

## 二、宏定义-课后实战题

1. 带等级的日志打印功能
2. 统计函数运行时间
3. 让 C 函数支持默认参数
4. 【搞笑】写出像摩斯密码一样的程序

# 让 C 函数支持默认参数

利用宏技巧，让 C 语言函数表现出支持默认参数的特性。

## 二、宏定义-课后实战题

1. 带等级的日志打印功能
2. 统计函数运行时间
3. 让 C 函数支持默认参数
4. 【搞笑】写出像摩斯密码一样的程序

# 写出像摩斯密码一样的程序

如何摧毁你同学的意志，消灭他一切抄你代码作业的欲望。

# 三、结构体、联合体与枚举类型

- 1. 结构体
- 2. 联合体
- 3. 枚举类型
- 4. 位域的概念与使用

# 结构体

```
1 struct person {  
2     char name[20];    // 姓名  
3     int age;         // 年龄  
4     char gender;     // 性别  
5     float height;    // 身高  
6 };
```



# 结构体

```
1 struct person {  
2     char name[20];    // 姓名  
3     int age;         // 年龄  
4     char gender;     // 性别  
5     float height;    // 身高  
6 };
```

0	<b>name</b>
19	...
20	<b>age</b>
21	
22	
23	
24	<b>gender</b>
25	<b>height</b>
26	
27	
28	

0	<b>name</b>
19	...
20	<b>age</b>
23	...
24	<b>gender</b>
25-27	空
28	<b>height</b>
29	
30	
31	

# 结构体-对齐补齐规则

1. 类型都有一个对齐值，内建类型的对齐值等于其自身大小
2. 结构体的对齐值，等于其成员中的最大对齐值
3. 成员被存储在其整数倍的对齐值位置上
4. 可以通过 `#pragma pack` 限制对齐值的最大值

```
1 struct person {  
2     char name[20]; // 姓名  
3     int age;      // 年龄  
4     char gender; // 性别  
5     float height; // 身高  
6 };
```

# 随堂练习题

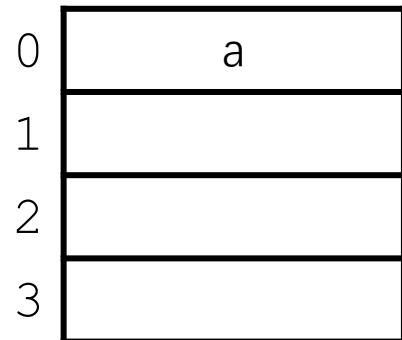
```
struct A {  
    char a;  
    int b;  
    short c;  
}
```

```
struct B {  
    char a;  
    short b;  
    int c;  
}
```

```
#pragma pack(1)  
struct C {  
    char a;  
    short b;  
    int c;  
}
```

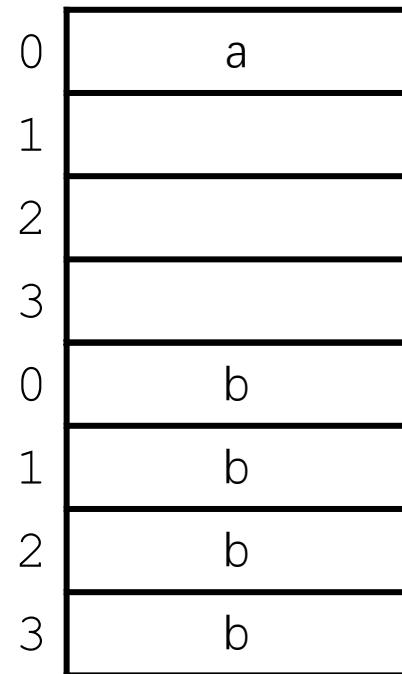
# 随堂练习题

```
struct A {  
    char a;  
    int b;  
    short c;  
}
```



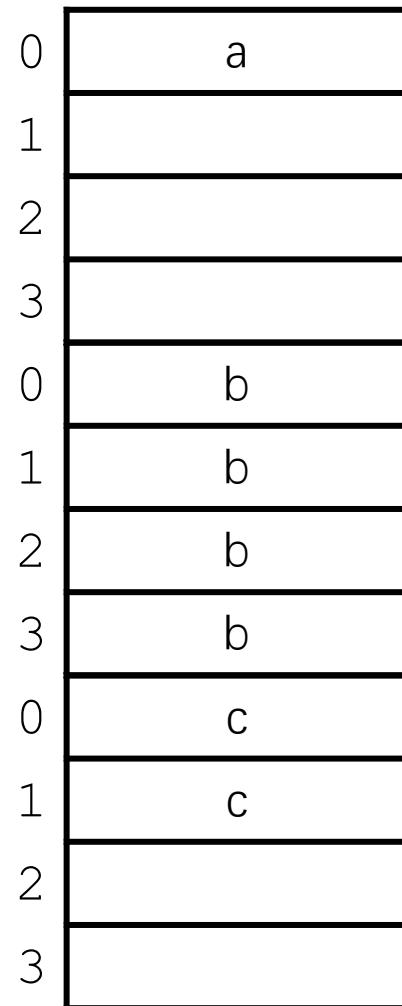
# 随堂练习题

```
struct A {  
    char a;  
    int b;  
    short c;  
}
```



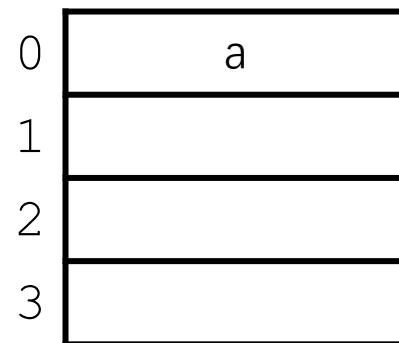
# 随堂练习题

```
struct A {  
    char a;  
    int b;  
    short c;  
}
```



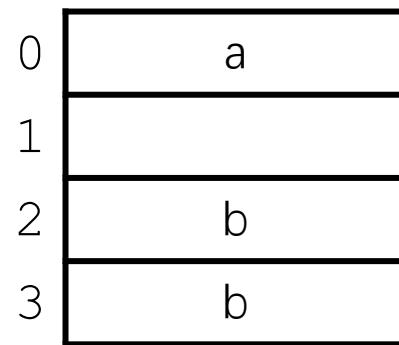
# 随堂练习题

```
struct B {  
    char a;  
    short b;  
    int c;  
}
```



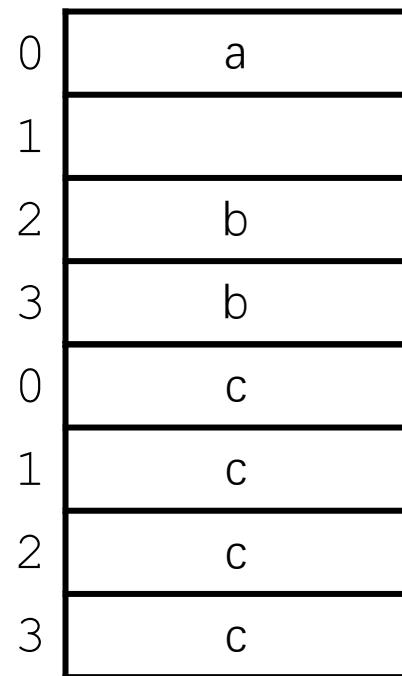
# 随堂练习题

```
struct B {  
    char a;  
    short b;  
    int c;  
}
```



# 随堂练习题

```
struct B {  
    char a;  
    short b;  
    int c;  
}
```



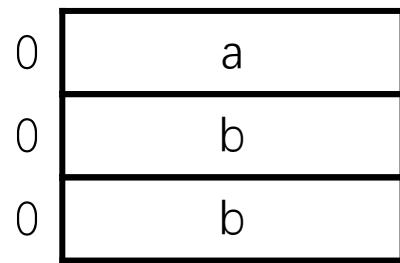
# 随堂练习题

```
#pragma pack(1)
struct C {
    char a;
    short b;
    int c;
}
```



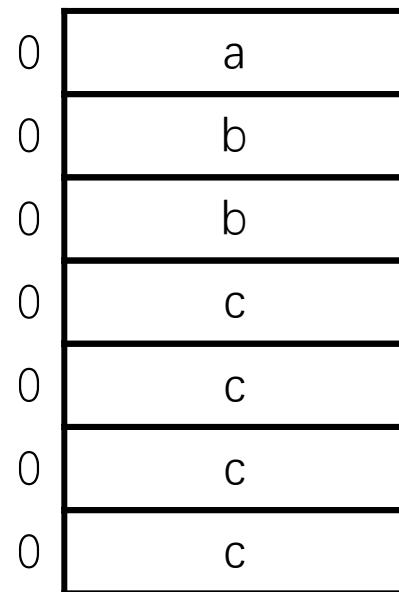
# 随堂练习题

```
#pragma pack(1)
struct C {
    char a;
    short b;
    int c;
}
```



# 随堂练习题

```
#pragma pack(1)
struct C {
    char a;
    short b;
    int c;
}
```



# 三、结构体、联合体与枚举类型

- 1. 结构体
- 2. 联合体
- 3. 枚举类型
- 4. 位域的概念与使用

# 联合体

```
1 union register {  
2     struct {  
3         unsigned char byte1;  
4         unsigned char byte2;  
5         unsigned char byte3;  
6         unsigned char byte4;  
7     } bytes;  
8     unsigned int number;  
9 };
```

0	bytes (byte1)	0	number
1	byte2	1	
2	byte3	2	
3	byte4	3	

# 随堂练习题

请仿照之前课程内容，画出 node 的内存占用结构图

```
12 union node {  
13     double a;  
14     char b;  
15     int c;  
16 };
```

# 三、结构体、联合体与枚举类型

- 1. 结构体
- 2. 联合体
- 3. 枚举类型
- 4. 位域的概念与使用

# 枚举类型

```
1 enum Number {  
2     zero,  
3     one,  
4     two,  
5     three,  
6     four,  
7     MAX_Number  
8 };
```

# 枚举类型

```
1 enum Number {  
2     zero,  
3     one,  
4     two,  
5     three,  
6     four,  
7     MAX_Number  
8 };
```

```
1 enum Week {  
2     Sunday,  
3     Monday,  
4     Therday,  
5     Wednesday,  
6     Thursday,  
7     Firday,  
8     Saturday  
9 };
```

# 枚举类型

```
1 enum Number {  
2     zero,  
3     one,  
4     two,  
5     three,  
6     four,  
7     MAX_Number  
8 };
```

```
1 enum Week {  
2     Sunday,  
3     Monday,  
4     Therday,  
5     Wednesday,  
6     Thursday,  
7     Firday,  
8     Saturday  
9 };
```

```
1 enum COLOR {  
2     RED = 31,  
3     GREEN,  
4     YELLOW,  
5     BLUE,  
6 };
```

# Printf 的有趣用法，输出彩色文字

\033[ A1;A2;A3;...An m

# Printf 的有趣用法，输出彩色文字

\033[ A1;A2;A3;...An m

通用格式控制	功能
0	重置所有属性
1	高亮/加粗
2	暗淡
4	下划线
5	闪烁
7	反转
8	隐藏

前景色	功能
30	黑色
31	红色
32	绿色
33	黄色
34	蓝色
35	品红
36	青色

背景色	功能
40	黑色
41	红色
42	绿色
43	黄色
44	蓝色
45	品红
46	青色

# Printf 的有趣用法，输出彩色文字

\033[ A1;A2;A3;...An m

通用格式控制	功能
0	重置所有属性
1	高亮/加粗
2	暗淡
4	下划线
5	闪烁
7	反转
8	隐藏

前景色	功能
30	黑色
31	红色
32	绿色
33	黄色
34	蓝色
35	品红
36	青色

背景色	功能
40	黑色
41	红色
42	绿色
43	黄色
44	蓝色
45	品红
46	青色

# 随堂练习题

请设计封装一套颜色输出的工具，至少设计两种方式。

```
1 enum COLOR {  
2     RED = 31,  
3     GREEN,  
4     YELLOW,  
5     BLUE,  
6 };
```

# 三、结构体、联合体与枚举类型

- 1. 结构体
- 2. 联合体
- 3. 枚举类型
- 4. 位域的概念与使用

# 随堂练习题

使用位域的相关技巧，实现输出整型数字的十六进制表示

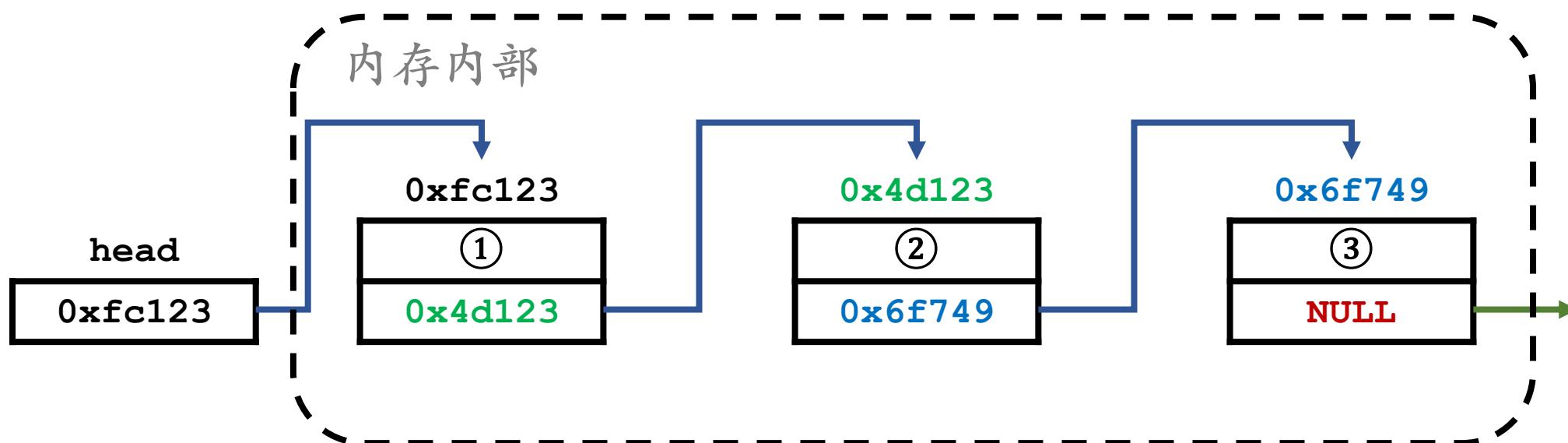
**注意：**不能使用 `%x` 和 `%X` 占位符

# 四、附加内容-链表

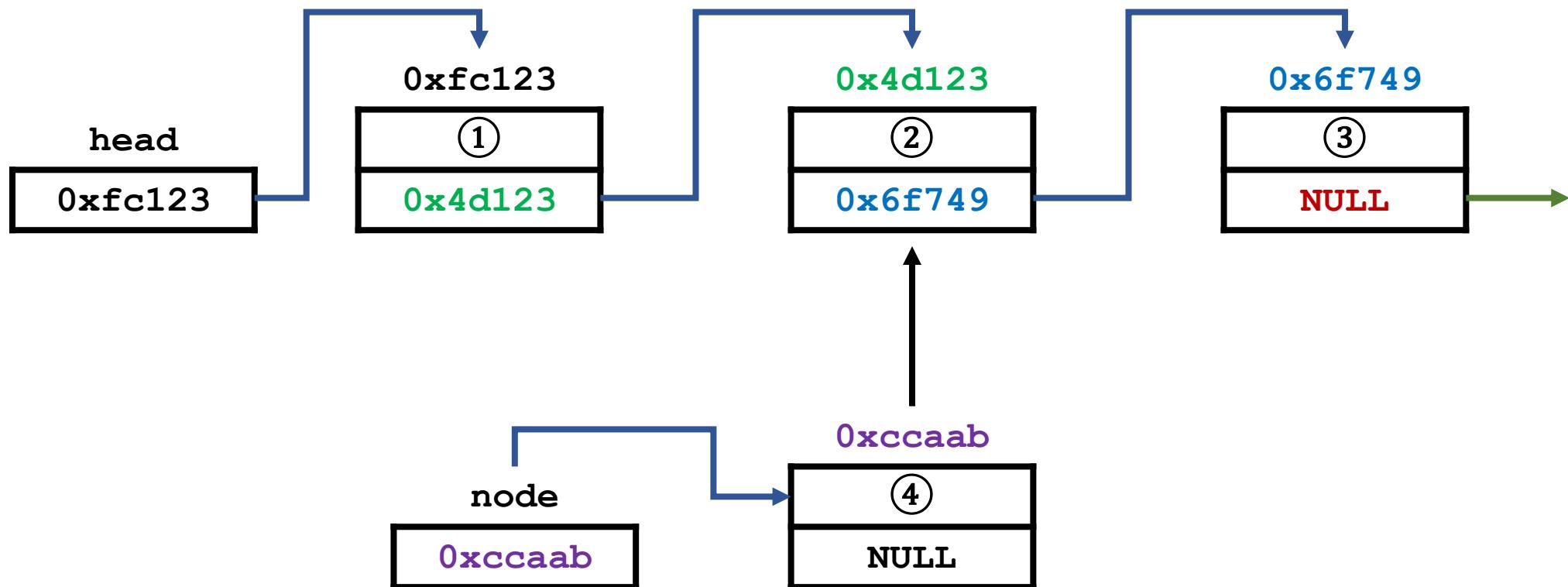
- 1. 链表：结构讲解
- 2. 链表：代码演示

# 链表：结构定义

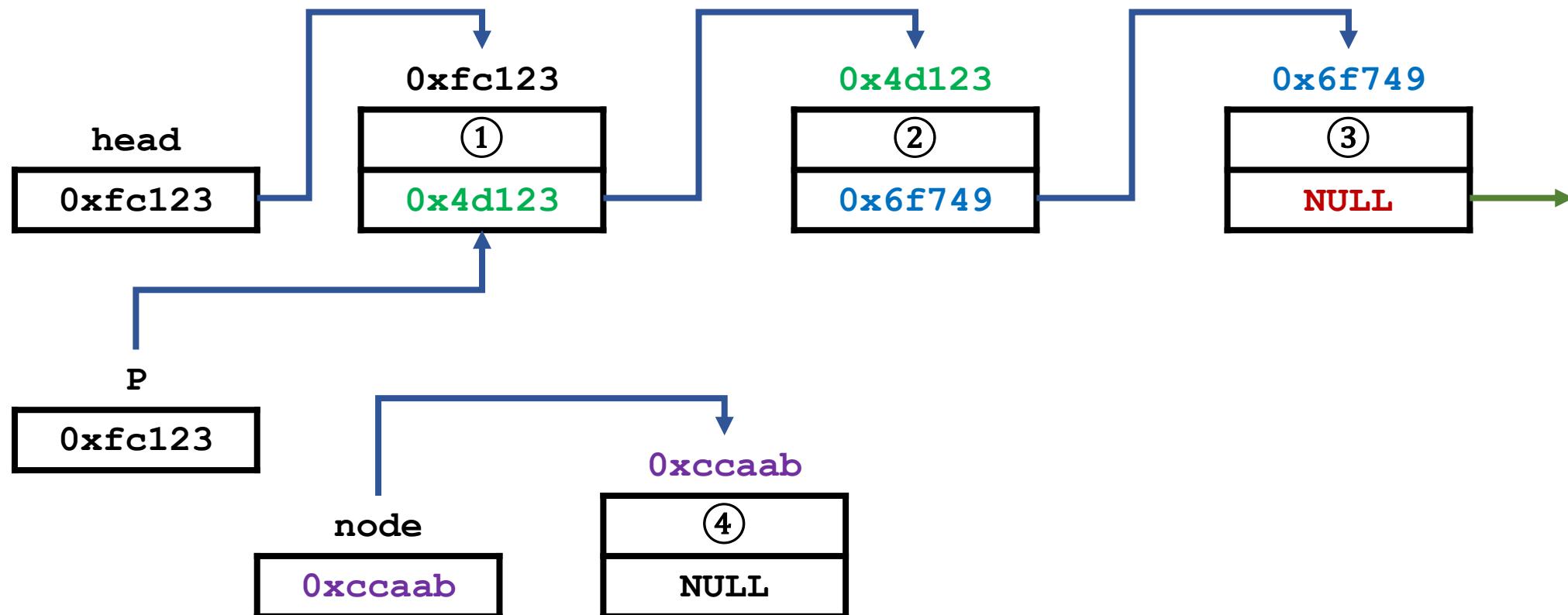
程序内部



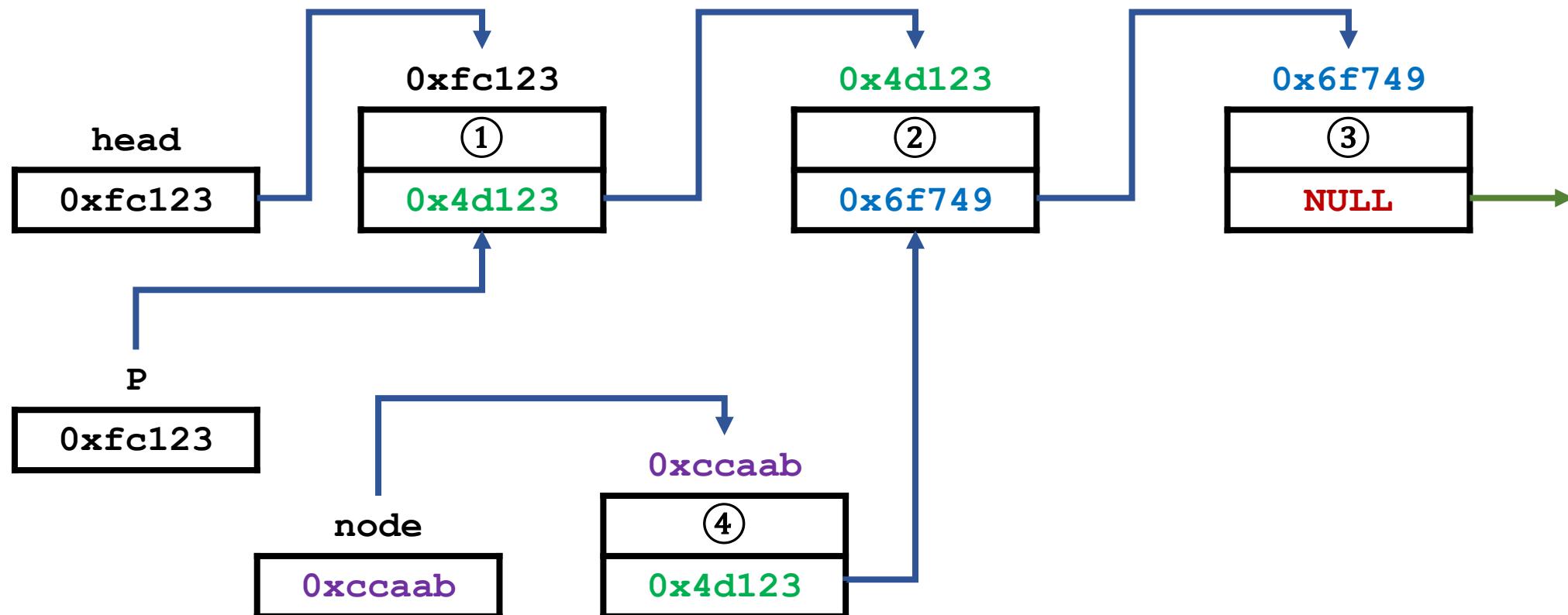
# 链表：插入元素



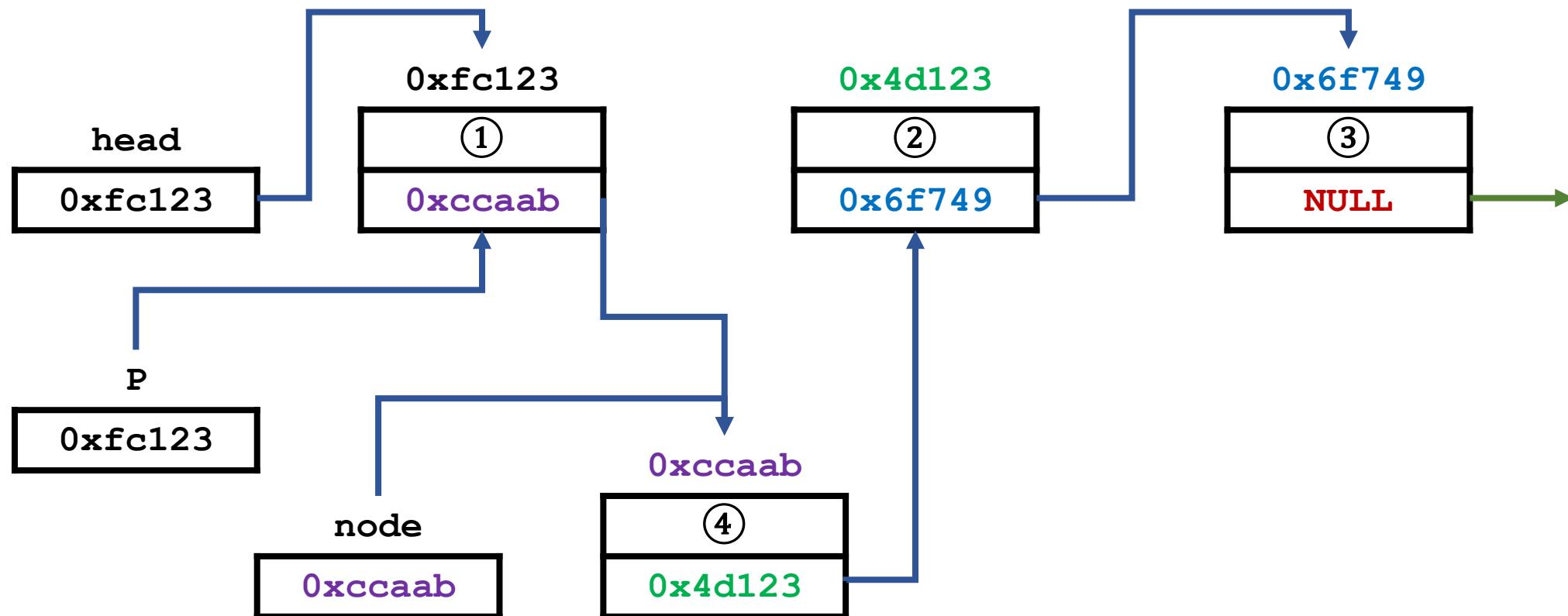
# 链表：插入元素



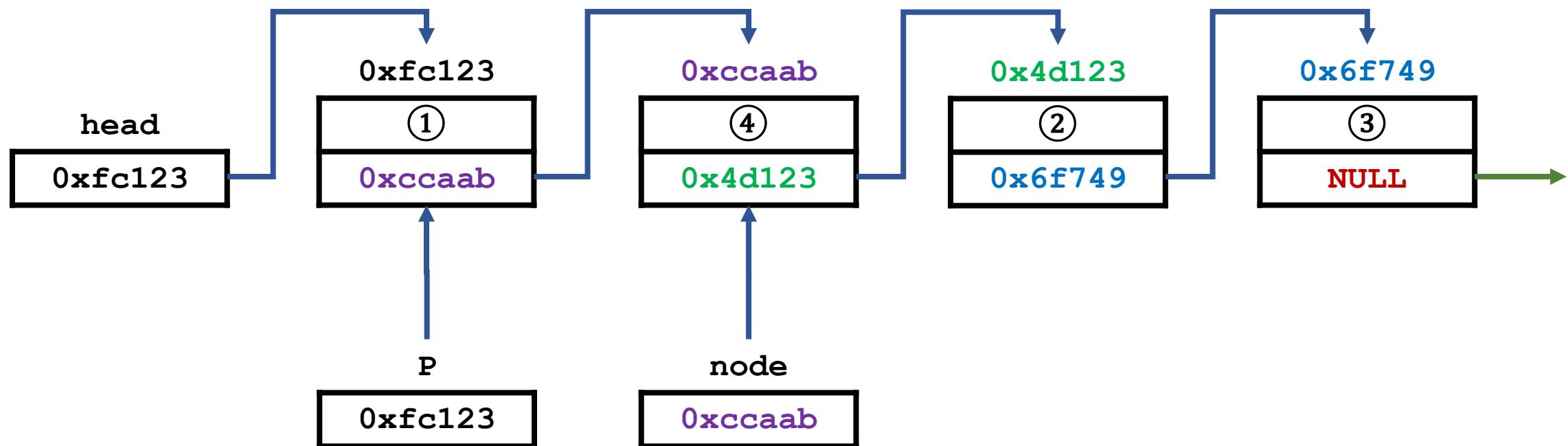
# 链表：插入元素



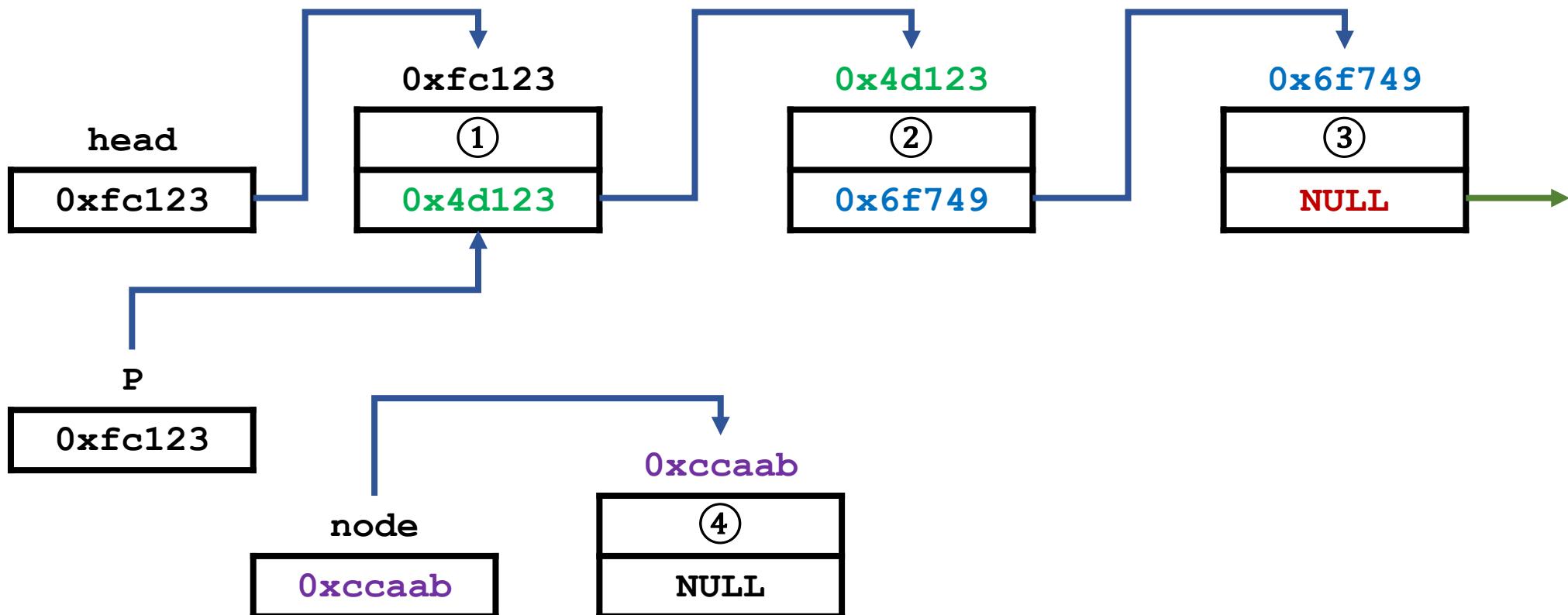
# 链表：插入元素



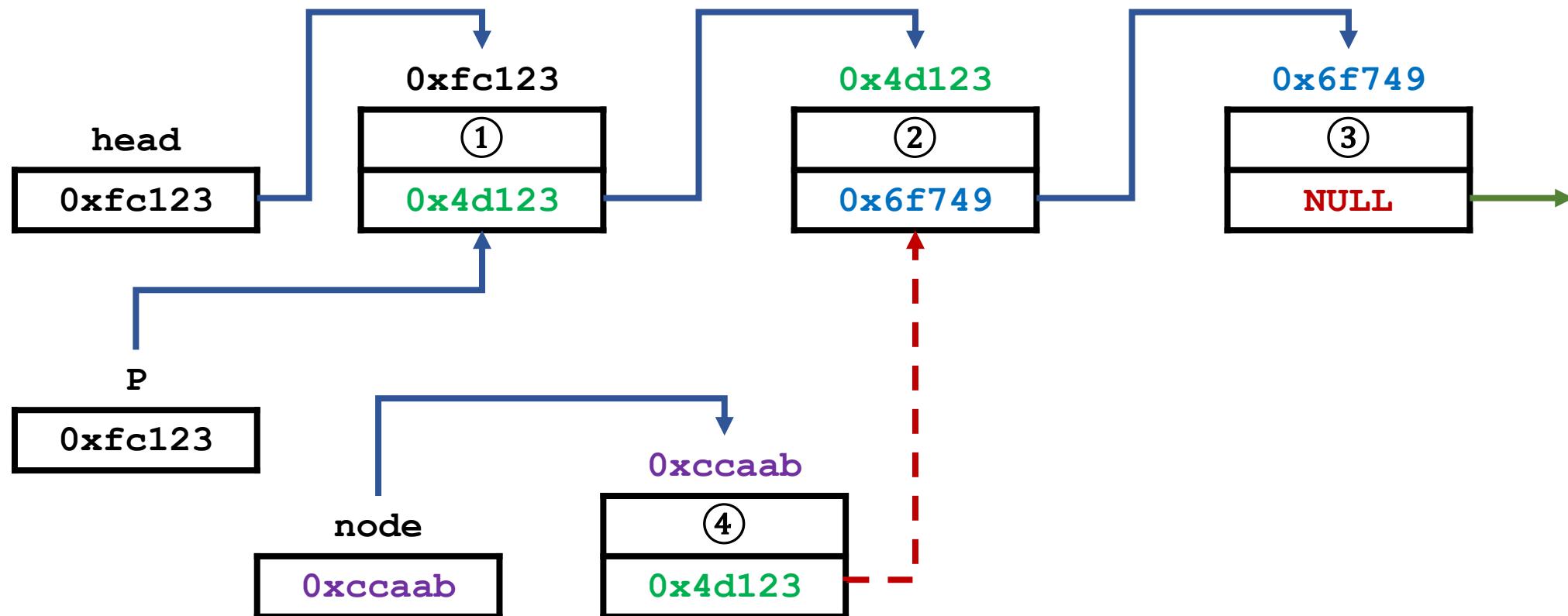
# 链表：插入元素



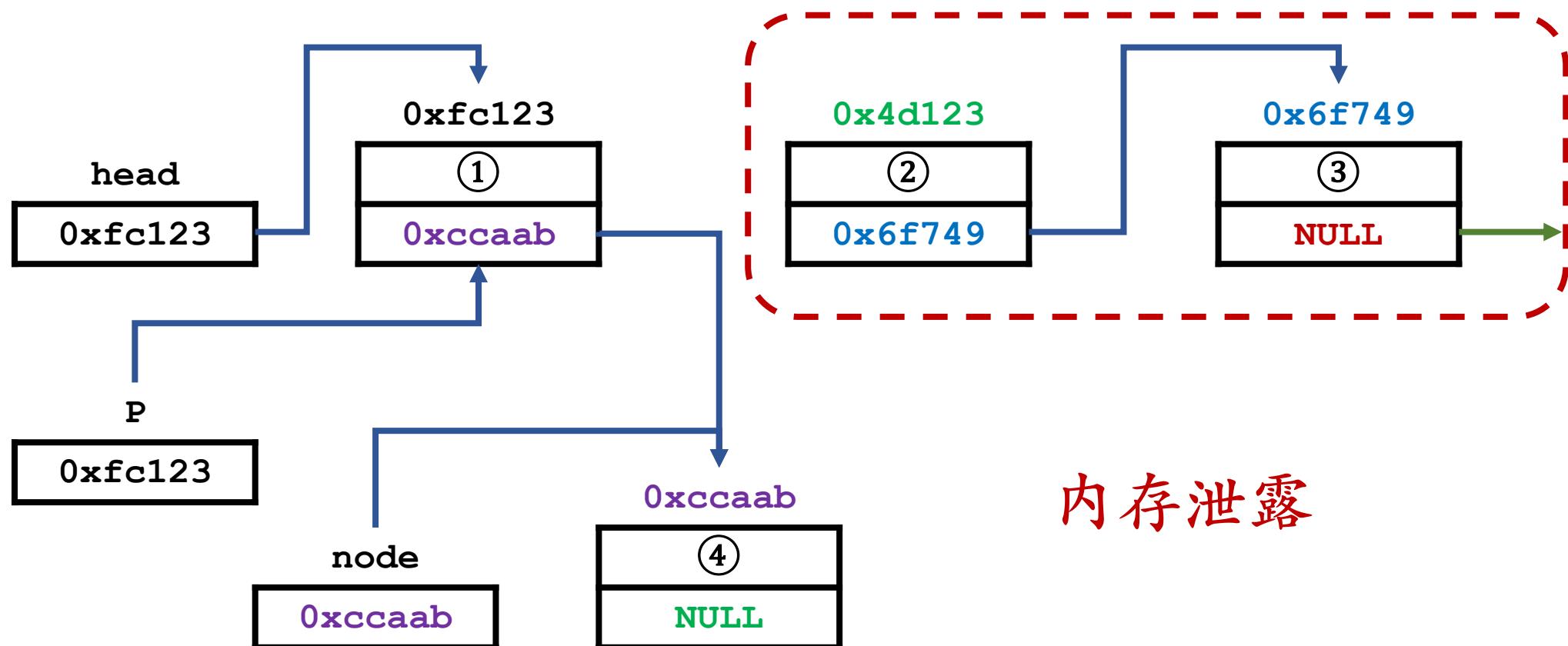
# 链表：错误插入



# 链表：错误插入

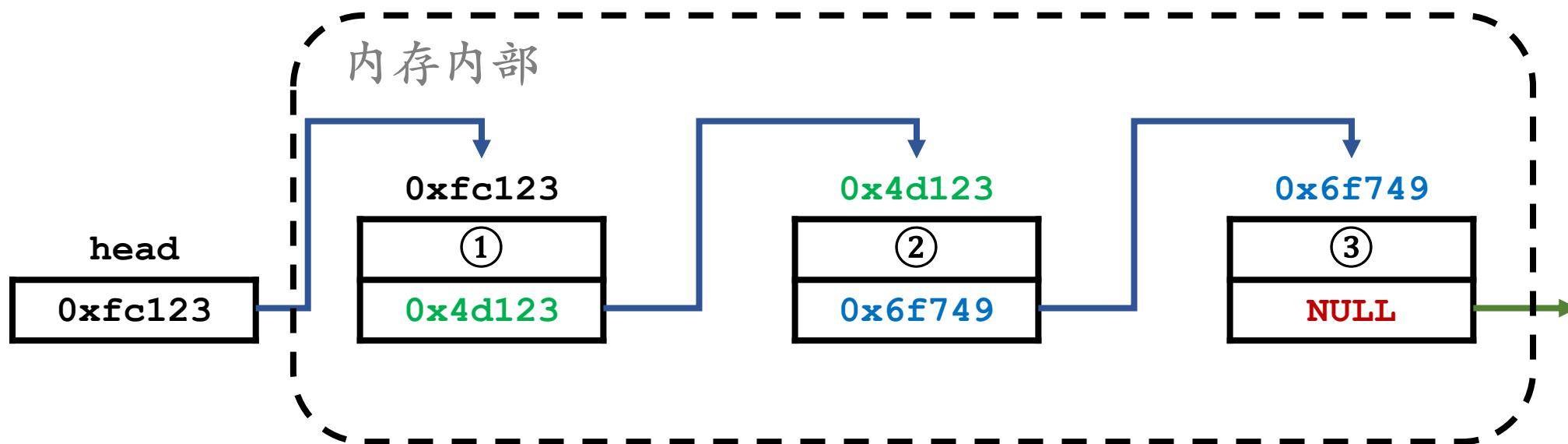


# 链表：错误插入



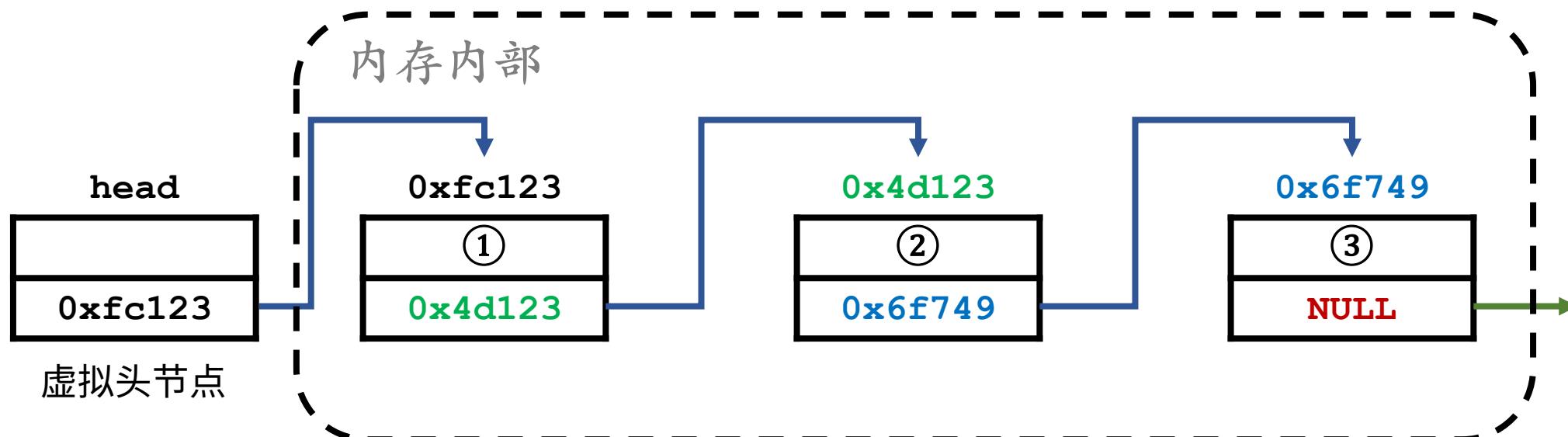
# 链表：无头链表

程序内部



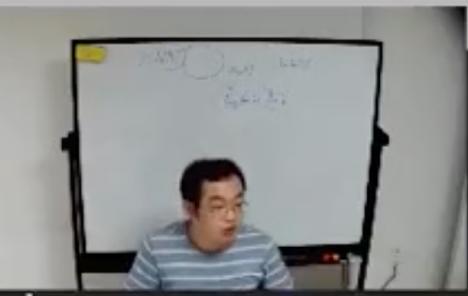
# 链表：有头链表

程序内部



1. vim

```
vim *1 bash *2 bash *3  
39 }  
40  
41 Node *insert_maintain(Node *root) {  
42     if (!hasRedChild(root)) return root;  
43     if (root->lchild->color == RED && root->rchild->color == RED)  
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;  
45         root->color = RED;  
46         root->lchild->color = root->rchild->color = BLACK;  
47         return root;  
48     }  
49     if (root->lchild->color == RED) {  
50         if (!hasRedChild(root->lchild)) return root;  
51  
52     } else {  
53         if (!hasRedChild(root->rchild)) return root;  
54     }  
55  
56 }  
57  
58 [ ]  
59  
60  
61 Node *__insert(Node *root, int key) {  
62     if (root == NIL) return getNode(key);
```

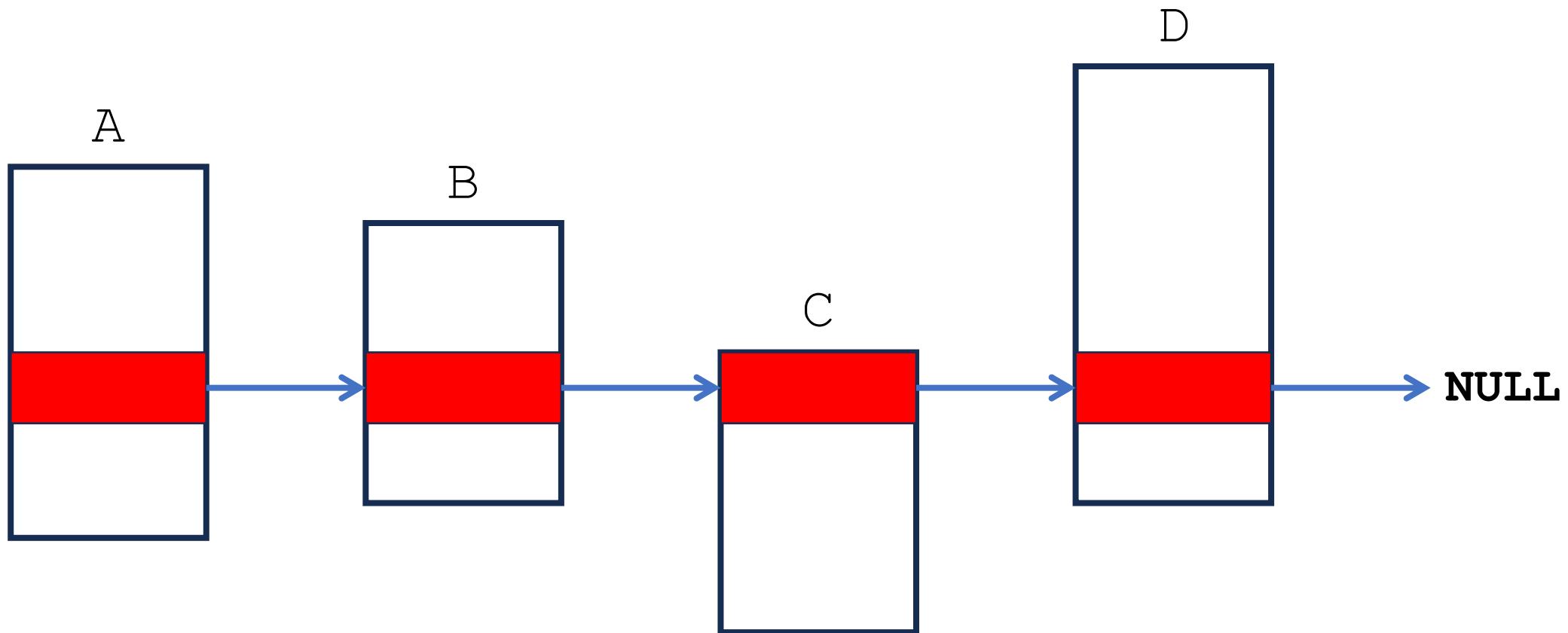


# 链表：代码演示

# 五、实现一种有趣的链表结构

1. 设计一根『绳子』，将数据绑在上面
2. 从『字段地址』到结构体『首地址』
3. 梦想照进现实，实现有趣的链表结构

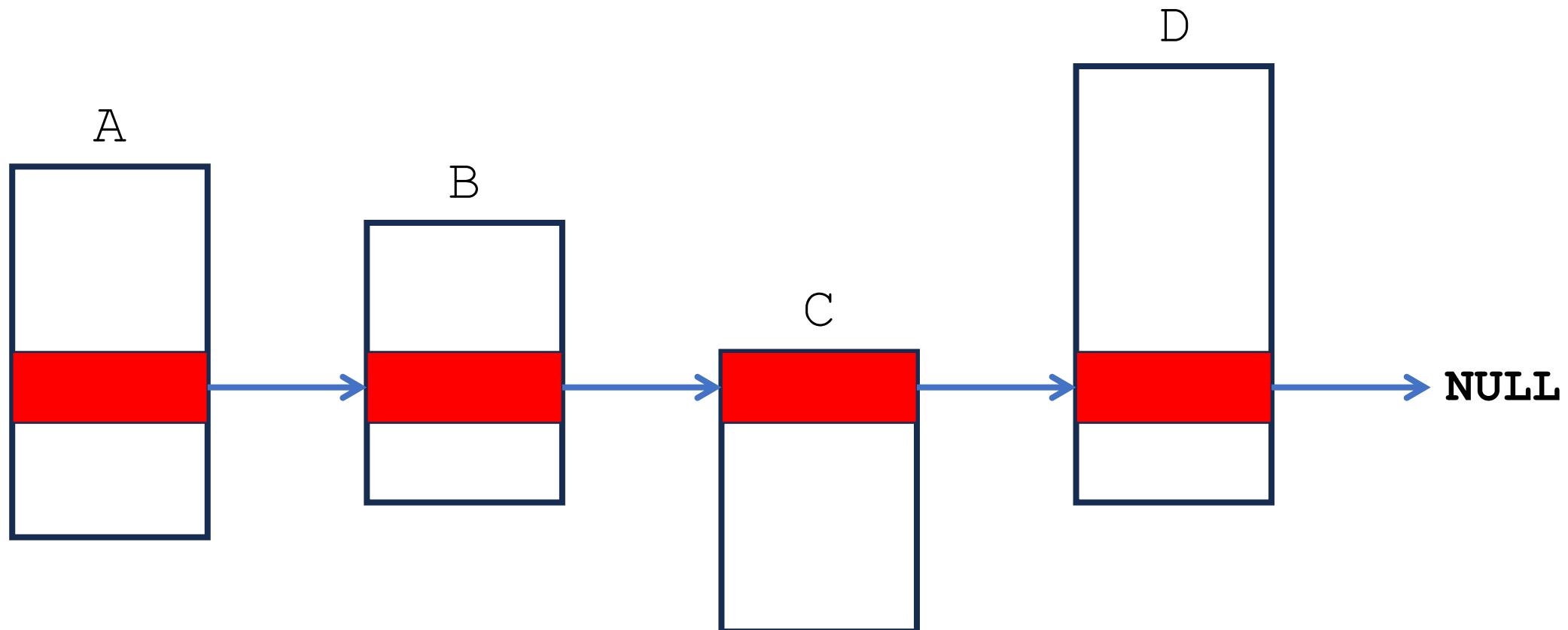
# 设计一根『绳子』



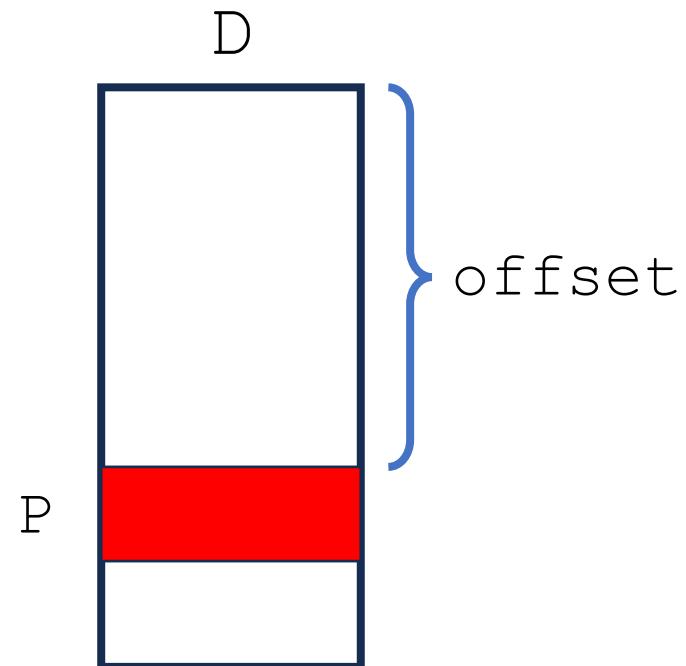
# 五、实现一种有趣的链表结构

1. 设计一根『绳子』，将数据绑在上面
2. 从『字段地址』到结构体『首地址』
3. 梦想照进现实，实现有趣的链表结构

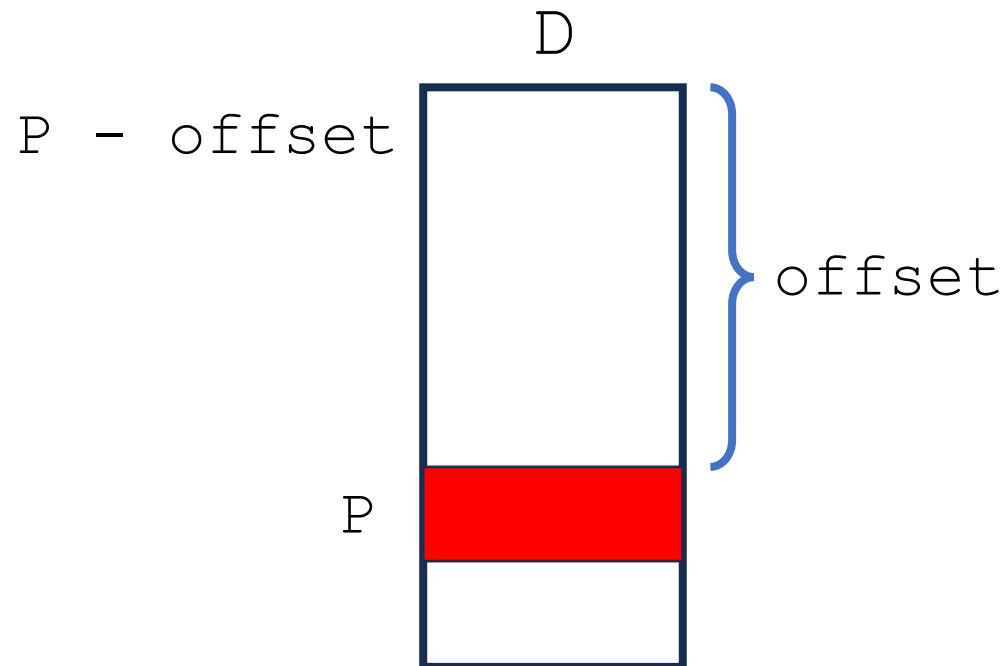
# 从『字段地址』到结构体『首地址』



# 从『字段地址』到结构体『首地址』



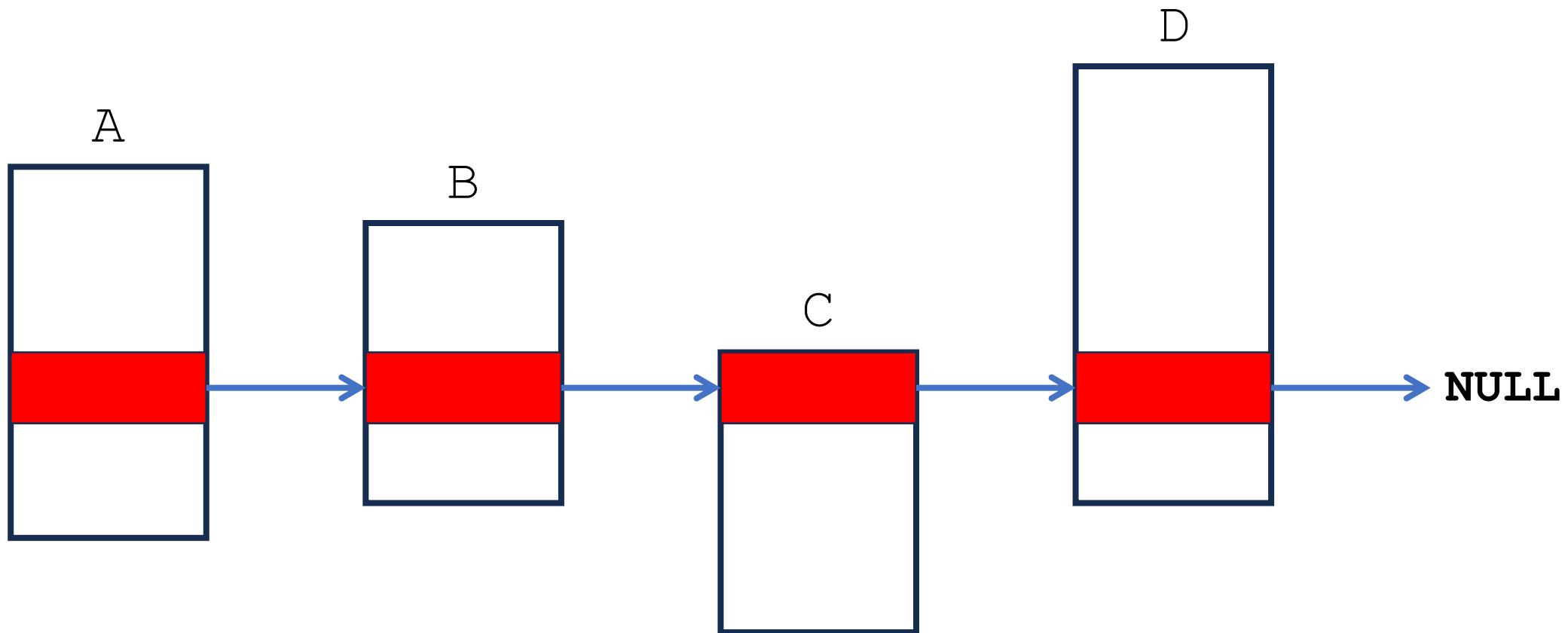
# 从『字段地址』到结构体『首地址』



# 五、实现一种有趣的链表结构

1. 设计一根『绳子』，将数据绑在上面
2. 从『字段地址』到结构体『首地址』
3. 梦想照进现实，实现有趣的链表结构

# 设计一根『绳子』



不要考虑太多，坚持看完，  
你就已经超过了95%的人。

5. 整型数据类型

 | 3.58万次播放

54. 主函数参数

 | 2892次播放