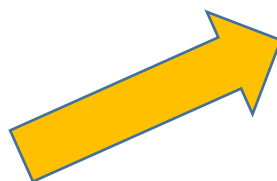


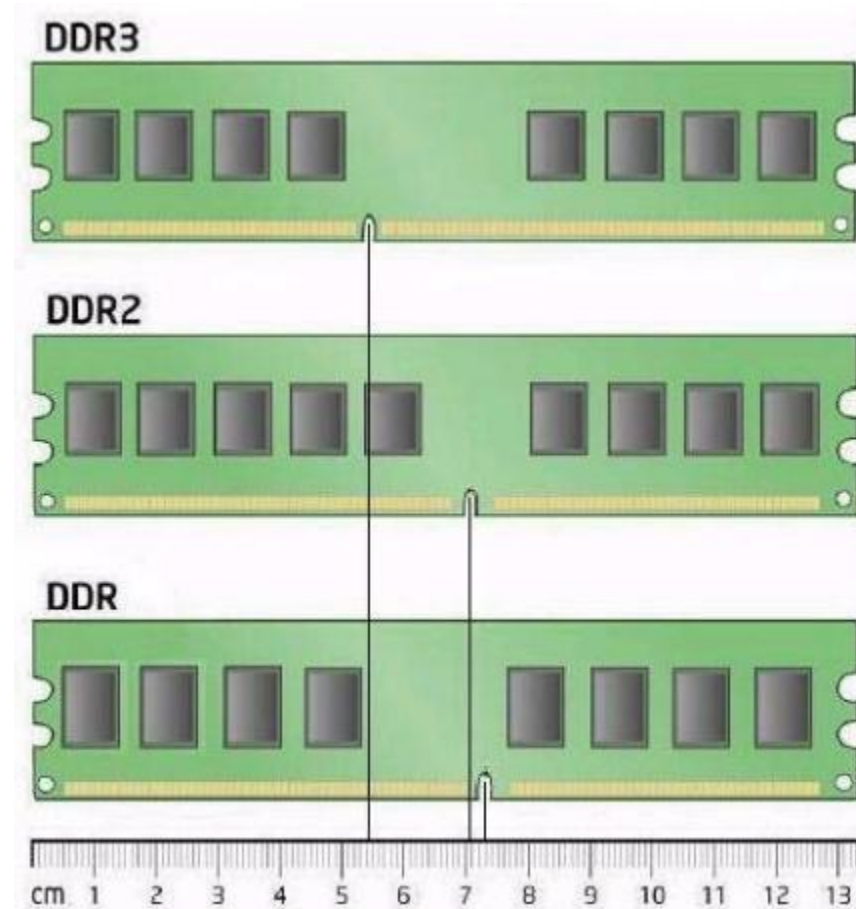
让计算机思考的每一个电压状态存储在哪呢？

——内存条

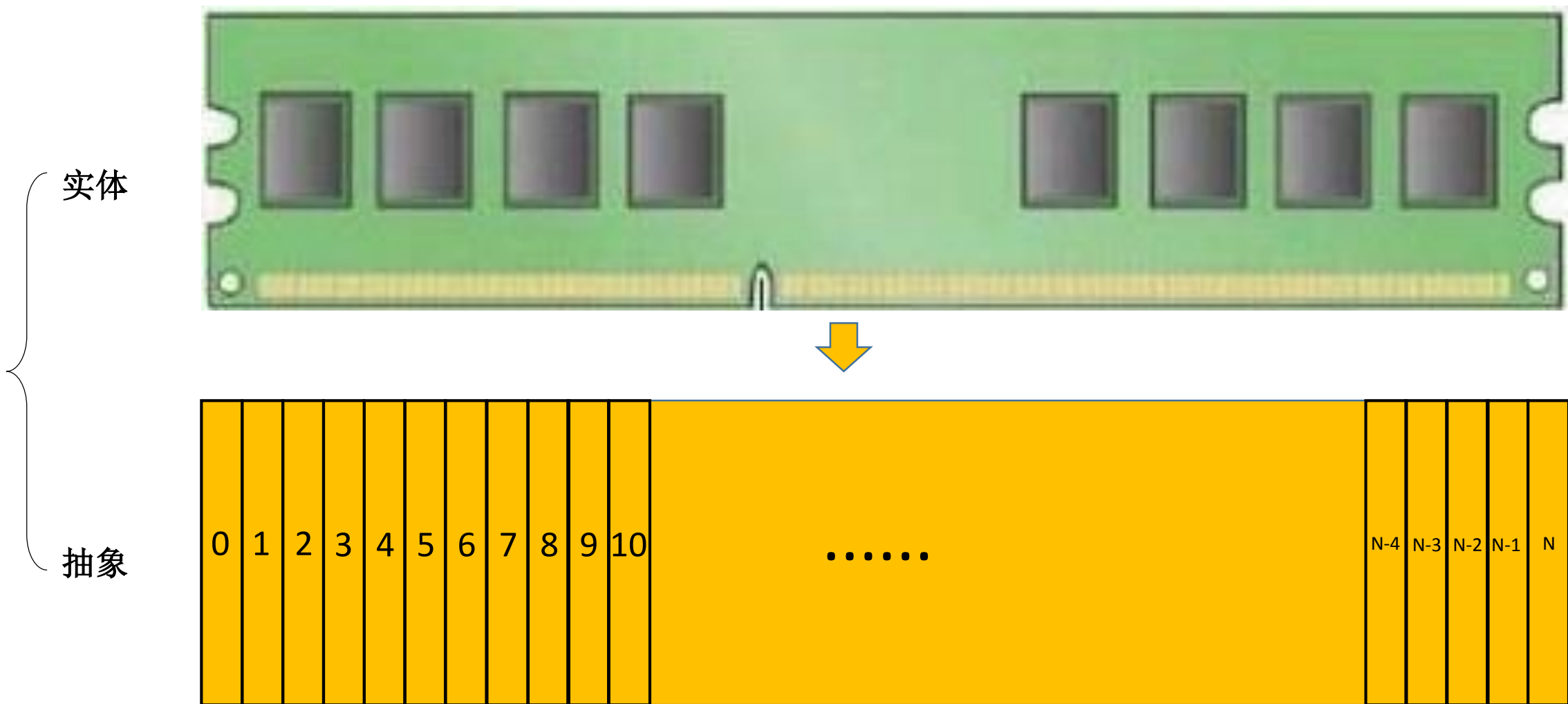
计算机主板



内存条

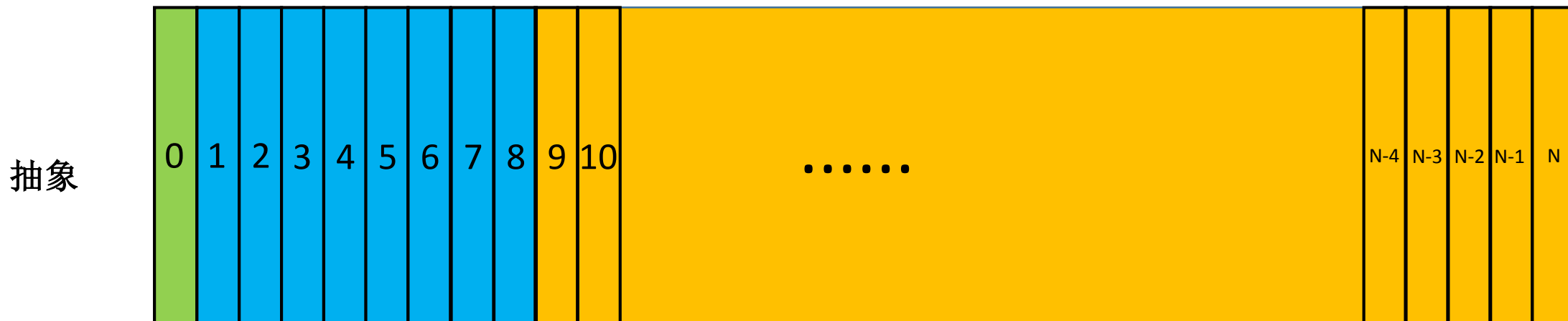


内存条：操作系统每时每刻在将不同电压信号传递过来
即思考不同的内容（计算任务），然后内存条传给CPU进行计算。



内存条可以抽象为具有N个空格的盒子，每个空格内存储1个电压状态（0或1）。

程序：将人类的问题（任务）进行逻辑顺序的描述，最后由操作系统转换为电压信号。



操作系统使用**统一的标准**来描述电压信号：

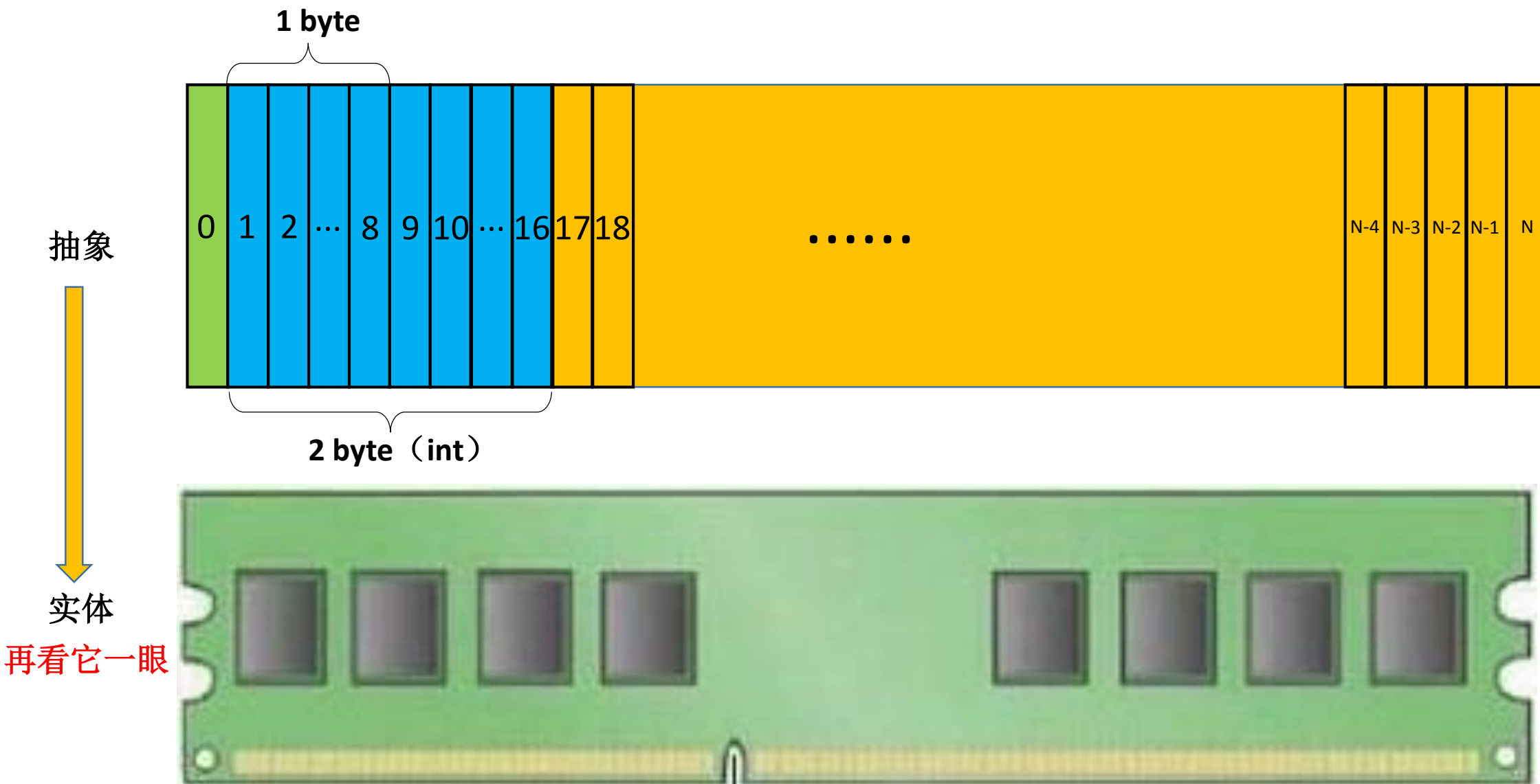
- 1) 位（bit）：1个电压信号（0或1），只能描述2种电压状态；
- 2) 字节（byte）：8个电压信号（8 bits），可能描述 $2^8=256$ 种电压状态；

256种电压状态：

- 256个数字
- 256个字符（ASCII字符表）
- 256个.....

但是，1个byte（256个状态）无法来描述所有的问题，比如数字。

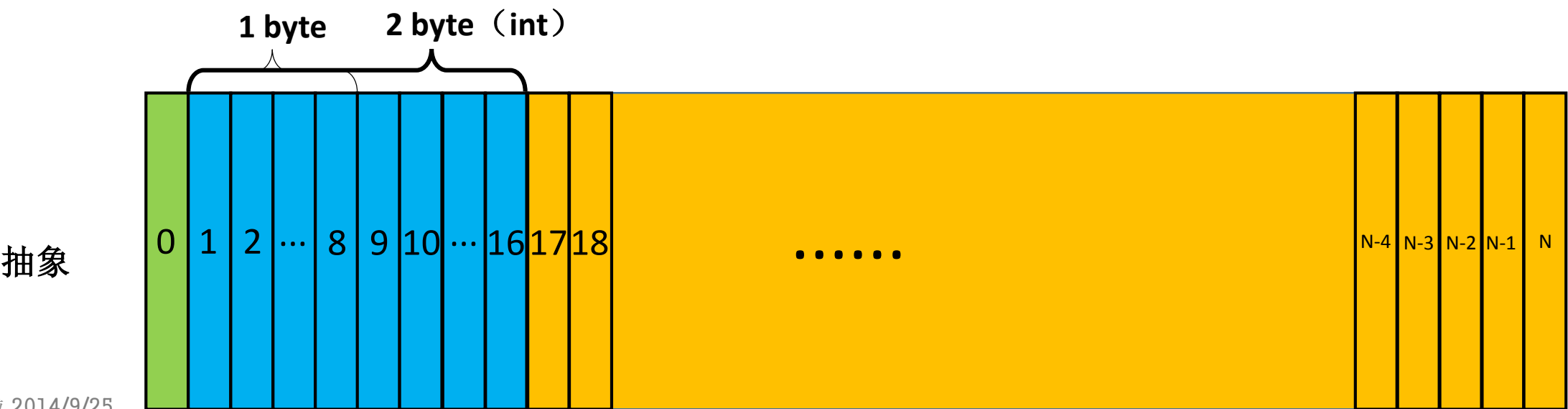
因此，在程序中使用int、long、float、double等等类型来描述尽可能多得的状态。



那程序中的变量是什么？——代表了所有状态的集合(还记得高数吗？)

char a; // **a**代表了 **2^8** 种状态的集合，在内存中占**8**个格子哦！
int b; // **b**代表了 **2^{16}** 种状态的集合，在内存中占**16**个格子哦！
long c; // **c**代表了 **2^{32}** 种状态的集合，在内存中占**32**个格子哦！
float d; // **d**代表了 **2^{32}** 种状态的集合，在内存中占**32**个格子哦！
double e; // **e**代表了 **2^{64}** 种状态的集合，在内存中占**64**个格子哦！

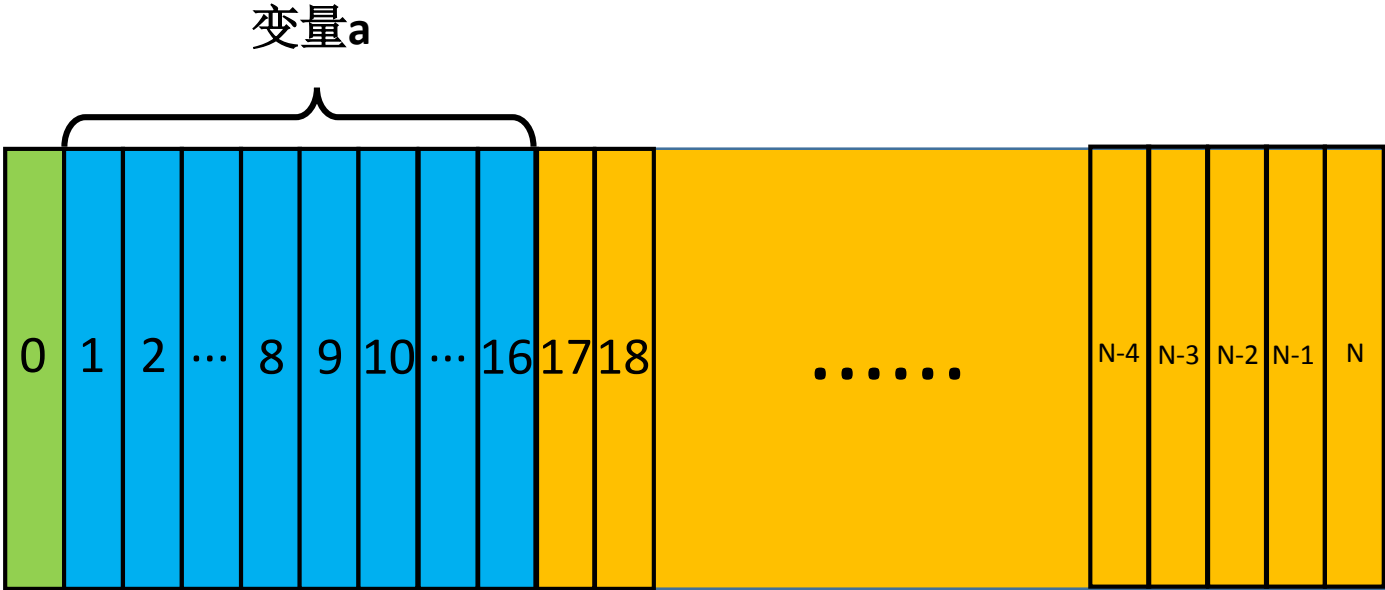
.....



程序中变量那么多，怎么知道每个变量在哪里？——内存地址(memory address)，即指针
只要知道变量的地址（指针），就能找到这个变量。



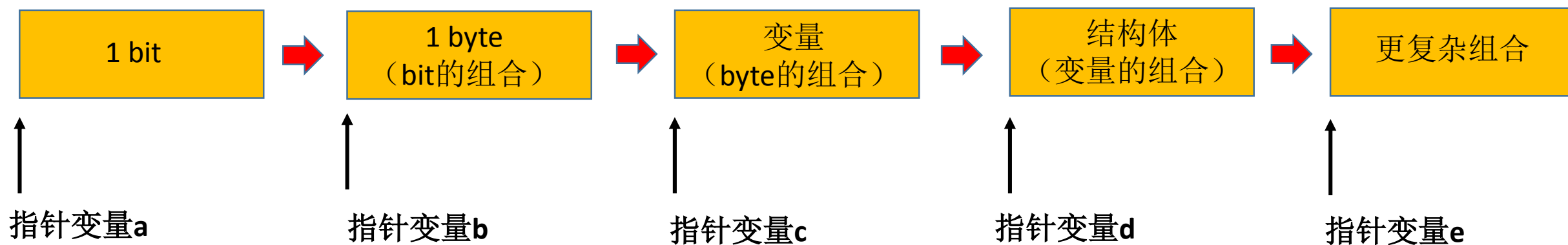
怎么找到你家住哪？——家庭地址



变量a内存地址 = 1 (内存块地址)

总结:

为了将人类的问题（任务）转变成电压信号，操作系统使用**不同长度**的内存格子来组合出**尽可能多**的电压状态，使用指针变量来记录这些组合的**位置**。



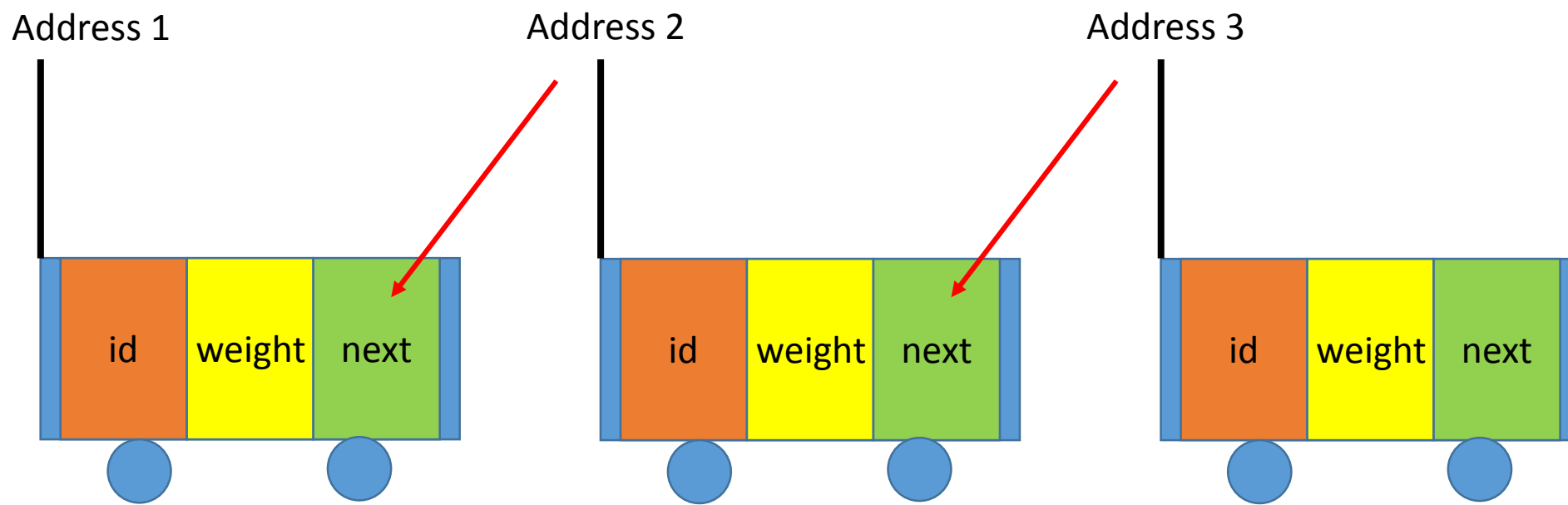
为了找到每个集合的位置，使用**指针变量**来存储内存地址。

```
int a=101;  
int* pointer=a;    //    pointer中存储了变量a的内存地址
```

C语言关键概念：**变量组合 + 指针**

1) 定义顺序链表结构：如同火车，一个变量链接一个变量，每个变量都相同。

```
3
4 typedef struct TrainBox{           // 车厢变量名（结构体）
5     int id;                        // 车厢编号
6     int weight;                   // 车厢载重
7     struct TrainBox* next;         // 下一节车厢（地址）
8 }TrainBox,*pTrainBox;              // 车厢变量名，TrainBox的指针变量名
9
```



2) 输入变量值：手动输入当前车厢载重大小

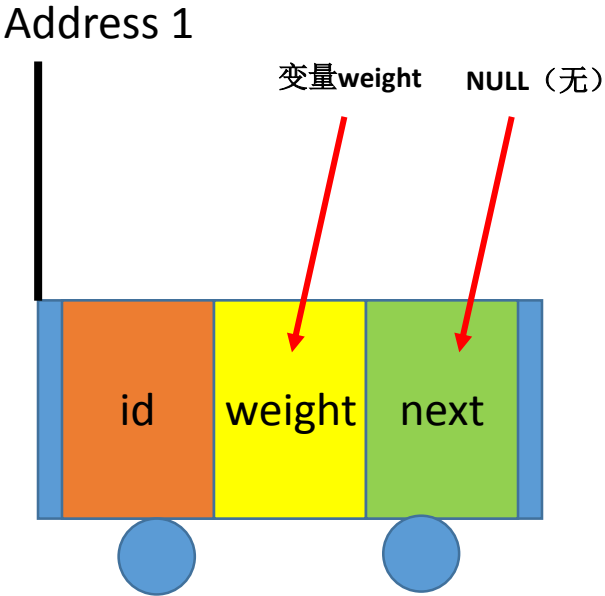
```
///input data
printf("input weight:\n");
scanf("%d",&weight);
```

提示信息
将输入值传递给变量weight

3) 创建新车厢变量：给该车厢记录编号和载重

```
///create a new trainbox
tmp=(pTrainBox)malloc(sizeof(TrainBox));
tmp->id=id;
tmp->weight=weight;
tmp->next=NULL;
```

// 分配内存格子
// 记录车厢的编号
// 将变量weight值记录到车厢变量中
// 当前车厢不链接其他车厢



4) 构建车厢链:

```
pTrainBox hLink=NULL; // 车厢链的头指针变量
int weight; // 临时变量:
int id=0; // tmp存储新车厢地址;
pTrainBox tmp,mark; // mark存储当前最后一节车厢地址
tmp=mark=NULL; //
while(1)
{
    ///input data
    printf("input weight:\n");
    scanf("%d",&weight); // 输入新车厢载重

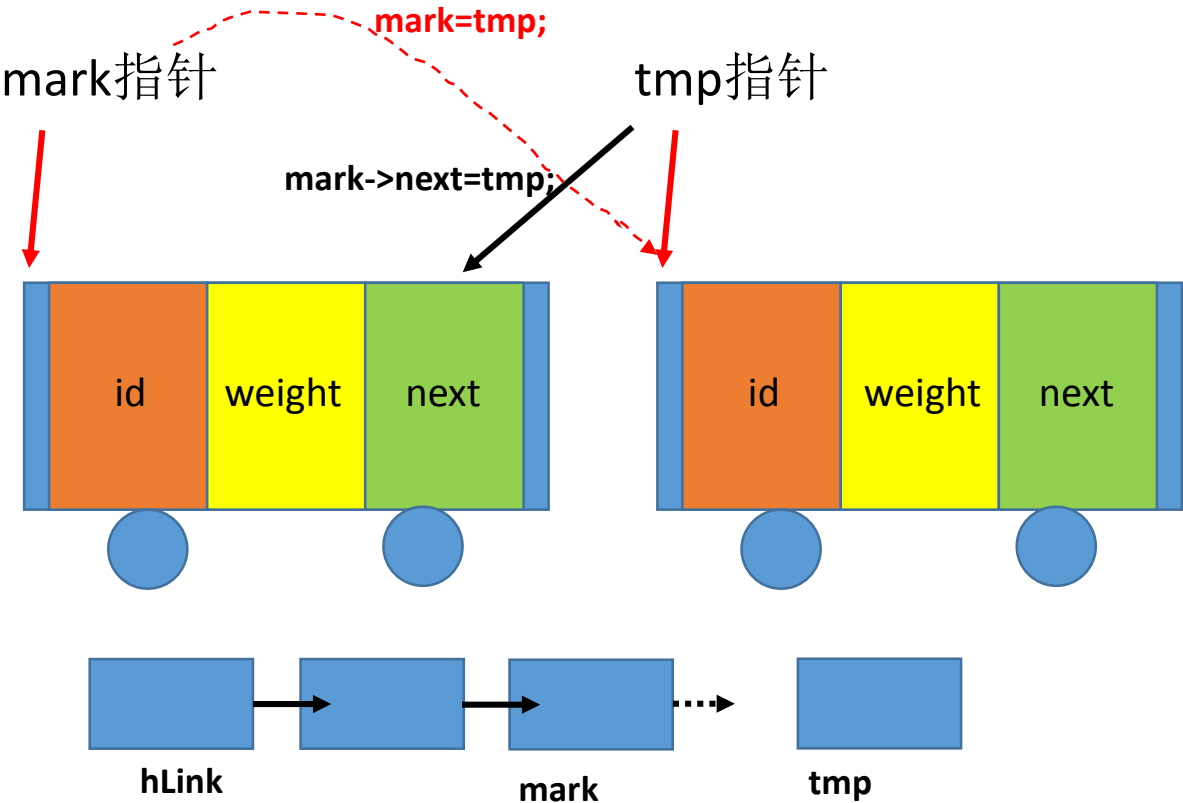
    ///stop inputting
    if(weight==-1)
    {
        printf("input over\n");
        break; // 停止链接车厢
    }
    id++; // 编号增加

    ///create a new trainbox
    tmp=(pTrainBox)malloc(sizeof(TrainBox));
    tmp->id=id;
    tmp->weight=weight;
    tmp->next=NULL; // 创建新车厢

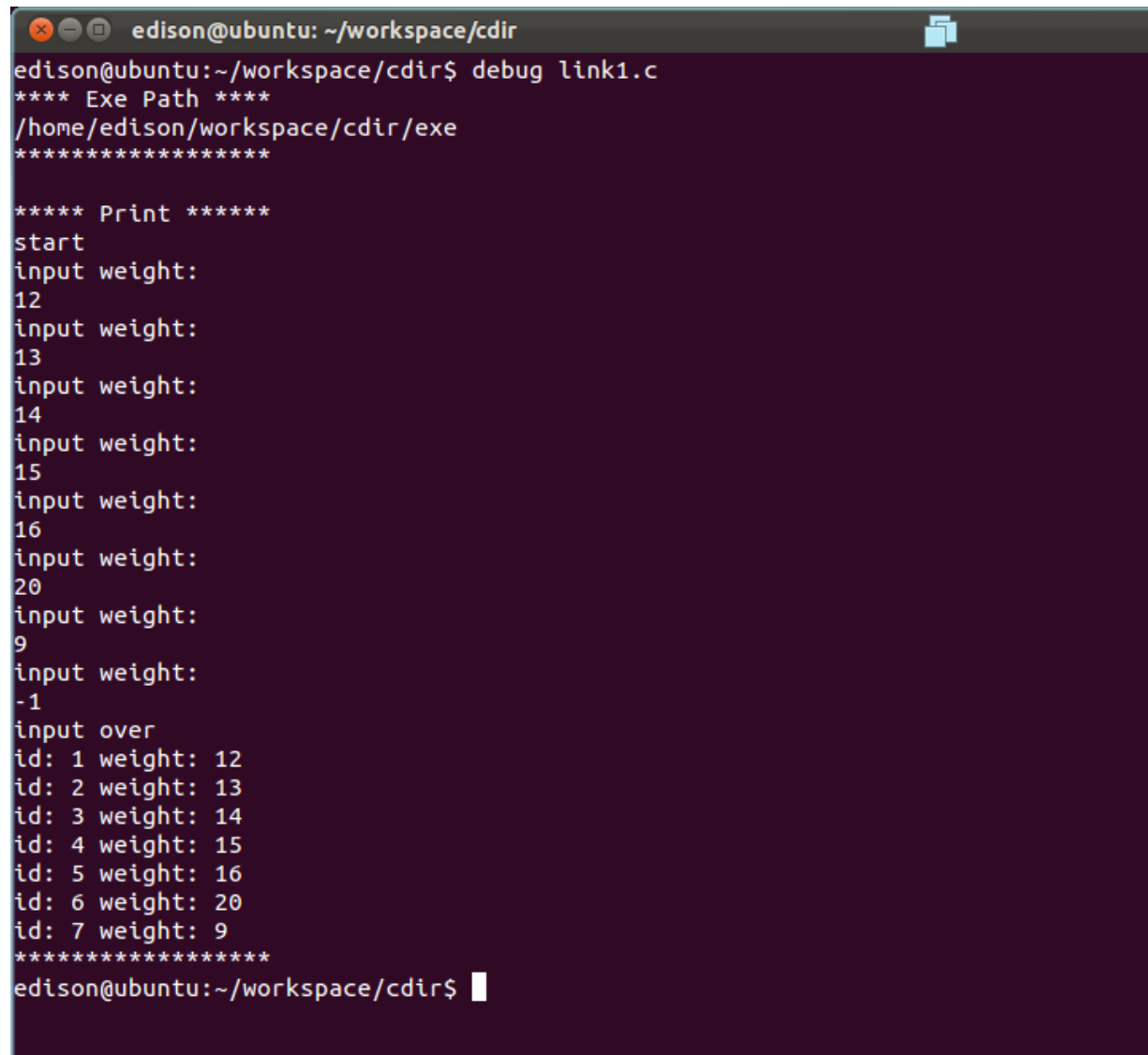
    ///start to link
    if(hLink==NULL)
    {
        hLink=tmp;
    }
    else
    {
        mark->next=tmp;
    }
    mark=tmp;
}
```

链接过程:

- a. 如果当前车厢链为空 (hLink==NULL)，则当前tmp存储的地址给hLink;
- b. 如果当前车厢链非空，当前最后一节车厢所记录的下一节车厢地址mark->next = tmp存储的地址；（火车新链接了一个车厢）
- c. 更新mark存储的地址值为新车厢地址；重复；



4) 执行程序，输入命令：**debug link1.c**



```
edison@ubuntu: ~/workspace/cdir
edison@ubuntu:~/workspace/cdir$ debug link1.c
**** Exe Path ****
/home/edison/workspace/cdir/exe
*****

**** Print ****
start
input weight:
12
input weight:
13
input weight:
14
input weight:
15
input weight:
16
input weight:
20
input weight:
9
input weight:
-1
input over
id: 1 weight: 12
id: 2 weight: 13
id: 3 weight: 14
id: 4 weight: 15
id: 5 weight: 16
id: 6 weight: 20
id: 7 weight: 9
*****
edison@ubuntu:~/workspace/cdir$
```