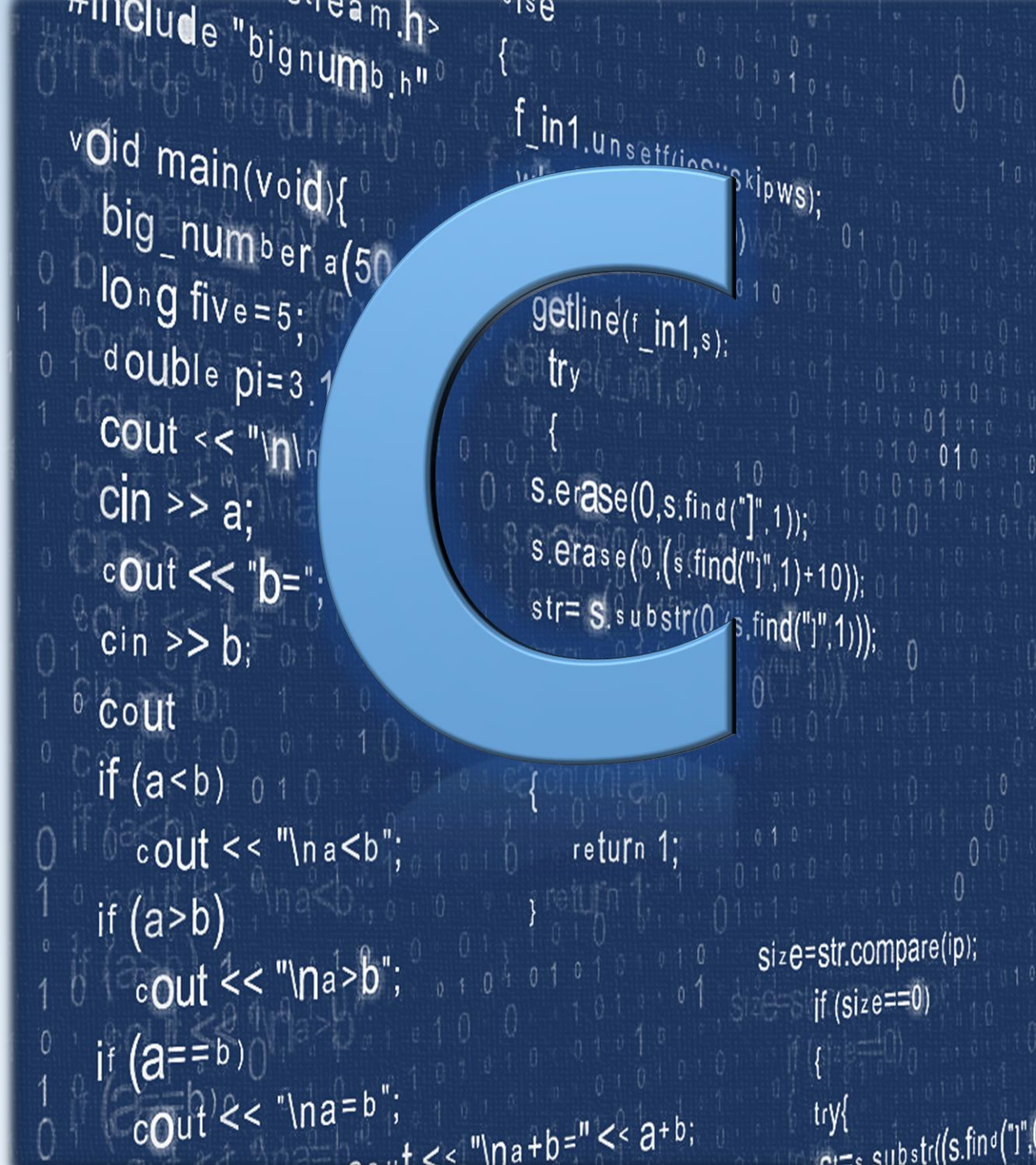
《 C语言程序设计》

丁盟
C语言课程组



本章授课内容

结构体与动态链表

结构体与动态链表

- 在前面使用结构体数组存储学生信息，但这样会存在以下问题：
  - 一旦定义好数组，则学生人数不能超过数组上限。
  - 如果学生数目远低于数组上限，则大量内存浪费。
  - 当频繁的插入、删除学生时需要移动大量元素。

- 能否有一个办法，保证系统资源的最合理运用？
  - 当我们需要添加一个学生时，手工分配内存。
  - 当我们需要删除一个学生时，手工删除该学生原来占有的内存。

动态数据结构-----**链表**

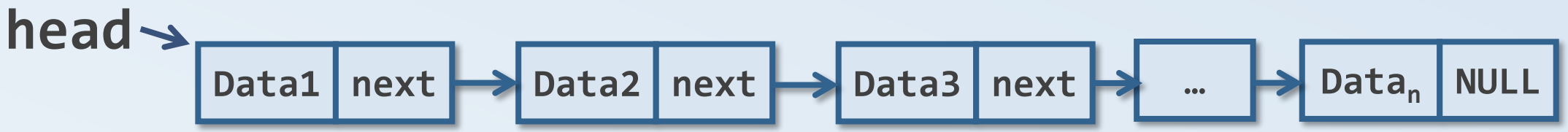
结构体与动态链表

- 动态链表的构成：
  - 动态链表有一个或者多个结点构成，每个节点都是一个结构体对象。
  - 每个结点有数据域和指针域(关系)

```
struct node
{
    int  data;
    struct node * next;
};
```

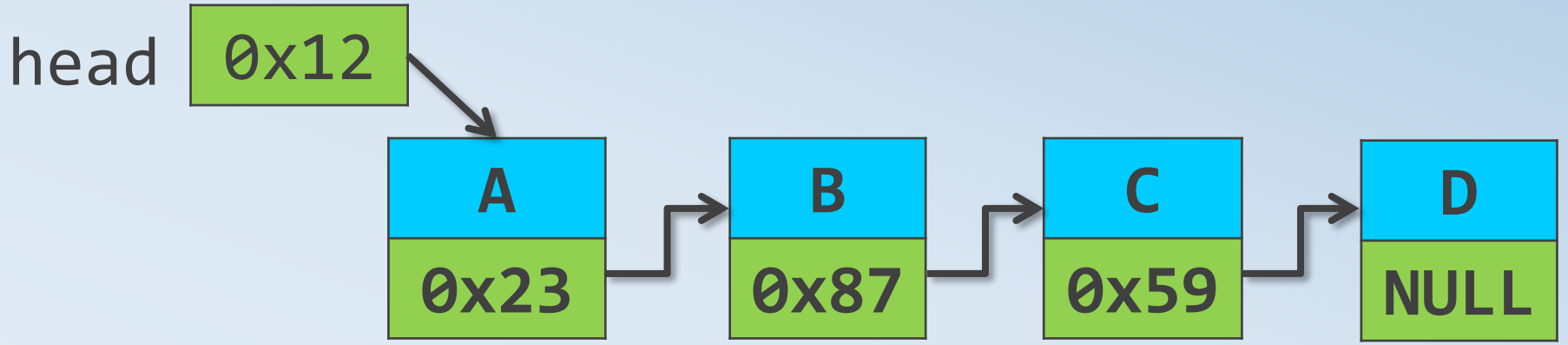
数据域：存储数据元素信息

指针域：存储直接后继的节点地址



不带头结点的动态链表

结构体与动态链表



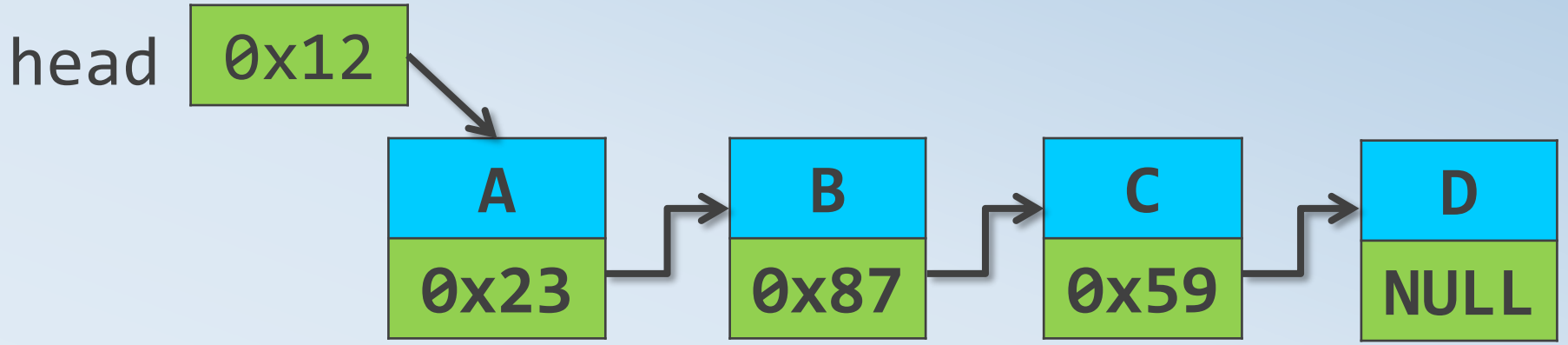
注意： ① 链表有一个**头指针**，存放第一个结点的地址

② 每一个**结点**由数据域和指针域构成

③ 最后一个结点指针域为NULL，称为“**表尾**”

④ 结点通常就是一个结构体变量

结构体与动态链表



链表特点：① 链表中各元素可以不是连续存放的.

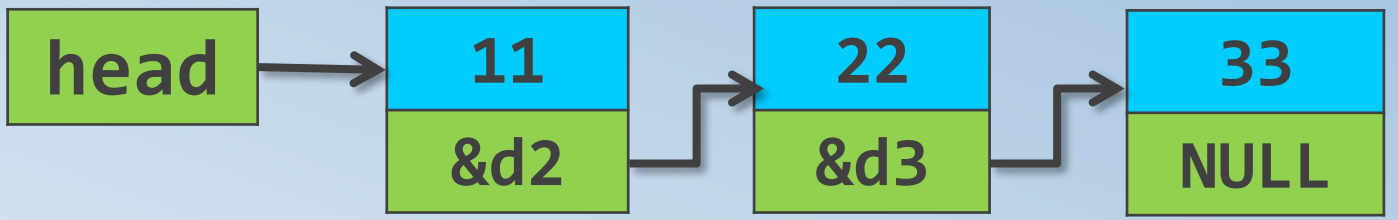
② 要找某结点必须先找到上个结点因此必须提供头指针.

③ 创建结点就是创建结构体和利用指针做成员.

结构体与动态链表

- 动态链表
  - 动态链表的结点是临时生成的。
  - 可以根据需要添加或者删除元素。
  - 程序员手工分配内存与释放内存（堆区）。
    - ✓**malloc**
    - ✓**calloc**
    - ✓**realloc**
    - ✓**free**

结构体与动态链表




```c
#include <stdio.h>

struct node
{
    int data;
    struct node * next;
};

int main(void)
{
    struct node d1, d2, d3;
    struct node * head, * p;

    d1.data = 11;
    d2.data = 22;
    d3.data = 33;

    head = &d1;
    d1.next = &d2;
    d2.next = &d3;
    d3.next = NULL;

    for(p = head; p!=NULL; p=p->next)
    {
        printf("%d\n", p->data);
    }
    return 0;
}
```

```
11
22
33
```

结构体与动态链表

- 链表的使用： 1、**创建空链表**
  1. 定义链表节点的数据结构。
  2. 创建一个节点指针head，使其指向NULL。

```
// 链表节点
typedef struct node
{
    int data;
    struct node * next;
} Node;

int main(void)
{
    // 创建head指针
    Node * head = NULL;

    return 0;
}
```

结构体与动态链表

- 链表的使用： 2、**链表末尾添加数据**

1. 利用malloc ( )函数向系统申请分配一个节点。
2. 将新节点的指针成员赋值为NULL。若是空表，将head指向新节点；若是非空表，循环找到链表尾节点，并使尾节点的next指针指向新节点。

```
void pushBackList(Node ** list, int data)
{
    Node * head = *list;
    Node * newNode = (Node *)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    if(*list == NULL)
        *list = newNode;
    else
    {
        while(head ->next != NULL)
            head = head->next;
        head->next = newNode;
    }
}
```

结构体与动态链表

- 链表的使用： 3、**链表数据打印**

1. 节点指针head，使其指向链表头结点。
2. 判断head是否为NULL，如果为NULL则结束，不为空打印head指向节点的数据。
3. 使head指向当前节点的下一个节点(head >next)。
4. 执行2。

```
void printList(Node * head)
{
    Node * temp = head;
    for(; temp != NULL; temp=temp->next)
        printf("%d ", temp->data);
    printf("\n");
}
```

结构体与动态链表

- 链表的使用： 4、**统计链表链表长度**（元素个数）

1. 节点指针head，使其指向链表头结点。
2. 判断head是否为NULL，如果为NULL则结束。
3. 不为空计数器+1，使head指向当前节点的下一个节点(head >next)。
4. 执行2。

```
int sizeList(Node * head)
{
    Node * temp = head;
    int len;
    for(len=0; temp!=NULL; len++)
        temp=temp->next;
    return len;
}
```

结构体与动态链表

- 链表的使用： 5、**清空链表**（malloc得到的内存需手动释放）

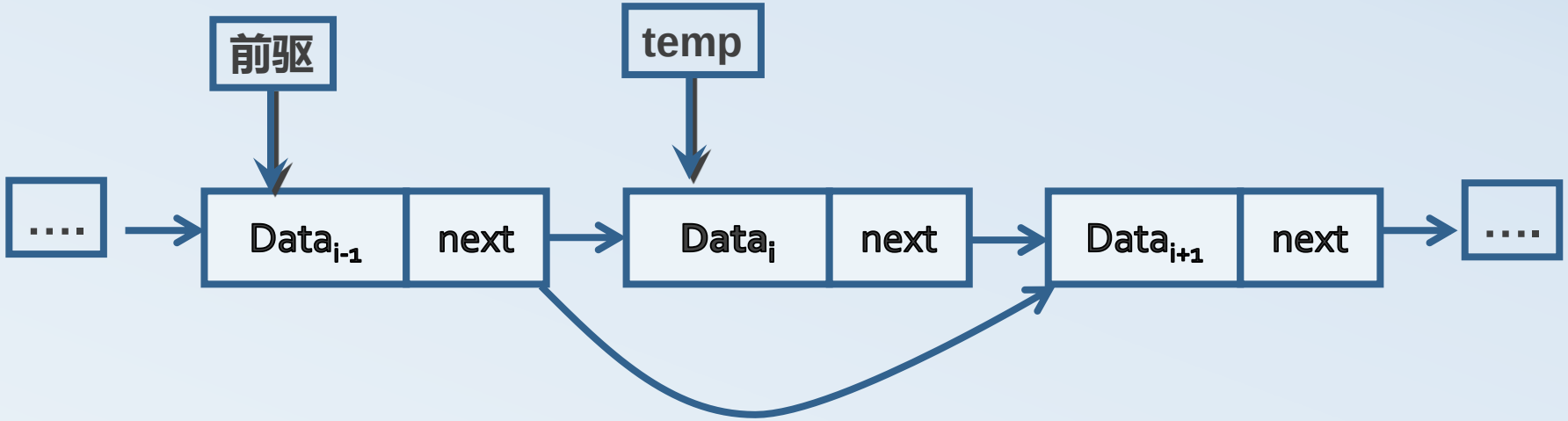
1. 节点指针head，使其指向链表头结点。
2. 判断head是否为NULL，如果为NULL则结束，并使head指向NULL。
3. 不为空计数器使用节点指针temp赋值为head，使temp指向当前节点的下一个节点(temp->next)，释放temp指针（free(temp)）。
4. 执行2。

```
void freeList(Node ** list)
{
    Node * head = *list;
    Node * temp = NULL;
    while(head != NULL)
    {
        temp = head;
        head = head->next;
        free(temp);
    }
    *list = NULL;
}
```

结构体与动态链表

- 链表的使用： 6、**删除链表节点**（数据匹配的节点）

1. 首先找到要删除节点的前驱节点 if(前驱->next->data满足条件)
2. 用temp指向要删除的节点 temp = 前驱->next;
3. 前驱->next = temp->next;
4. free(temp);



结构体与动态链表

- 链表的使用： 6、**删除链表节点**（数据匹配的节点）

```
void deleteList(Node ** list,
int data)
{
    Node * head = *list;
    Node * temp;
    while(head->next!=NULL)
    {
        if(head->next->data !=
data)
        {
            head=head->next;
            continue;
        }
        temp = head->next;
        if(head->next->next == NULL)
            head->next = NULL;
        else
            head->next = temp->next;
        free(temp);
    }
    head = *list;
    if(head->data == data)
    {
        temp = head;
        *list = head->next;
        head = head->next;
        free(temp);
    }
}
```

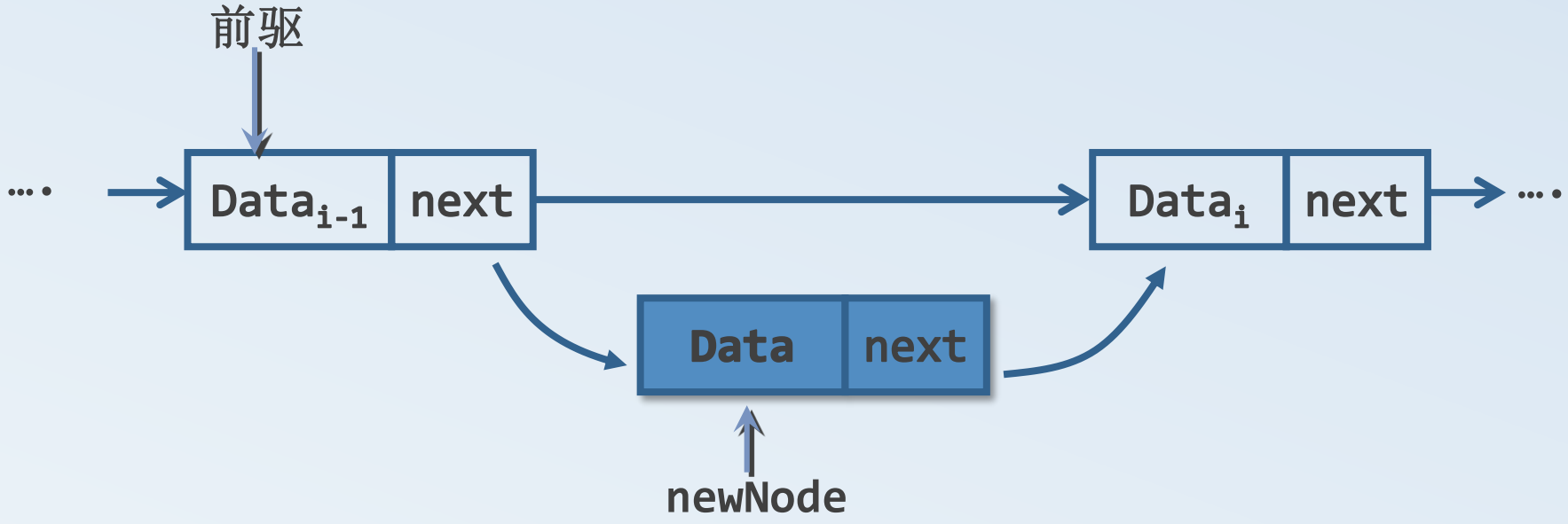
如果尾结点需删除

如果头结点需删除

结构体与动态链表

- 链表的使用： 7、**链表插入节点**（指定位置插入节点）

1. 首先查找第i-1个节点（前驱节点） if（前驱满足条件）
2. 给新节点分配空间newNode =(Node *)malloc（sizeof(Node)）;给newNode数据域赋值。
3. newNode ->next = 前驱->next;
4. 前驱->next = newNode;



结构体与动态链表

- 链表的使用： 7、**链表插入节点**（指定位置插入节点）

链表头部插入节点

```
int insertList(Node ** list, int
index, int data)
{
    int n;
    int size = sizeList(*list);
    Node * head = *list;
    Node * newNode, * temp;

    if(index<0 || index>size)
        return 0;
```

创建新节点

```
    newNode = (Node
*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
```

```
    if(index == 0)
    {
        newNode->next = head;
        *list = newNode;
        return 1;
    }

    for(n=1; n<index; n++)
        head = head->next;

    if(index != size)
        newNode->next = head->next;
    head->next = newNode;
    return 1;
}
```

链表尾部next不需指定

Thank You !