数据结构

结构体指针

```
struct Hero {
    string name;
    int a, b, rp;
};
Hero ironman = {"Ironman", 80, 60, 30};
```

修改超级英雄信息，攻击力、防御力、人品都改为90

```
ironman.a=90;
ironman.b=90;
ironman.rp=90;
```

直接修改结构体变量

```
Hero *p;
p = &ironman;
(*p).a=90;
(*p).b=90;
(*p).rp=90;
```

使用结构体指针

结构体指针

```
struct Hero {
    string name;
    int a, b, rp;
};
Hero ironman = {"Ironman", 80, 60, 30};
```

修改超级英雄信息，攻击力、防御力、人品都改为90

```
ironman.a=90;
ironman.b=90;
ironman.rp=90;
```

直接修改结构体变量

```
Hero *p;
p = &ironman;
p->a=90;
p->b=90;
p->rp=90;
```

使用结构体指针

结构体指针：2种方法

```
Hero *p;
p = &ironman;
(*p).a=90;
(*p).b=90;
(*p).rp=90;
```

```
Hero *p;
p = &ironman;
p->a=90;
p->b=90;
p->rp=90;
```

说明	->是所有运算符中优先级最高的 ->用法更常用

动态内存分配

```
1  #include<iostream>
2  using namespace std;
3  int main(){
4      int *p;
5      p = new int;
6      *p= 99;
7      cout << *p << endl;
8      return 0;
9  }
```

用运算符**new**
动态分配内存

定义指针p指向整数类型变量

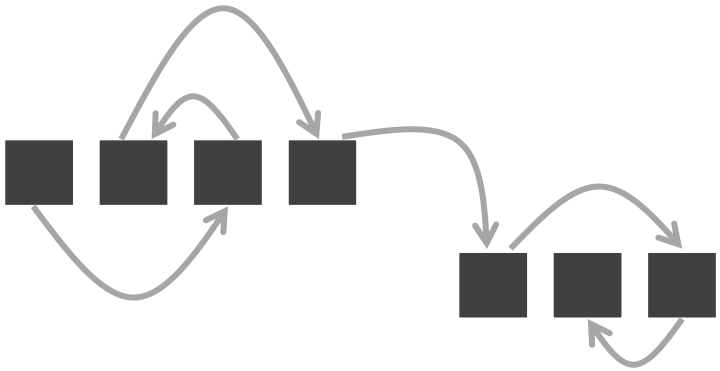
动态分配一个新整数
内存地址赋值给p

动态内存分配

请理解每一行的含义

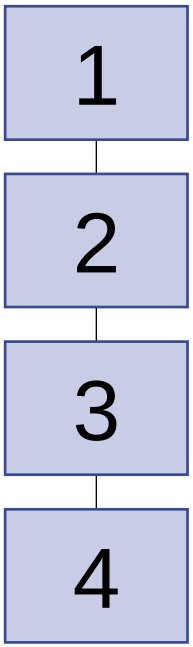
```
13  int main() {
14      Hero *p;
15      p = new Hero;
16      p->name="Spiderman";
17      p->a=60;
18      p->b=70;
19      p->rp=80;
20      print(p);
21      return 0;
22  }
```

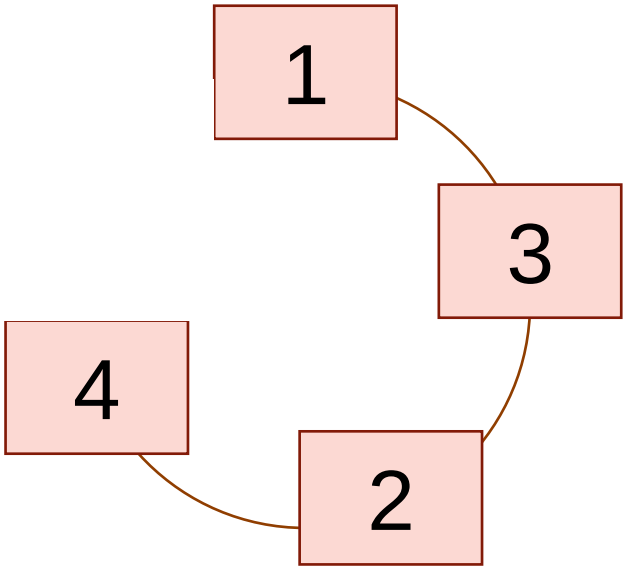
单链表

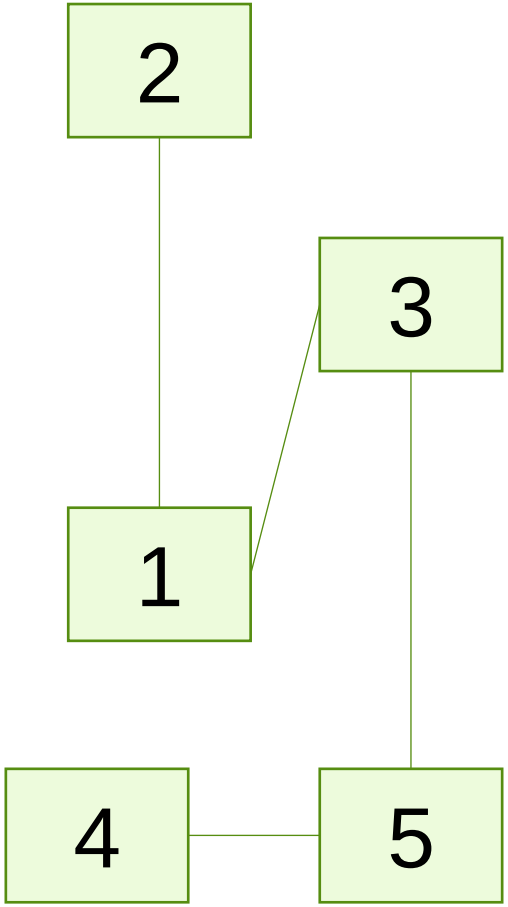


线性表

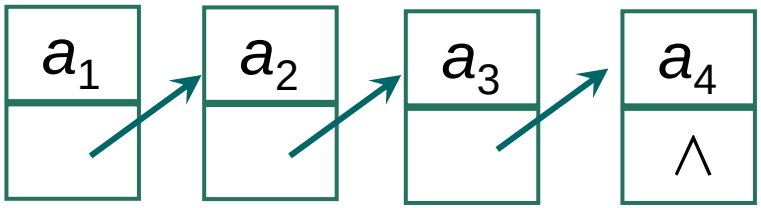
判断下列结构是否是线性表

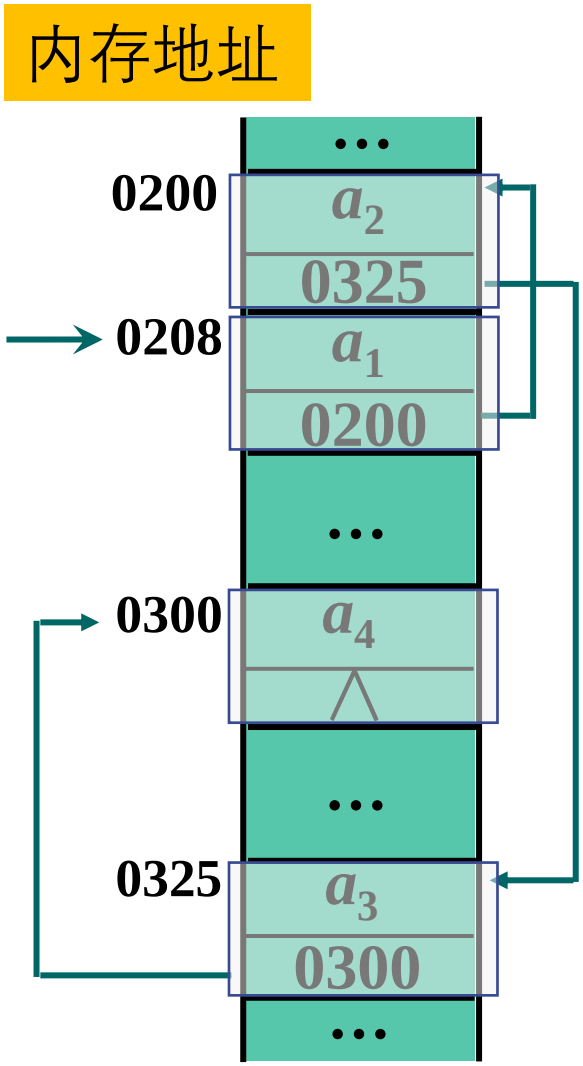






线性表的链式存储





(a_1, a_2 ,a_3, a_4)的存储示意图

链表存储要求

线性表

- 顺序表
  - 静态存储分配
  - 事先确定容量
- 链表
  - 动态存储分配
  - 运行时分配空间

单链表对内存要求

连续/不连续/零散分布均可以

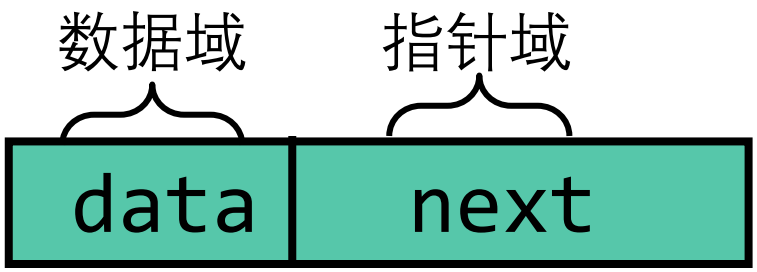
没有要求

单链表存储结构及实现

单链表里每个结点
都包含一个指针域next

单链表结点

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



存储 数据	存储后继结点 的地址

单链表存储结构及实现

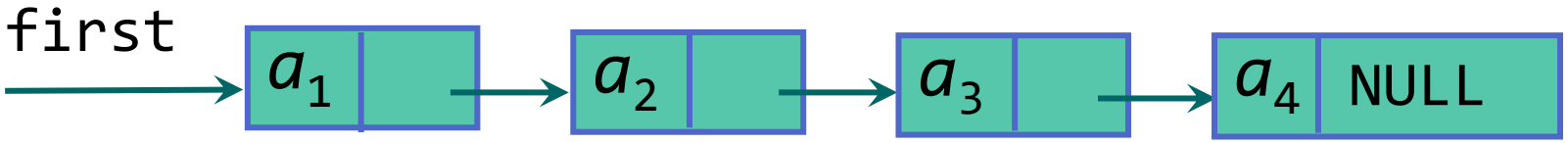
头指针first：储存第一个结点的地址

尾标志：终端结点的指针域为空，即NULL

空表

first=NULL

非空表



单链表的遍历

```
21      s = first;
22      while(s != NULL) {
23          cout << s->data << " ";
24          s = s->next;
25      }
```

遍历指针s赋值为头指针

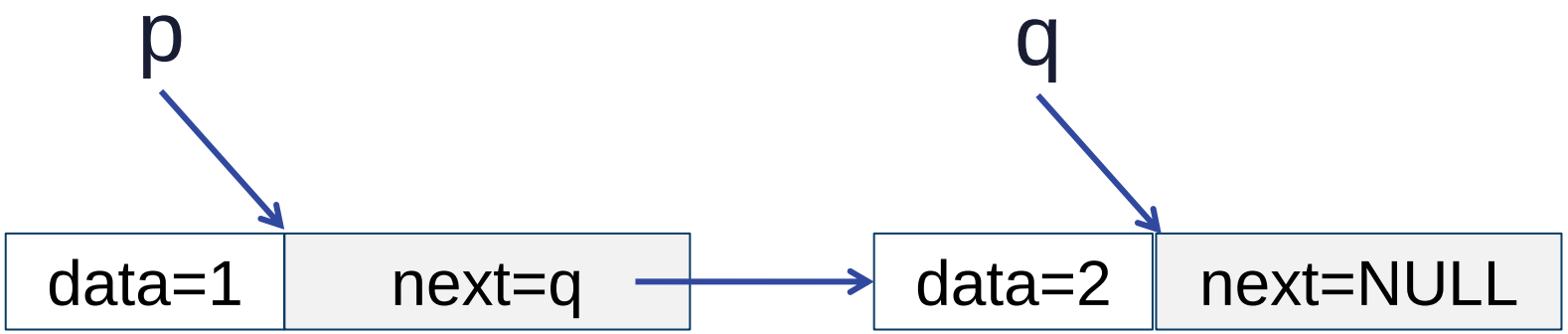
指针尚未到达链表末尾

移动指针到下一个结点

走一步看一步

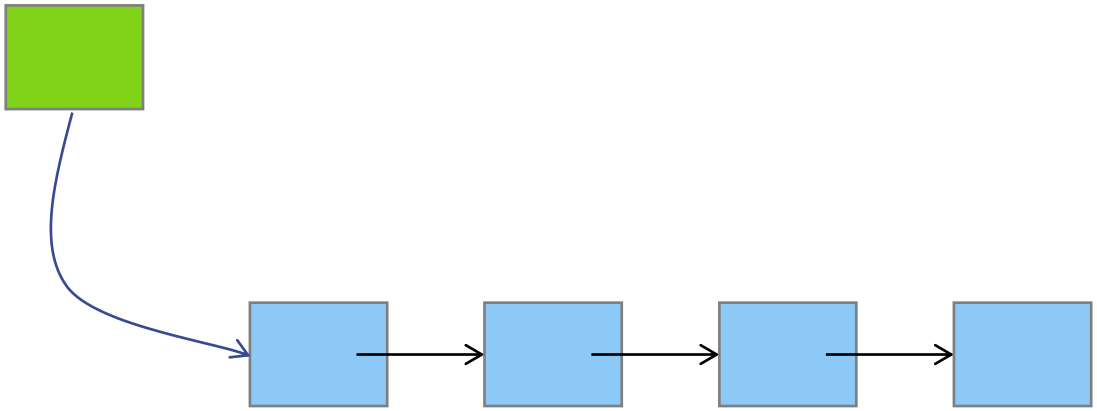
摸着石头过河

单链表连接2个结点

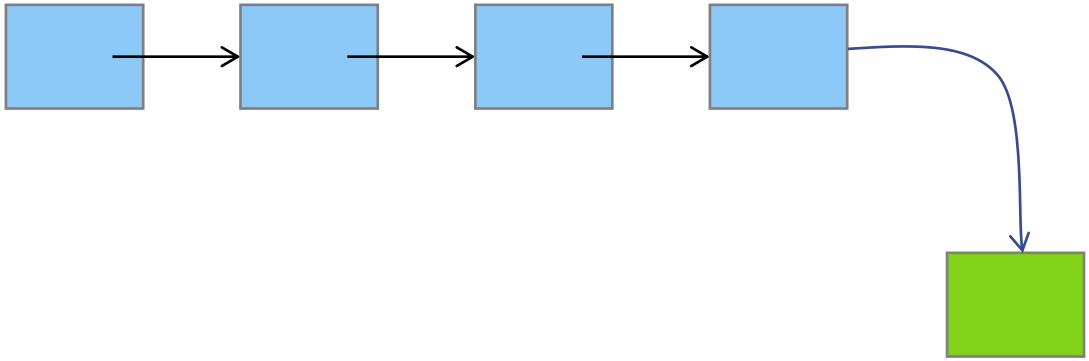



```
 8    Node *p, *q, *s;
 9    p = new Node;
10    q = new Node;
11    p->data = 1;
12    q->data = 2;
13    p->next = q;
14    q->next = NULL;
```

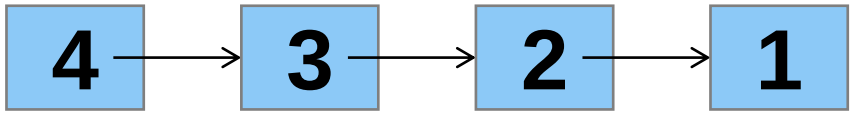
头插法



尾插法



用头插法建立链表



头插法：将待插入结点插在开头结点前

头插法构建单链表

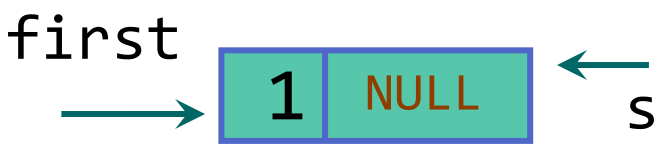
初始化

空链表
头指针为空

代码实现

```
first=NULL;
```

插入第一个结点

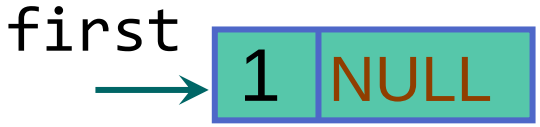


代码实现

```
s=new Node;
s->data=1;
s->next=NULL;
first=s;
```

头插法构建单链表

依次插入每一个结点

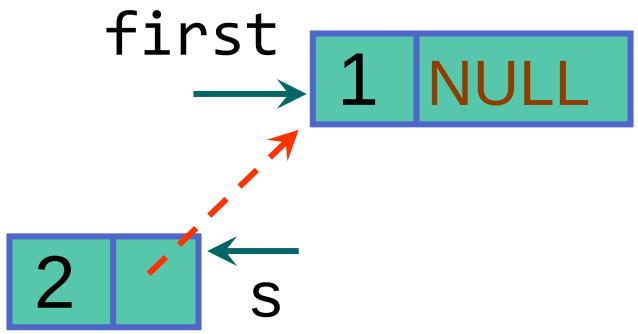


代码实现

```
s=new Node;
s->data=2;
s->next=first;
first=s;
```

头插法构建单链表

依次插入每一个结点

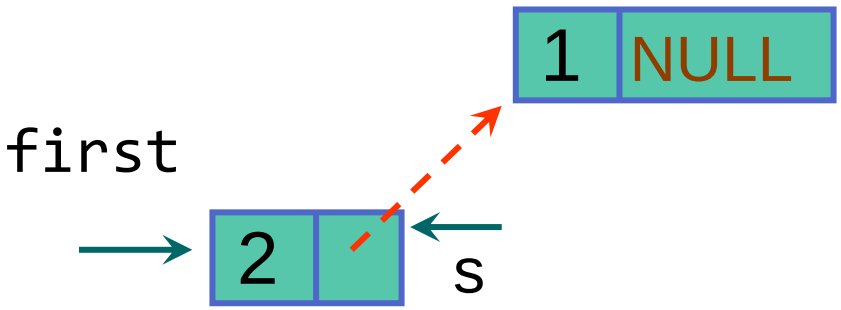


代码实现

```
s=new Node;
s->data=2;
s->next=first;
first=s;
```

头插法构建单链表

依次插入每一个结点



代码实现

```
s=new Node;
s->data=2;
s->next=first;
first=s;
```

头插法构建单链表

运行程序 头插法建立单链表

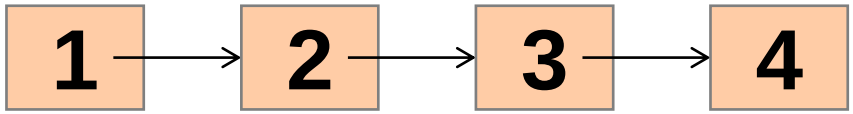
输入链表长度和数据

输入:
5
35 12 24 33 42

```
 9      Node *first, *s;
10      first = NULL;
11      cin >> n;
12      for(int i=0; i<n; i++) {
13          cin >> x;
14          s = new Node;
15          s->data = x;
16          s->next = first;
17          first = s;
18      }
```

思考：链表中元素的顺序

用尾插法建立链表



尾插法：将待插入结点插在终端结点的后面

尾插法构建单链表

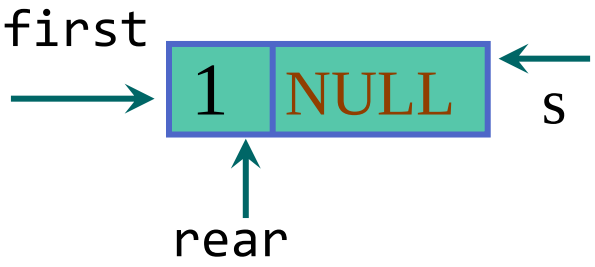
初始化

空链表
头指针为空
尾指针也为空

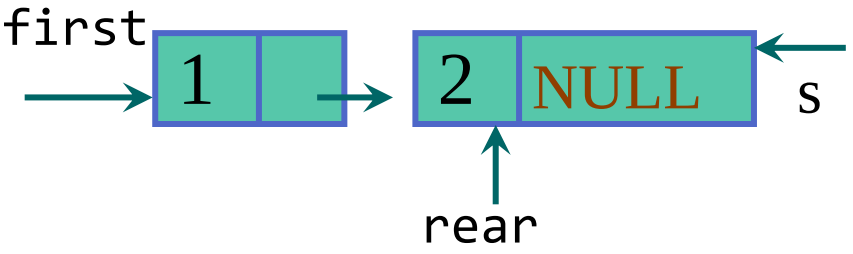
```
first=NULL
```

```
rear=NULL
```

插入第一个结点



再插入其他结点



思考每一步代码
如何实现

尾插法构建单链表

运行程序 头插法建立单链表

输入链表长度和数据

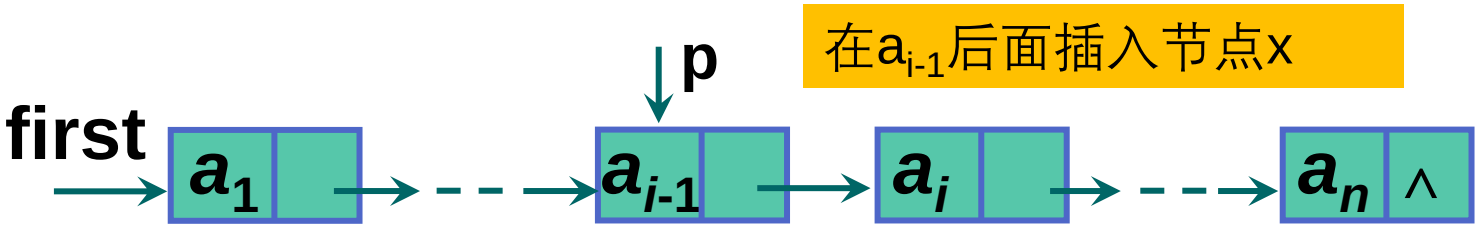
```
 9      Node *first, *rear, *s;
10      cin >> n >> x;
11      s = new Node;
12      s->data = x;
13      s->next = NULL;
14      rear = first = s;
15      for(i=1; i<n; i++) {
16          cin >> x;
17          s = new Node;
18          s->data = x;
19          s->next = NULL;
20          rear->next = s;
21          rear = s;
22      }
```

输入
5
35 12 24 33 42

思考第一个结点的特殊性

之后节点的插入不再改变first只改变rear

定位后插入



操作步骤

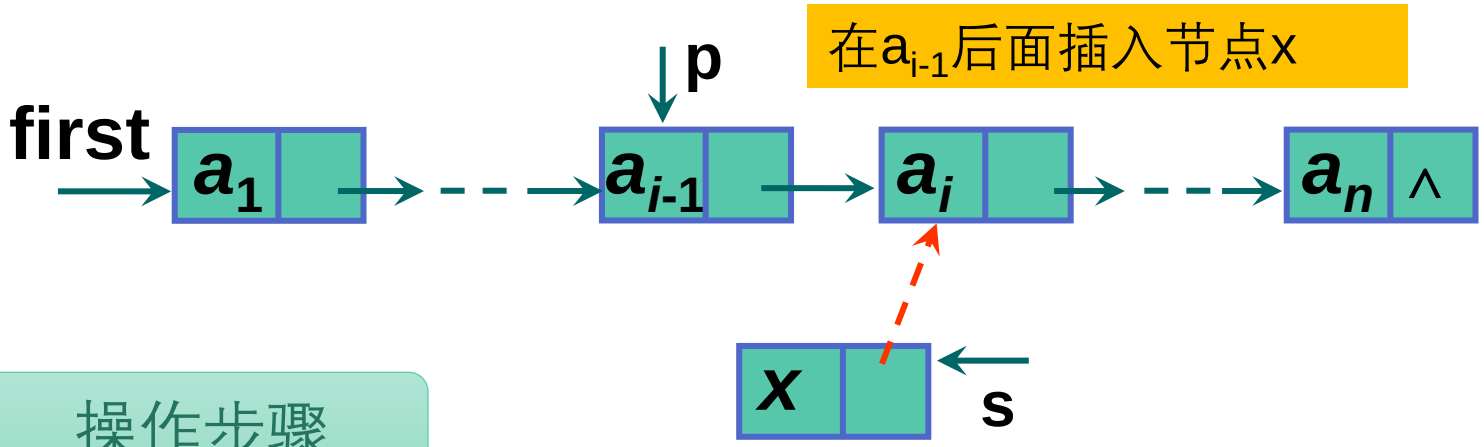
1. 查找第i-1个结点，使指针p指向该结点
2. 创建一个元素值为x的新结点s
3. 将新结点s插入到结点p之后

代码实现

```
s=new Node;
s->data=x;
s->next=p->next;
p->next=s;
```

这两行能否互换?

定位后插入



操作步骤

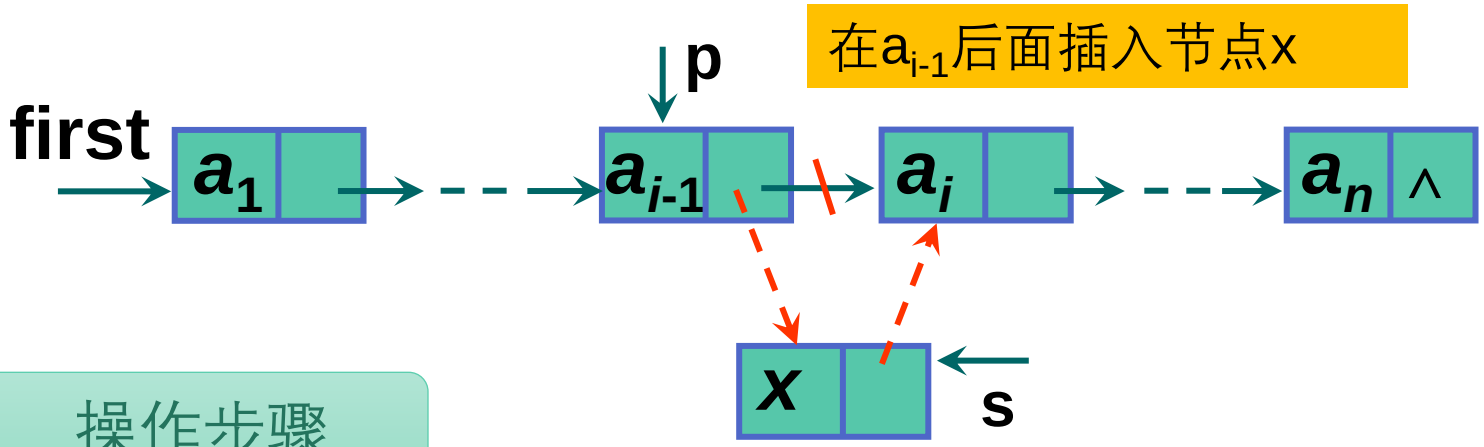
1. 查找第i-1个结点，使指针p指向该结点
2. 创建一个元素值为x的新结点s
3. 将新结点s插入到结点p之后

代码实现

```
s=new Node;
s->data=x;
s->next=p->next;
p->next=s;
```

这两行能否互换?

定位后插入



操作步骤

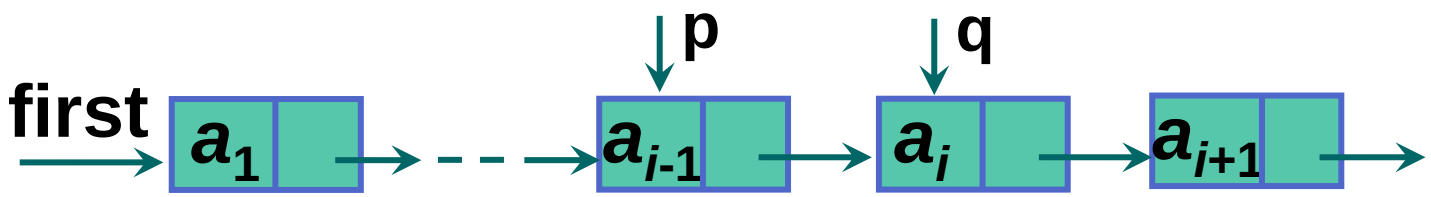
1. 查找第i-1个结点，使指针p指向该结点
2. 创建一个元素值为x的新结点s
3. 将新结点s插入到结点p之后

代码实现

```
s=new Node;
s->data=x;
s->next=p->next;
p->next=s;
```

这两行能否互换?

定位后删除



删除a_{i-1}后面的节点a_i

操作步骤

1. 查找第i-1个结点并使指针p指向该结点
2. q为p的下一个节点，即要删除的节点
3. 将结点q从链表中删除

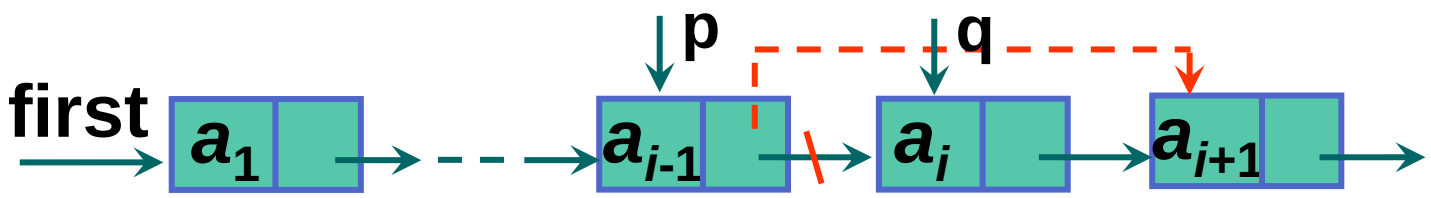
代码实现

```
q=p->next;
p->next=q->next;
```

另类代码

```
p->next=p->next->next;
```

定位后删除



删除a_{i-1}后面的节点a_i

操作步骤

1. 查找第i-1个结点并使指针p指向该结点
2. q为p的下一个节点，即要删除的节点
3. 将结点q从链表中删除

代码实现

```
q=p->next;
p->next=q->next;
```

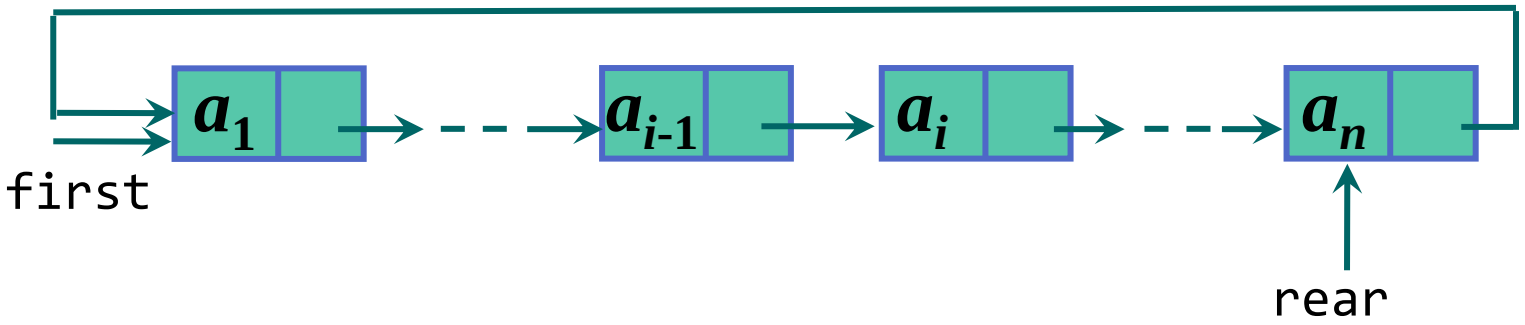
另类代码

```
p->next=p->next->next;
```

循环链表

循环链表

带尾指针的循环链表，可以方便的访问表首或表尾。实际中多采用尾指针指示的循环链表。



表首：rear->next 也就是first

表尾：
rear

太戈编程

	533这不是编程题
	1067单链表选择题
拓展题	1046 分发蛋糕
	单链表插入操作
拓展题	882 单链表的操作
	参考单链表插入删除操作