

C++程序设计(拾肆)

徐东/计算数学

内容

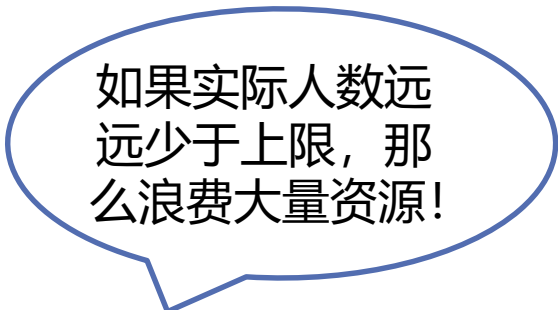
- 创建自定义的新数据类型
 - 结构体类型
- 应用：链表

任务肆拾陆

- 存储一些学生的个人信息
 - 姓名
- 问题
 - 人数不确定

任务肆拾陆：静态数组

```
int main(){  
    const int NUMBER = 100;  
  
    string students[NUMBER] = {" "};  
  
    for(int i=0;i<NUMBER;++i){  
        cin >> students[i] ;  
    }  
  
    return 0;  
}
```

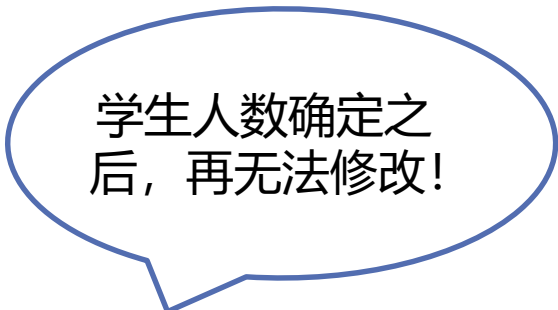


如果实际人数远
远少于上限，那
么浪费大量资源！

- 设置人数的上限

任务肆拾陆：动态数组

```
int main(){  
    int number = 0;  
    cout << "请输入学生人数: ";  
    cin >> number;  
  
    string *students = new string[number];  
  
    for(int i = 0; i < number; ++i){  
        cin >> students[i] ;  
    }  
  
    return 0;  
}
```



学生人数确定之后，再无法修改！

- 程序运行时，确定学生人数。

任务肆拾陆

```
const int NUMBER = 100;

string students[NUMBER] = {" "};

for(int i=0;i<NUMBER;++i){
    cin >> students[i] ;
}
```

```
int number = 0;
cout << "请输入学生人数: ";
cin >> number;

string *students = new string[number];

for(int i = 0; i < number; ++i){
    cin >> students[i] ;
}
```

- 数组的最大缺陷
 - 数组大小固定（无法扩容）

任务肆拾陆：扩容

```
int main(){
    const int NUMBER = 100;
    string students[NUMBER] = {" "};
    for(int i=0;i<NUMBER;++i){
        cin >> students[i] ;
    }
    //人数超过上限，新增了24人。
    const int NEWNUMBER = NUMBER + 24;
    string newStudents[NEWNUMBER] = {" "};

    for(int i = 0; i < NUMBER; ++i){
        newStudents[i] = students[i];           // 复制已有数据
    }
    for(int i = NUMBER; i < NEWNUMBER; ++i){
        cin >> newStudents[i];                 // 输入新增数据
    }
    return 0;
}
```


任务肆拾陆

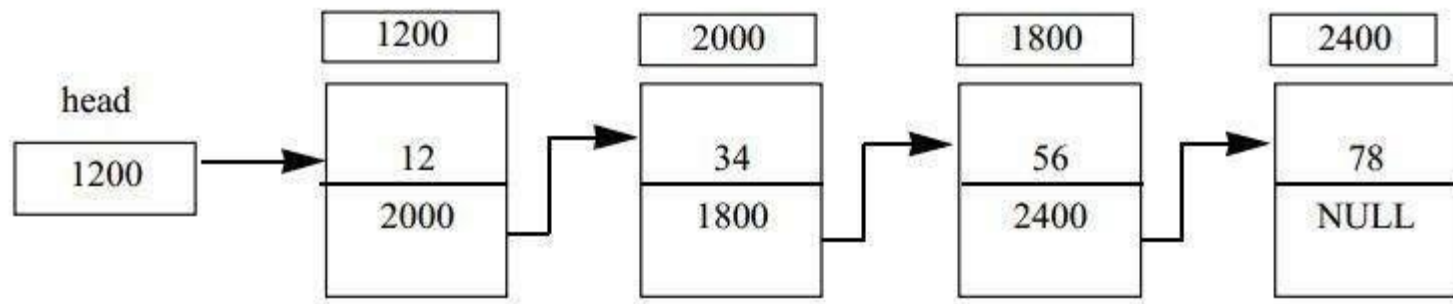
- 存储学生的个人信息
 - 姓名
- 学生的人数不确定
- 管理长度或数量不确定的数据
- 链表
 - 相对于数组，链表处理这类数据时比较节省内存。

链表

- 常用于在程序中存储一组不定长的线性数据。
- 具有这样的特点的数据可以用链表来保存：
 1. 数据是逐渐增加的
 2. 数据的数量不确定
 3. 不需要按照索引值对数据进行访问
 4. 添加、删除数据的动作的时间复杂度都是 $O(1)$

链表

- 链表由一系列结点（链表中每一个元素称为结点）组成。
- 结点可以在运行时动态生成。
- 每个结点包括两个部分
 - 存储数据元素的数据域
 - 存储下一个结点地址的指针域。



链表

- **优点**

- **无需预先知道数据的数量**
- **充分利用计算机内存空间，实现灵活的内存动态管理。**

- **缺点**

- **失去了数组随机读取的能力**
- **由于增加了结点的指针域，空间开销比较大。**

链表

- 每个结点包括两个部分
 - 存储数据元素的数据域
 - 存储下一个结点地址的指针域
- 创建新的数据类型
 - struct

创建新的数据类型

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

创建新的数据类型

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

- struct
 - 关键字

创建新的数据类型

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

- **新类型的名称**
 - 遵循 C++ 标识符命名规范
 - 作用类似于 `int`, `double`, `string` 等

创建新的数据类型

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

- {}

- 封装新类型的成员（和外界其他代码进行隔离）

创建新的数据类型

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

- **成员变量的声明语句**
 - **组成新类型（变量）的各部分的名称（成员域）**
 - **新类型的变量是一个组合体（由若干变量组成）**

创建新的数据类型

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

- ;

- **标志新类型的定义到此结束**

创建新的数据类型

```
struct Person{  
    string name;  
    double weight;  
    double height;  
};
```

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

- **新数据类型** Person
 - **字符串成员变量** name
 - **双精度成员变量** weight
 - **双精度成员变量** height

使用新的数据类型

- 声明变量

- 数据类型 变量名称 { 成员变量的初始值 };

```
Person tom{"tom", 78.0, 1.77};  
cout << "姓名 : " << tom.name ;
```

- 使用变量

- 变量.成员名称

- 点号 (.) 成员访问符

使用新的数据类型

```
struct Person{
    string name;
    double weight;
    double height;
};
int main(){
    Person tom{"tom",78.0, 1.77};
    cout << "姓名 : " << tom.name << endl;
    cout << "体重 : " << tom.weight << endl;
    cout << "身高 : " << tom.height << endl;
    cout << "bmi : " << tom.weight/(tom.height*tom.height) << endl;
    cout << "*****" << endl;
    tom.name = "jerry";
    tom.height += 0.05;
    cout << "请输入" << tom.name << "的体重 :";
    cin >> tom.weight;
    cout << "*****" << endl;
    cout << "姓名 : " << tom.name << endl;
    cout << "体重 : " << tom.weight << endl;
    cout << "身高 : " << tom.height << endl;
    cout << "bmi : " << tom.weight/(tom.height*tom.height) << endl;
    return 0;
}
```

链表：创建结点的数据类型

- 每个结点包括两个部分
 - 存储数据元素的数据域
 - 存储下一个结点地址的指针域
- 结点

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

链表：创建结点的数据类型

- 每个结点包括两个部分
 - 存储数据元素的数据域
 - 存储下一个结点地址的指针域

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

- 结点

新数据类型

链表：创建结点的数据类型

- 每个结点包括两个部分
 - 存储数据元素的数据域
 - 存储下一个结点地址的指针域

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

- 结点 新数据类型 Student

链表：创建结点的数据类型

- 每个结点包括两个部分
 - 存储数据元素的数据域
 - 存储下一个结点地址的指针域

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

- 结点 新数据类型 Student
 - 存储的数据

链表：创建结点的数据类型

- 每个结点包括两个部分
 - 存储数据元素的数据域
 - 存储下一个结点地址的指针域

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

- 结点
 - 存储的数据
- | 新数据类型 | Student |
|-------|---------|
| 姓名 | |

链表：创建结点的数据类型

- 每个结点包括两个部分
 - 存储数据元素的数据域
 - 存储下一个结点地址的指针域

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

- | | | |
|---------|-------|---------|
| • 结点 | 新数据类型 | Student |
| • 存储的数据 | 姓名 | string |

链表：创建结点的数据类型

- 每个结点包括两个部分
 - 存储数据元素的数据域
 - 存储下一个结点地址的指针域

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

- | | | |
|--------------|-------|---------|
| • 结点 | 新数据类型 | Student |
| • 存储的数据 | 姓名 | string |
| • 存储下一个结点的地址 | | |

链表：创建结点的数据类型

- 每个结点包括两个部分
 - 存储数据元素的数据域
 - 存储下一个结点地址的指针域

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

- | | | |
|--------------|-------|---------|
| • 结点 | 新数据类型 | Student |
| • 存储的数据 | 姓名 | string |
| • 存储下一个结点的地址 | 指针 | |

链表：创建结点的数据类型

- 每个结点包括两个部分
 - 存储数据元素的数据域
 - 存储下一个结点地址的指针域

```
struct 新类型的名称{  
    成员变量的声明语句;  
};
```

- | • 结点 | 新数据类型 | Student |
|--------------|---------|-----------|
| • 存储的数据 | 姓名 | string |
| • 存储下一个结点的地址 | 指针 | Student * |
| • 结点的数据类型是 | Student | |

链表：创建结点的数据类型

```
struct Student{  
    string name;  
    Student *next;  
};
```


链表：创建结点的数据类型

```
struct Student{  
    string name;  
    Student *next;  
};
```

- **定义新数据类型** Student

链表：创建结点的数据类型

```
struct Student{  
    string name;  
    Student *next;  
};
```

- **类型 Student 由两部分组成**

链表：创建结点的数据类型

```
struct Student{  
    string name;  
    Student *next;  
};
```

- **类型 Student 由两部分组成**
 - **成员变量** name

链表：创建结点的数据类型

```
struct Student{  
    string name;  
    Student *next;  
};
```

- **类型 Student 由两部分组成**
 - **成员变量** name
 - **成员变量** next

链表：创建结点的数据类型

```
struct Student{  
    string name;  
    Student *next;  
};
```

- **类型 Student 由两部分组成**
 - **成员变量 name**
 - **成员变量 next (存储Student变量地址的指针)**

链表的建立：图示



链表的建立

head 

- 头指针
 - 记录第一个结点的地址并保持不动

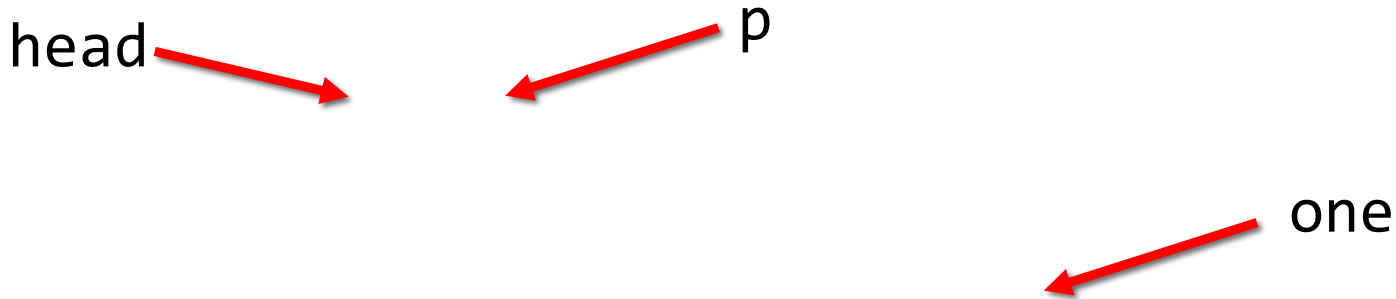
链表的建立



- 临时指针
 - 记录当前结点的地址
 - 向链表尾部添加新的结点

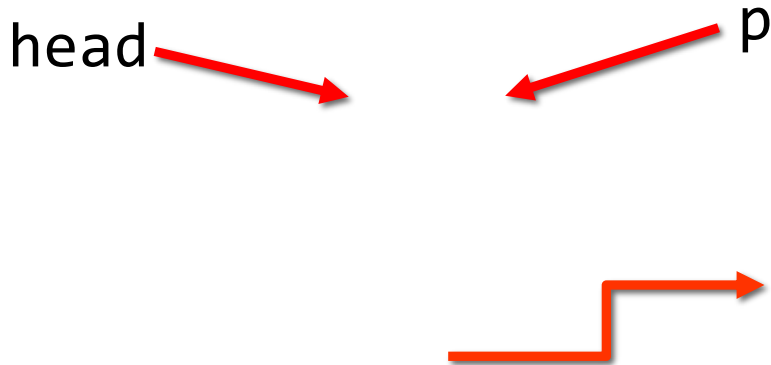
链表的建立

```
Student *one = new Student();  
one->name = "Jerry";
```



- 向链表尾部添加新的结点
 1. 动态创建新的结点
 2. 把新结点添加至链表尾部

链表的建立

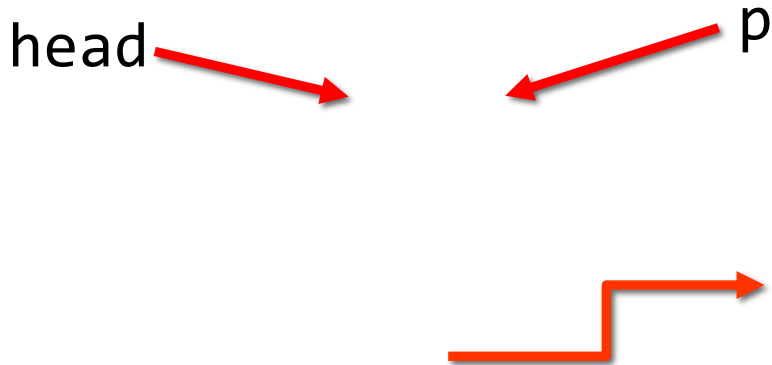


```
Student *one = new Student();  
one->name = "Jerry";  
p->next = one;
```



- 向链表尾部添加新的结点
 1. 动态创建新的结点
 2. 把新结点添加至链表尾部（连接前后两个结点）

链表的建立



```
Student *one = new Student();
```

```
one->name = "Jerry";
```

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


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



- 把指针 `p` 移动至新增的尾部结点

链表的建立

head 


p 

```
Student *one = new Student();
```

```
one->name = "Jerry";
```

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

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

 one



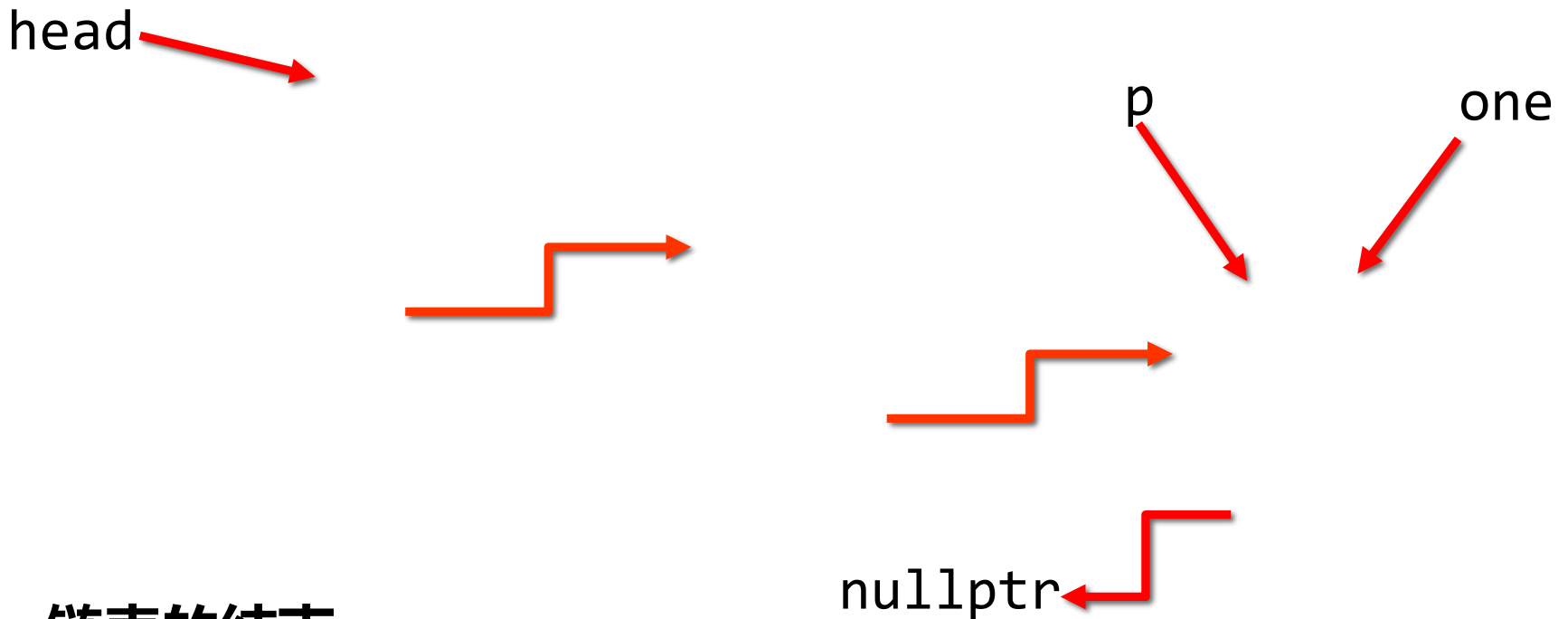
- 把指针 p 移动至新增的尾部结点
- 继续动态创建新结点并陆续添加至链表的尾部

链表的建立



- 链表的结束
 - 尾部结点的 next 指向 nullptr

链表的建立



- 链表的结束

- 尾部结点的 next 指向 nullptr (代表链表结束)

- `one->next = nullptr; // p->next = nullptr;`

链表的建立

```
struct Student{
    string name;
    Student *next;
};
int main(){
    Student *head = nullptr;
    Student *p = nullptr;
    string name ;
    while(cin>>name){
        Student *one = new Student();
        one->name = name;
        one->next = nullptr;

        if(head == nullptr){
            head = one;
            p = one;
        }
        else{
            p->next = one;
            p = p->next;
        }
    }
    return 0;
}
```



```
Student *head = nullptr;
Student *p = nullptr;

string name ;
while(cin>>name){
    Student *one = new Student();
    one->name = name;
    one->next = nullptr;

    if(head == nullptr){
        head = one;
        p = one;
    }
    else{
        p->next = one;
        p = p->next;
    }
}
```

链表的建立

```
Student *head = nullptr;  
Student *p = nullptr;  
string name ;  
while(cin>>name){  
    Student *one = new Student();  
    one->name = name;  
    one->next = nullptr;  
  
    if(head == nullptr){  
        head = one;  
        p = one;  
    }  
    else{  
        p->next = one;  
        p = p->next;  
    }  
}
```

- 声明链表的头指针
- 维护整个链表

链表的建立

```
Student *head = nullptr;
Student *p = nullptr;
string name ;
while(cin>>name){
    Student *one = new Student();
    one->name = name;
    one->next = nullptr;

    if(head == nullptr){
        head = one;
        p = one;
    }
    else{
        p->next = one;
        p = p->next;
    }
}
```

- 声明指针 p
 - 记录链表的当前结点
 - 连接结点
 - 结点之间的移动
 - 确保 p 指向尾部结点

链表的建立

```
Student *head = nullptr;
Student *p = nullptr;
string name ;
while(cin>>name){
    Student *one = new Student();
    one->name = name;
    one->next = nullptr;

    if(head == nullptr){
        head = one;
        p = one;
    }
    else{
        p->next = one;
        p = p->next;
    }
}
```

- 输入姓名
- 按 Ctrl + Z 结束循环

链表的建立

- 创建一个新的结点 one

```
Student *head = nullptr;
Student *p = nullptr;
string name ;
while(cin>>name){
    Student *one = new Student();
    one->name = name;
    one->next = nullptr;

    if(head == nullptr){
        head = one;
        p = one;
    }
    else{
        p->next = one;
        p = p->next;
    }
}
```

链表的建立

```
Student *head = nullptr;
Student *p = nullptr;
string name ;
while(cin>>name){
    Student *one = new Student();
    one->name = name;
    one->next = nullptr;

    if(head == nullptr){
        head = one;
        p = one;
    }
    else{
        p->next = one;
        p = p->next;
    }
}
```

- 创建一个新的结点 one
- 给结点 one 的成员赋值

链表的建立

```
Student *head = nullptr;
Student *p = nullptr;
string name ;
while(cin>>name){
    Student *one = new Student();
    one->name = name;
    one->next = nullptr;

    if(head == nullptr){
        head = one;
        p = one;
    }
    else{
        p->next = one;
        p = p->next;
    }
}
```

- 创建一个新的结点 one
- 给结点 one 的成员赋值
- 结点 one 没有后续结点

链表的建立

```
Student *head = nullptr;
Student *p = nullptr;
string name ;
while(cin>>name){
    Student *one = new Student();
    one->name = name;
    one->next = nullptr;
```

- 把新结点添加至链表尾部

```
    if(head == nullptr){
        head = one;
        p = one;
    }
    else{
        p->next = one;
        p = p->next;
    }
}
```

链表的建立

```
Student *head = nullptr;
Student *p = nullptr;
string name ;
while(cin>>name){
    Student *one = new Student();
    one->name = name;
    one->next = nullptr;

    if(head == nullptr){
        head = one;
        p = one;
    }
    else{
        p->next = one;
        p = p->next;
    }
}
```

- 把新结点添加至链表尾部
- 如果链表尚未建立

链表的建立

```
Student *head = nullptr;
Student *p = nullptr;
string name ;
while(cin>>name){
    Student *one = new Student();
    one->name = name;
    one->next = nullptr;

    if(head == nullptr){
        head = one;
        p = one;
    }
    else{
        p->next = one;
        p = p->next;
    }
}
```

- 把新结点添加至链表尾部
- 如果链表尚未建立，那么开始建立链表。
 - head 和 p 指向链表中的第一个结点。

链表的建立

```
Student *head = nullptr;
Student *p = nullptr;
string name ;
while(cin>>name){
    Student *one = new Student();
    one->name = name;
    one->next = nullptr;

    if(head == nullptr){
        head = one;
        p = one;
    }
    else{
        p->next = one;
        p = p->next;
    }
}
```

- 把新结点添加至链表尾部
- 如果链表已经建立

链表的建立

```
Student *head = nullptr;
Student *p = nullptr;
string name ;
while(cin>>name){
    Student *one = new Student();
    one->name = name;
    one->next = nullptr;

    if(head == nullptr){
        head = one;
        p = one;
    }
    else{
        p->next = one;
        p = p->next;
    }
}
```

- 把新结点添加至链表尾部
- 如果链表已经建立，那么
 - 把新结点添加至链表的尾部。

链表的建立

```
Student *head = nullptr;
Student *p = nullptr;
string name ;
while(cin>>name){
    Student *one = new Student();
    one->name = name;
    one->next = nullptr;

    if(head == nullptr){
        head = one;
        p = one;
    }
    else{
        p->next = one;
        p = p->next;
    }
}
```

- 把新结点添加至链表尾部
- 如果链表已经建立，那么
 - 把新结点添加至链表的尾部。

链表的建立

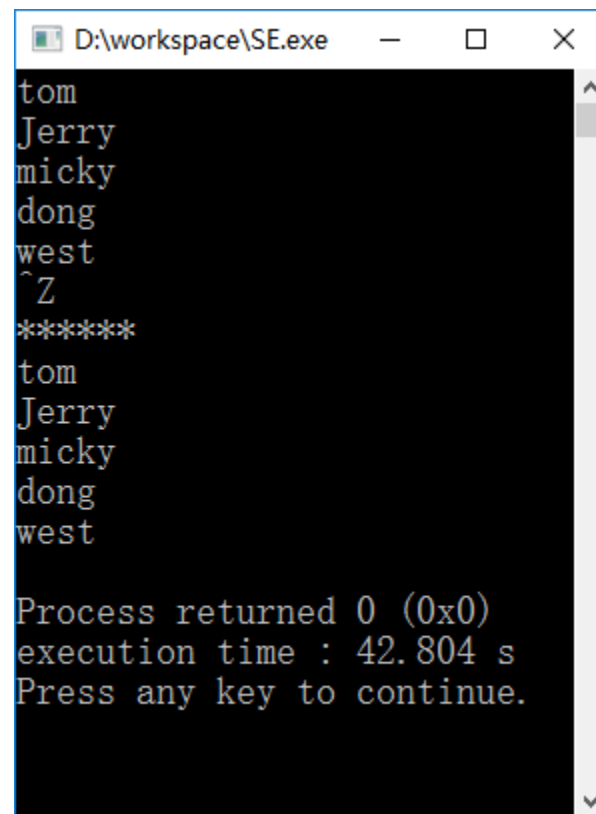
```
Student *head = nullptr;
Student *p = nullptr;
string name ;
while(cin>>name){
    Student *one = new Student();
    one->name = name;
    one->next = nullptr;

    if(head == nullptr){
        head = one;
        p = one;
    }
    else{
        p->next = one;
        p = p->next;
    }
}
```

- 把新结点添加至链表尾部
- 如果链表已经建立，那么
 - 把新结点添加至链表的尾部。
 - 随后，把指针 p 移动至新的尾部结点。

遍历链表

```
int main(){  
  
    Student *head = nullptr;  
    Student *p = nullptr;  
  
    string name ;  
    while(cin>>name){  
  
        cout << "*****" <<endl;  
        while(head!=nullptr){  
            cout << head->name <<endl;  
            head = head->next;  
        }  
  
        return 0;  
    }  
}
```



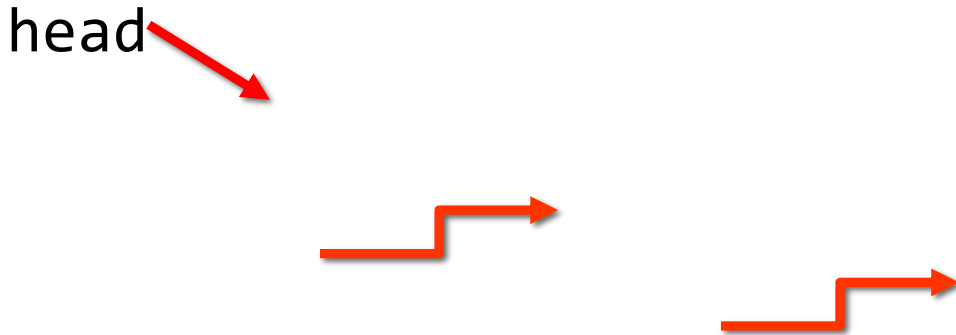
```
D:\workspace\SE.exe  
tom  
Jerry  
micky  
dong  
west  
^Z  
*****  
tom  
Jerry  
micky  
dong  
west  
  
Process returned 0 (0x0)  
execution time : 42.804 s  
Press any key to continue.
```

- 遍历链表中的每一个结点并输出每个结点的成员变量值

遍历链表

```
while( head != nullptr ){  
    cout << head->name <<endl;  
    head = head->next;  
}
```

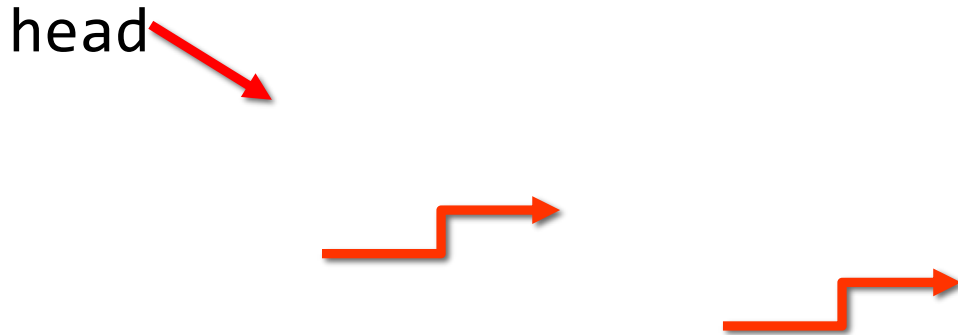
- 输出头指针所指向的当前结点的成员变量 name 的值



遍历链表

```
while( head != nullptr ){  
    cout << head->name <<endl;  
    head = head->next;  
}
```

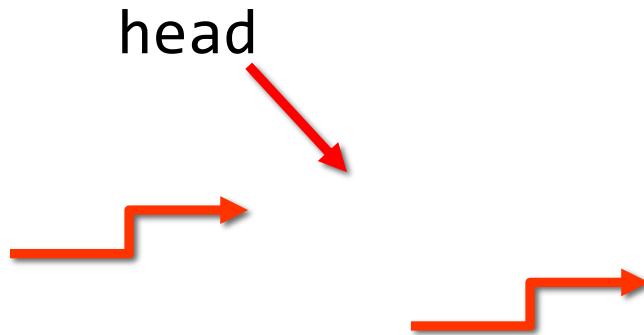
- 移动头指针



遍历链表

```
while( head != nullptr ){  
    cout << head->name <<endl;  
    head = head->next;  
}
```

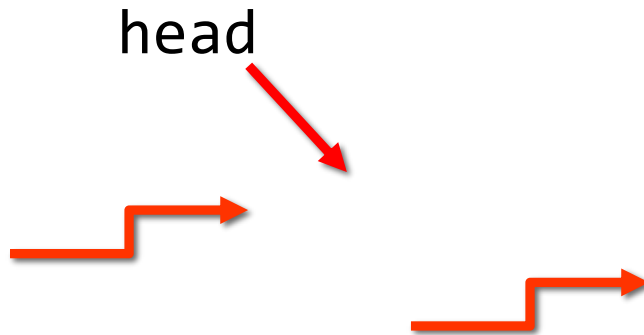
- 移动头指针



遍历链表

```
while( head != nullptr ){  
    cout << head->name <<endl;  
    head = head->next;  
}
```

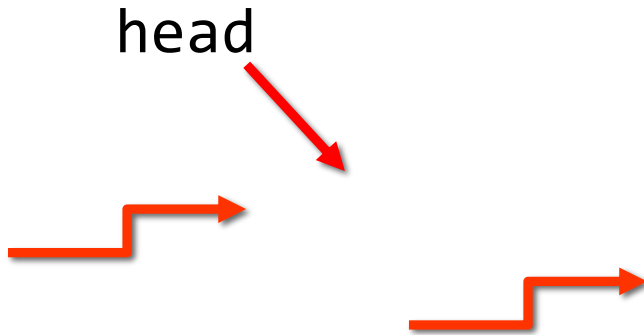
- **移动头指针** （头指针指向下一个结点）



遍历链表

```
while( head != nullptr ){  
    cout << head->name <<endl;  
    head = head->next;  
}
```

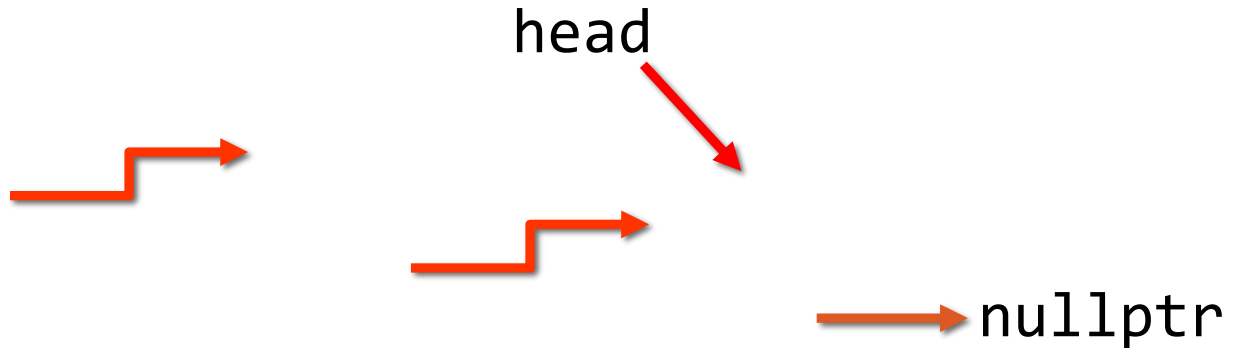
- 判断链表中是否存在后续结点



遍历链表

```
while( head != nullptr ){  
    cout << head->name <<endl;  
    head = head->next;  
}
```

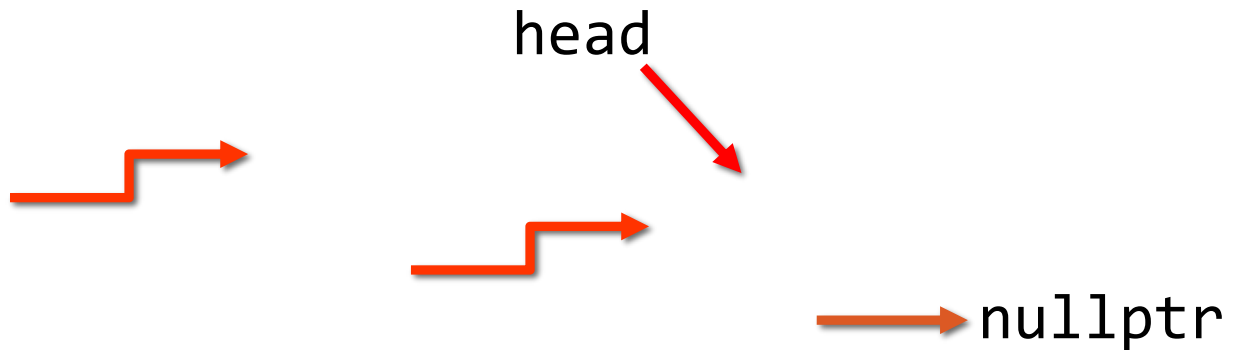
- 判断链表中是否存在后续结点



遍历链表

```
while( head != nullptr ){  
    cout << head->name <<endl;  
    head = head->next;  
}
```

- **nullptr 表示没有后续结点（链表结束）**



遍历链表

```
while( head != nullptr ){  
    cout << head->name <<endl;  
    head = head->next;  
}
```

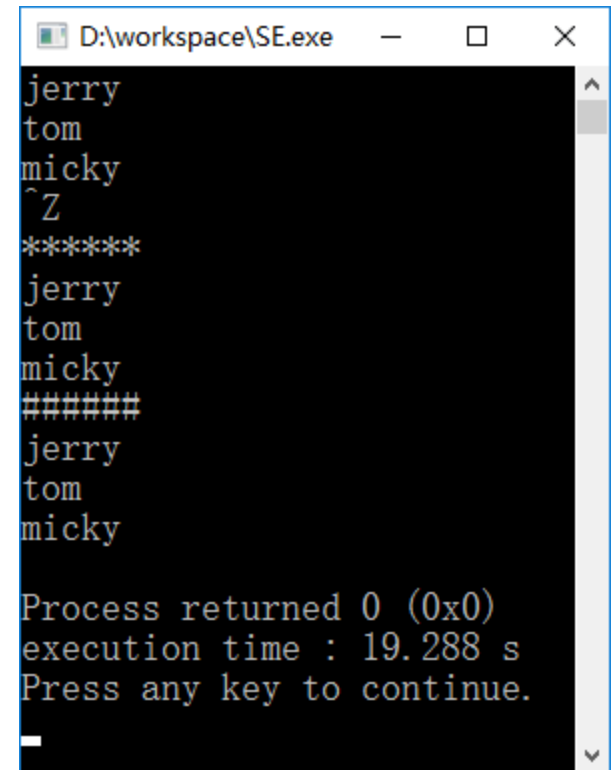
- **存在的问题**

- **修改头指针**
- **导致链表不可重用**
- **head 最终指向 nullptr**

遍历链表

```
cout << "*****" <<endl;
Student *p1 = head;
while(p1!=nullptr){
    cout << p1->name <<endl;
    p1 = p1->next;
}
cout << "#####" <<endl;
Student *p2 = head;
while(p2!=nullptr){
    cout << p2->name <<endl;
    p2 = p2->next;
}
```

- 保持头指针不变



```
D:\workspace\SE.exe
jerry
tom
micky
^Z
*****
jerry
tom
micky
#####
jerry
tom
micky

Process returned 0 (0x0)
execution time : 19.288 s
Press any key to continue.
```

遍历链表

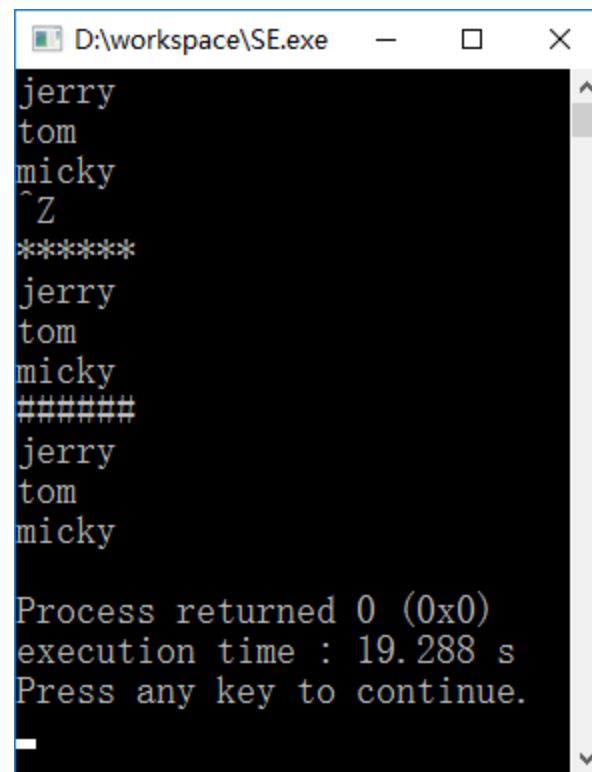
```
cout << "*****" <<endl;
```

```
Student *p1 = head;  
while(p1!=nullptr){  
    cout << p1->name <<endl;  
    p1 = p1->next;  
}
```

```
cout << "#####" <<endl;
```

```
Student *p2 = head;  
while(p2!=nullptr){  
    cout << p2->name <<endl;  
    p2 = p2->next;  
}
```

• 代码复用 (函数)



```
D:\workspace\SE.exe  
jerry  
tom  
micky  
^Z  
*****  
jerry  
tom  
micky  
#####  
jerry  
tom  
micky  
  
Process returned 0 (0x0)  
execution time : 19.288 s  
Press any key to continue.
```

遍历链表：函数

```
void printlist(Student *p){  
    while(p != nullptr){  
        cout << p->name <<endl;  
        p = p->next;  
    }  
    return ;  
}
```

- 形式参数 p
 - 地址传递
 - 可以改变实际参数

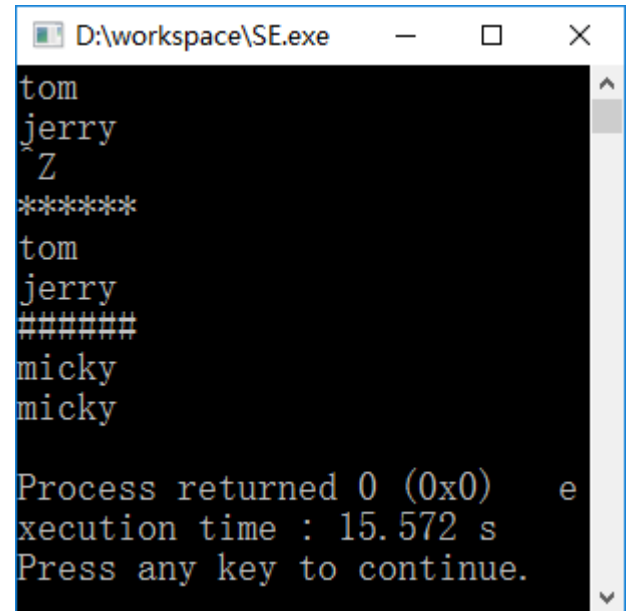
```
cout << "*****" <<endl;  
printlist(head);  
  
cout << "#####" <<endl;  
Student *p2 = head;  
while(p2!=nullptr){  
    cout << p2->name <<endl;  
    p2 = p2->next;  
}
```


遍历链表：函数

```
void printlist(Student * p){
    while(p!=nullptr){
        cout << p->name <<endl;
        p->name ="micky";
        p = p->next;
    }
    return ;
}

int main(){
    .....
    cout << "*****" <<endl;
    printlist(head);

    cout << "#####" <<endl;
    Student *p2 = head;
    while(p2!=nullptr){
        cout << p2->name <<endl;
        p2 = p2->next;
    }
    return 0;
}
```



```
D:\workspace\SE.exe
tom
jerry
Z
*****
tom
jerry
#####
micky
micky

Process returned 0 (0x0)
Execution time : 15.572 s
Press any key to continue.
```

遍历链表：函数

```
void printlist(const Student * p){  
    while(p!=nullptr){  
        cout << p->name <<endl;  
        p = p->next;  
    }  
    return ;  
}
```

- 指向常量的指针（形式参数）

遍历链表：函数

```
void printlist(const Student * p){  
    while(p!=nullptr){  
        cout << p->name <<endl;  
        p = p->next;  
    }  
    return ;  
}
```

- 指向常量的指针（形式参数） 避免修改实际参数值

遍历链表

```
SE.cpp - Code::Blocks 16.01
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

<global>
printlist(Student* const p) : void

Management
Projects Symbols
Workspace

3
4 struct Student{
8
9 void printlist(Student * const p){
10 while(p!=nullptr){
11     cout << p->name <<endl;
12     p = p->next;
13 }
14 return ;
15 }
16
17 int main(){
50

Logs & others
Code::Blocks Search results Cccc Build log Build messages CppCheck CppCheck messages Cscope Debugger

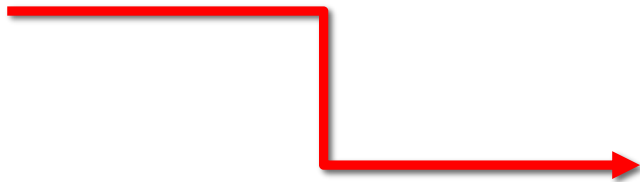
File Line Message
D:\workspace\S... == Build file: "no target" in "no project" (compiler: unknown) ==
In function 'void printlist(Student*)':
D:\workspace\S... 12 error: assignment of read-only parameter 'p'
== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ==

D:\workspace\SE.cpp Windows (CR+LF) WINDOWS-936 Line 12, Column 1 Insert Read/Write default
```

- 形式参数 p 是指针常量（不能移动）

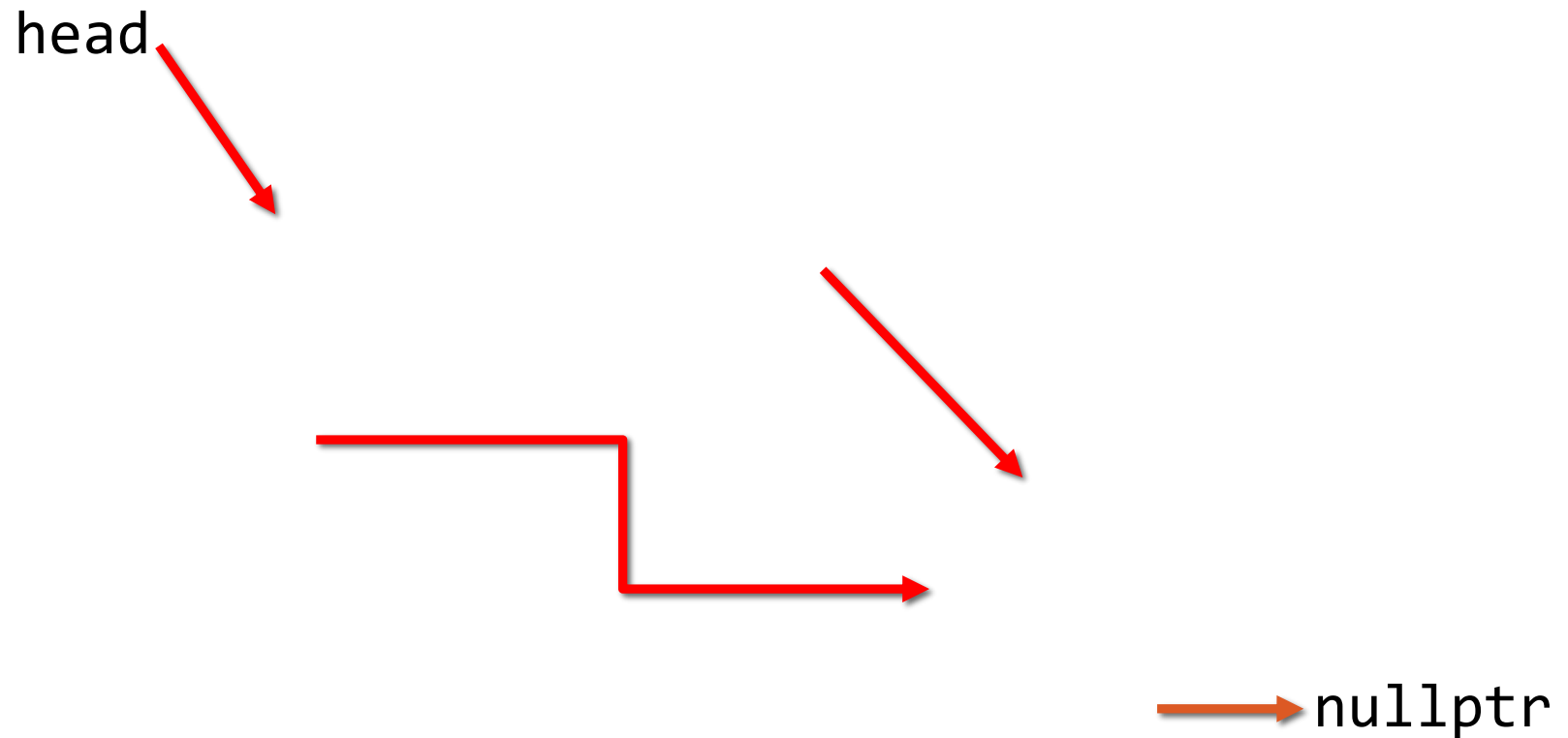
链表：插入结点

head



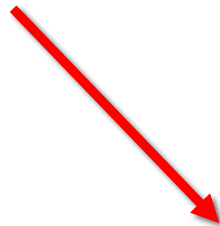
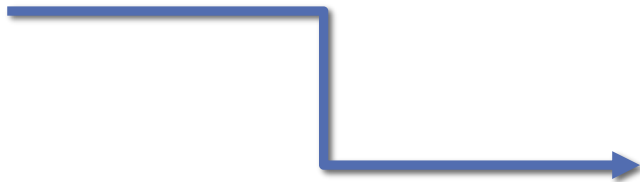
→ nullptr

链表：插入结点



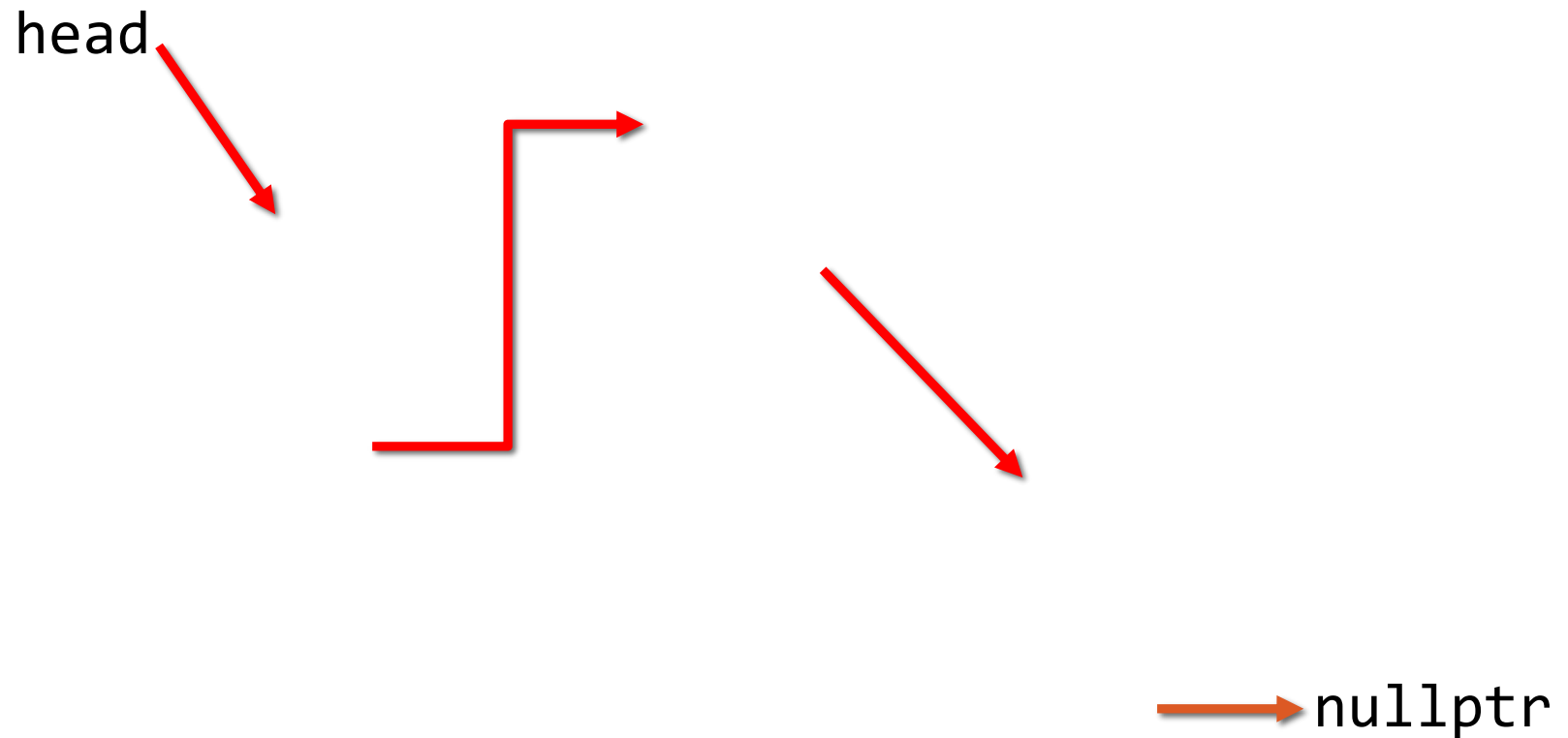
链表：插入结点

head



→ nullptr

链表：插入结点



链表：插入结点

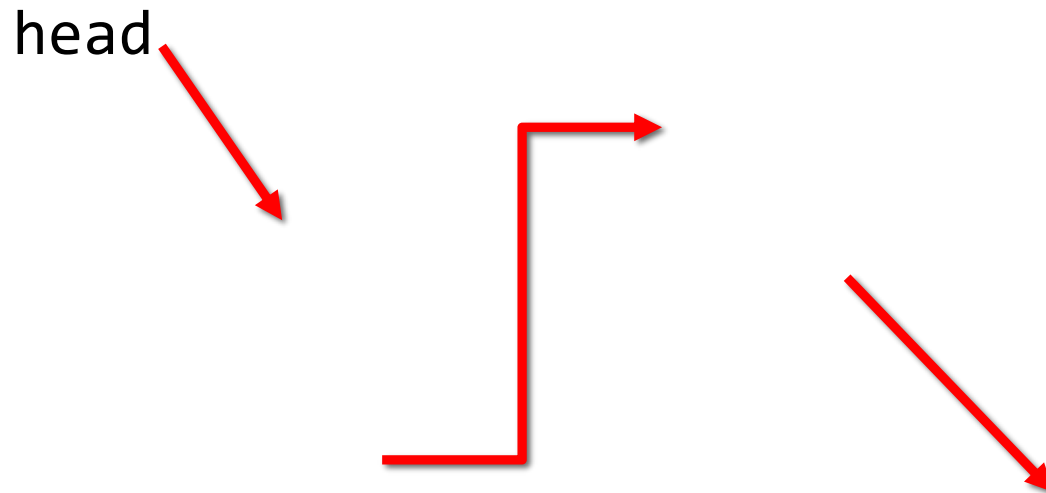
```
Student *p = head; //指针 p 指向链表的第一个元素
```

```
Student *insert_node = new Student(); //创建新结点  
insert_node->name = "seven";  
insert_node->next = nullptr;
```

.....//找到插入位置（p指向插入位置）。假设在第一个结点后插入新结点。

```
if(p->next==nullptr){ //判断插入点是否位于链表尾部  
    p->next = insert_node;  
}  
else{  
    insert_node->next = p->next;  
    p->next = insert_node;  
}
```

链表：删除结点

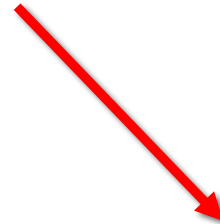
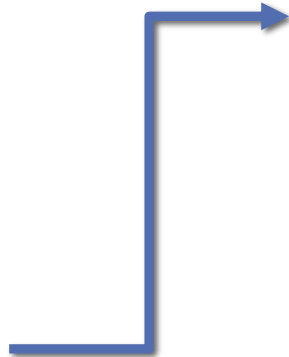


- 删除结点 “Jerry”

→ nullptr

链表：删除结点

head

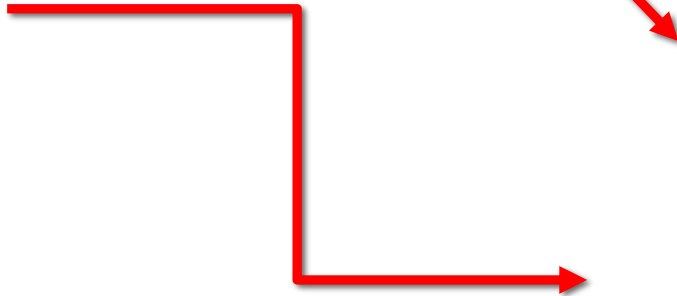


- 删除结点 “Jerry”

—→ nullptr

链表：删除结点

head

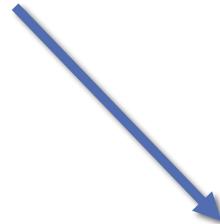
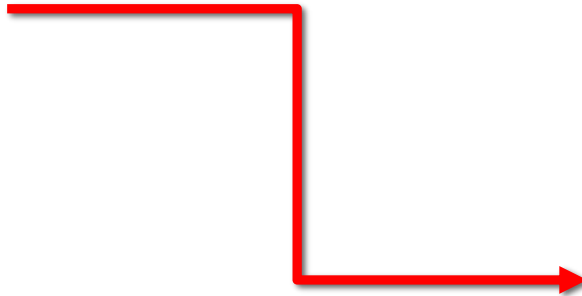


- 删除结点 “Jerry”

→ nullptr

链表：删除结点

head

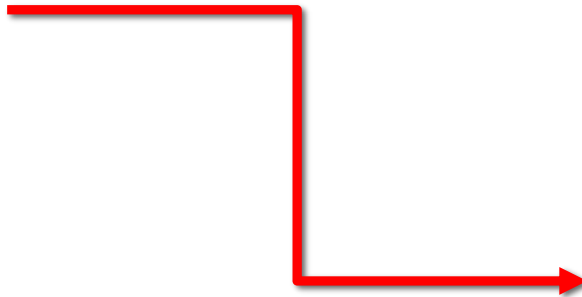


→ nullptr

- 删除结点 “Jerry”

链表：删除结点

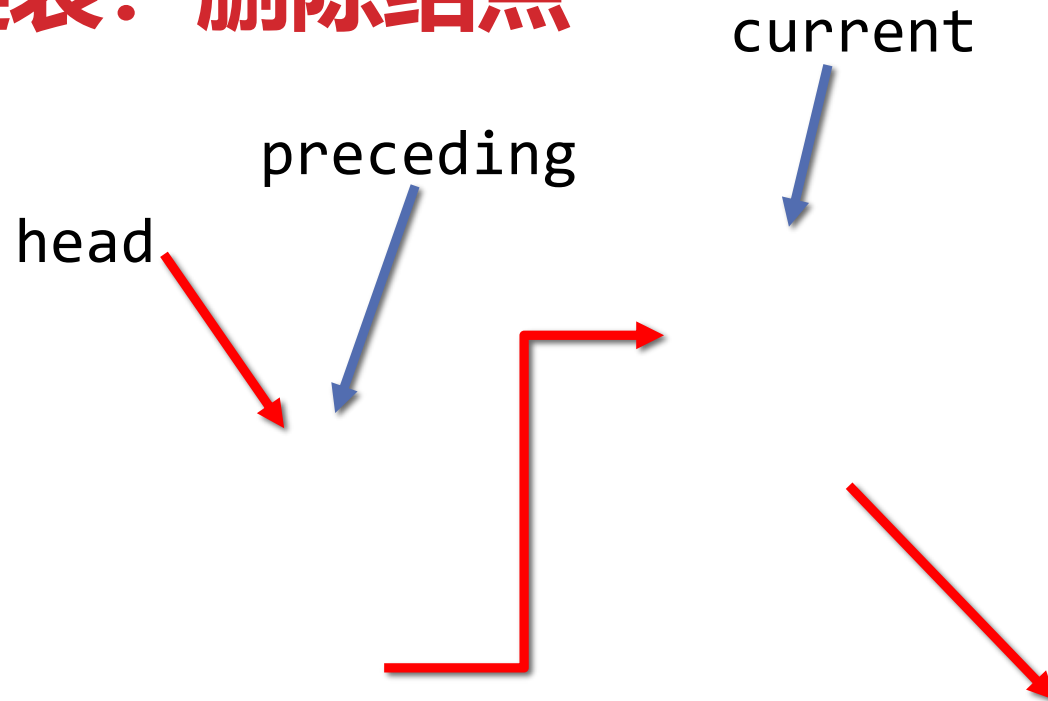
head



→ nullptr

- 删除结点 “Jerry”

链表：删除结点



- 删除结点 “Jerry”

→ nullptr

链表：删除结点

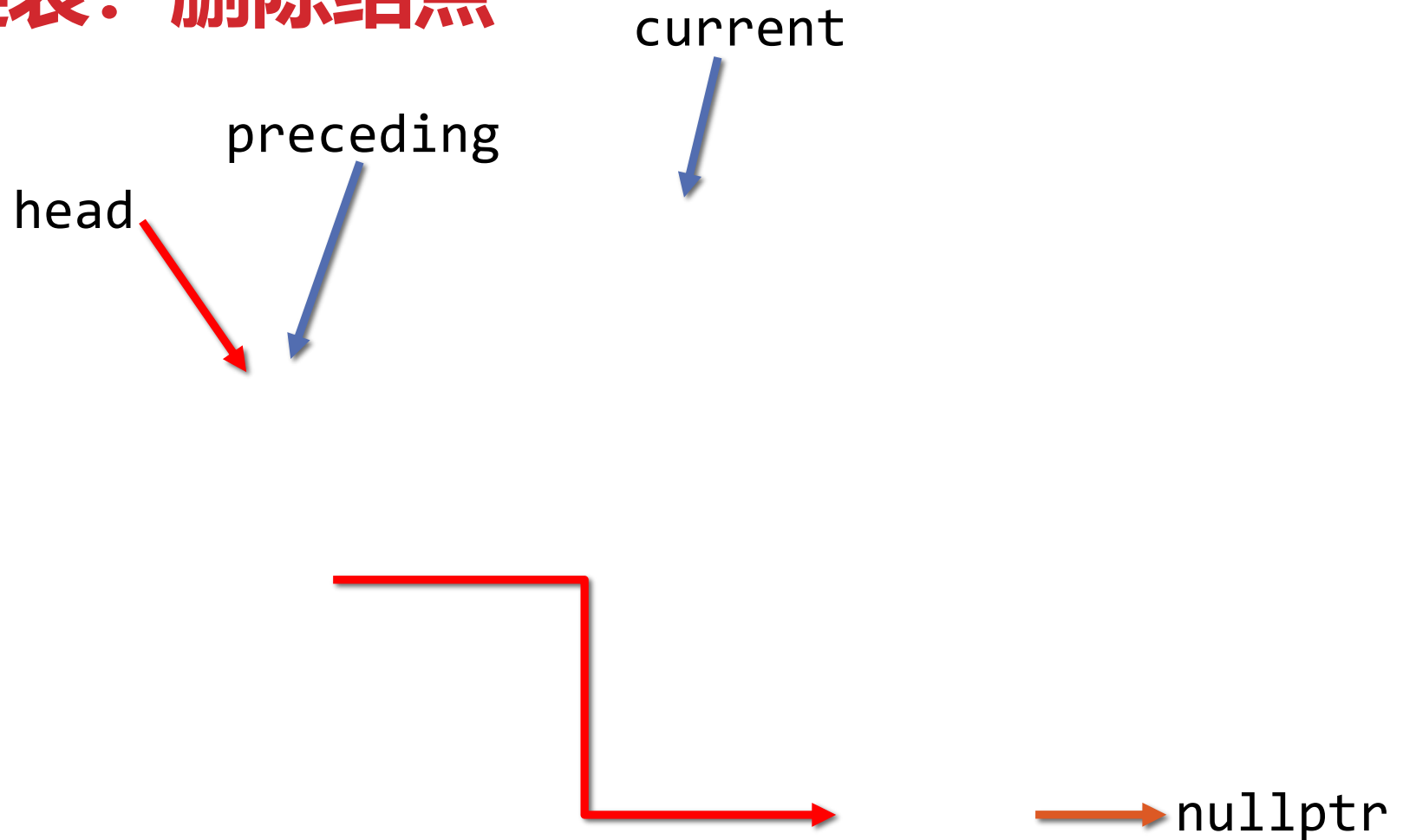
```
Student *preceding = head;  
Student *current = head;
```

```
//寻找待删除结点。假设，删除第二个结点。  
//移动指针，分别指向相邻的前后两个结点。  
preceding = current;  
current = current->next;
```

```
if(current->next==nullptr){//判断待删结点是否是链表的尾部结点  
    preceding->next = nullptr;  
}  
else{  
    preceding->next = current->next;  
}
```

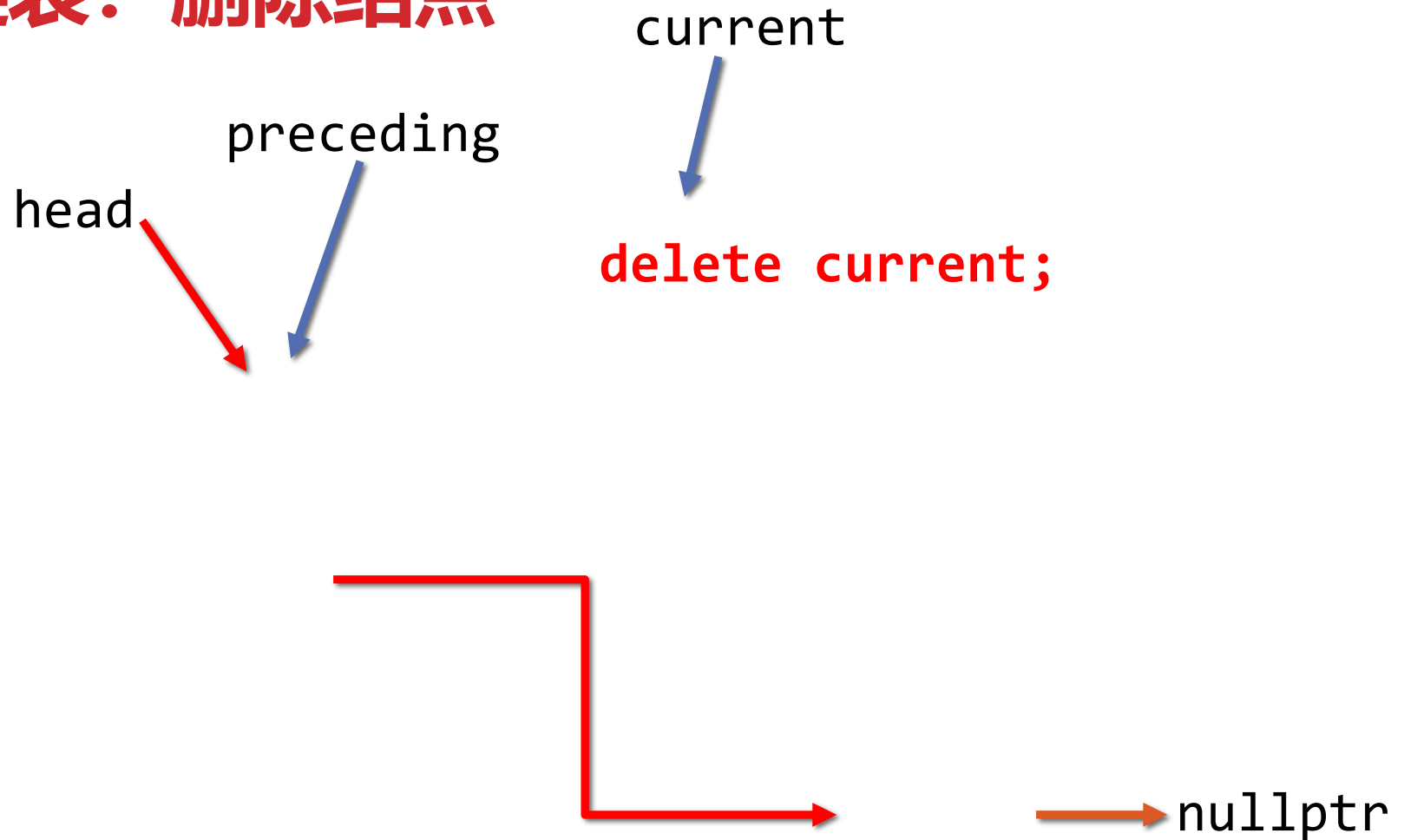
```
delete current; //释放已删结点所占据的内存空间
```


链表：删除结点



- 释放结点 “Jerry” 的内存空间

链表：删除结点



- 释放结点 “Jerry” 的内存空间

链表

- C++ STL
 - list容器
 - 链表
 - vector容器
 - 支持随机访问
 - 在增加数据时，如果原先分配的连续内存已经用完，那么需要重新分配内存并对原有数据进行复制。

待续.....