数据结构与算法
（二）
基础数据结构补充：链表和list

杜育根

ygdu@sei.ecnu.edu.cn

链式存储结构

- 链表是一种链式存储结构，对于链表的每个数据元素来说，除了要存储它本身的信息（数据域、data）外，还要存储它的直接后继元素的存储位置（指针域、link 或 next）。把这两部分信息合在一起称为一个“节点node”。在程序的执行过程中，通过向计算机随时申请存储空间或随时释放存 储空间，以达到动态管理、使用计算机的存储空间，保证存储资源的充分利用。这样的存储方式称为动态存储，所定义的变量称为动态变量。它的优点如下：
- 【优点】：可以用一组任意的存储单元（这些存储单元可以是连续的，也可以不连续的）存储线性 表的数据元素，这样就可以充分利用存储器的零碎空间；

C语言链表的定义和操作

（一）单链表

1．类型和变量的说明

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

2．申请存储单元　　//动态申请、空间大小由指针变量的基类型决定

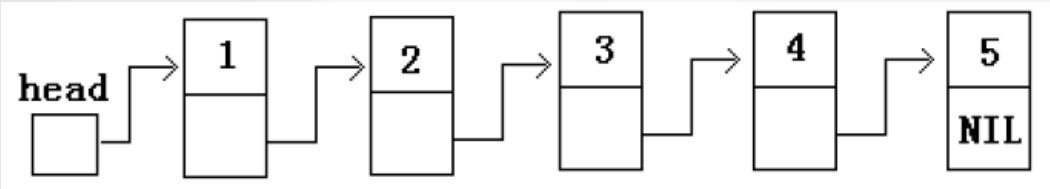
```
p=new Node;
```

3．指针变量的赋值

```
指针变量名=NULL;　//初始化，暂时不指向任何存储单元
```

如何表示和操作指针变量？不同于简单变量（如 A=0；），c/c++规定用“指针变量名->”的 形式引用指针变量（如 p->data=0；）

单链表的结构建立、输出



- Node *head,*p;
- head=(struct Node *)malloc(sizeof(struct Node)); //申请头节点空间
- p=(struct Node*)malloc(sizeof(struct Node)); //申请一个新节点
- p->data=x;//给新节点数据赋值
- p->next=NULL;//暂时不指向任何存储单元
- head->next=p;//head下一个指向当前p节点

例子 单链表创建

```c
#include<stdio.h>
#include <stdlib.h>
struct Node{
    int data;
    struct Node *next;
} *head,*p,*r; //r指向链表的当前最后一个结点，可以称为尾指针

int main()
{   int x;
    head=(struct Node *)malloc(sizeof(struct Node));//申请头节点
    head->next=NULL;
    r=head;
    while(scanf("%d",&x)==1){
        p=(struct Node*)malloc(sizeof(struct Node)); //申请一个新节点
        p->data=x;
        p->next=NULL;
        r->next=p; //把新结点链接到前面的链表中，实际上r是p的直接前趋
        r=p; //尾指针后移一个
    }
    p=head->next; //头指针没有数据，只要从第一个结点开始就可以了
    while(p->next!=NULL){
        printf("%d ",p->data);
        p=p->next;
    }
    printf("%d\n",p->data); //最后一个结点的数据单独输出
}
```

查找“数据域满足一定条件的结点”

- //找到第一个就结束

```
p=head->next;
while((p->data!=x)&&(p->next!=NULL) p=p->next;
if(p->data==x)找到了处理;
else 输出不存在;
```

- //如果想找到所有满足条件的结点，则修改如下：

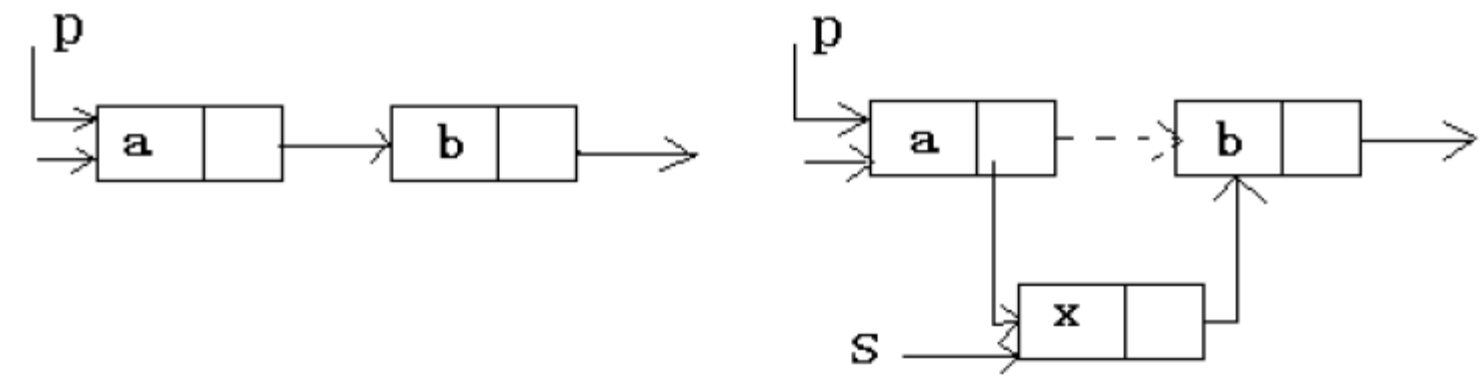
```
p=head->next;
while(p->next!=NULL) //一个一个判断
{
    if(p->data==x)找到一个处理一个;
    p=p->next;
}
```

取出单链表的第i个结点的数据域

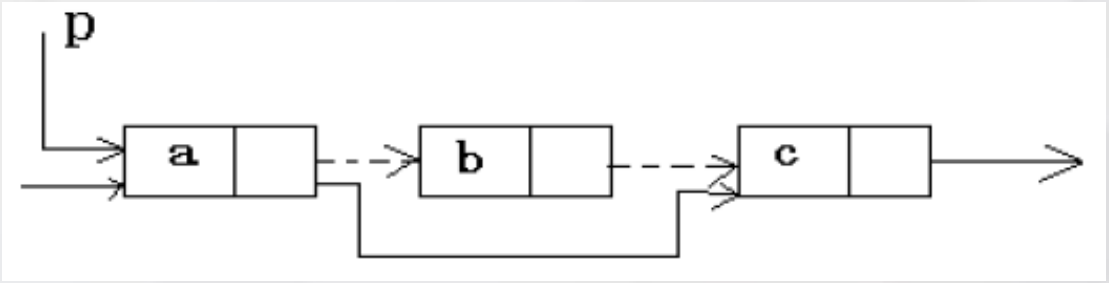
```
void get(struct Node *head,int i){
    struct Node *p;int j;
    p=head->next;
    j=1;
    while((p!=NULL)&&(j<i)){
        p=p->next;
        j=j+1;
    }
    if((p!=NULL)&&(j==i) printf( “%d” ,p->data);
    else printf( “%d not exsit! “,i)
}
```

插入一个结点在单链表中去

```
void insert(struct Node *head,int i,int x) {//插入X到第i个元素之前
    struct Node *p,*s;int j;
    p=head;
    j=0;
    while((p!=NULL)&&(j<i-1)) {//寻找第i-1个结点,插在它的后面
        p=p->next;
        j=j+1;
    }
    if(p==NULL) printf("no this position! “);
    else{ //插入
        s=new Node;
        s->data=x;
        s->next=p->next;
        p->next=s;
    };
}
```



删除单链表中的第i个结点（如下图的“b”结点）




```
void delete(Node *head,int i) {//删除第i个元素
    Node *p,*s;int j;
    p=head;
    j=0;
    while((p->next!=NULL)&&(j<i-1)){
        p=p->next;
        j=j+1;
    } //p指向第i-1个结点
    if ((p->next==NULL)||(i<=0))  printf("no this position! ");
    else { //删除p的后继结点，假设为s
        s=p->next;
        p->next=s->next; // p->next指向p->next ->next
        free(s);
    }
}
```

求单链表的实际长度

```
int len(struct Node *head) {
    int n=0;
    p=head;
    while(p!=NULL) {
        n=n+1;
        p=p->next;
    }
    return n;
}
```

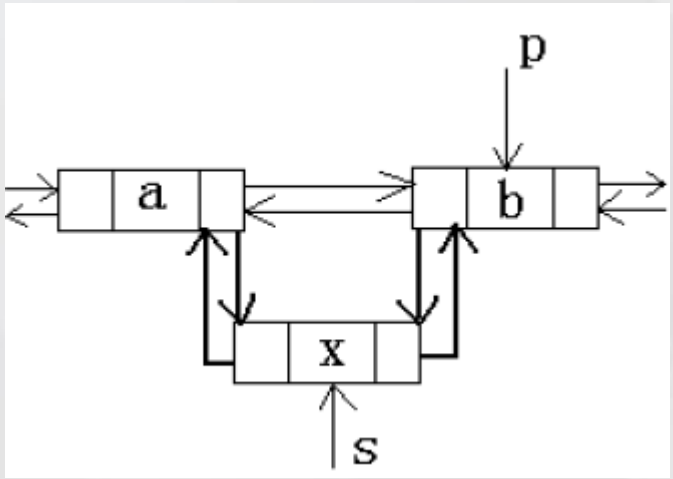
（二）双向链表

- 双向链表的每个结点有两个指针域和若干数据域，其中一个指针域指向它的前趋结点，一个指向它的后继结点。它的优点是访问、插入、删除更方便，速度也快了。但“是以空间换时间”。
- 【数据结构的定义】：

```
struct node
{
    int data;
    struct node *pre,*next; //pre指向前趋，next指向后继
}
struct node *head,*p,*q,*r;
```

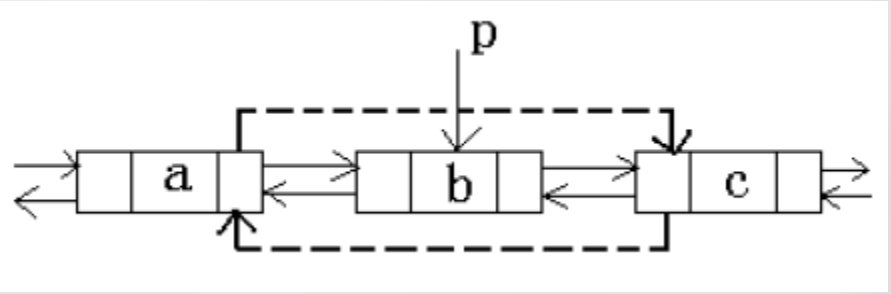
双向链表的第i个结点之前插入X

```
void insert(struct node *head,int i,int x) {//在双向链表的第i个结点之前插入X
        struct node *s,*p;
        int j;
        s=(struct Node *)malloc(sizeof(struct Node));
        s->data=x;
        p=head;
        j=0;
        while((p->next!=NULL)&&(j<i)) {
                p=p->next;
                j=j+1;
        } //p指向第i个结点
        if(p==NULL) printf("no this position! ");
        else { //将结点S插入到结点P之前
                s->pre=p->pre; //将S的前趋指向P的前趋
                p->pre=s; //将S作为P的新前趋
                s->next=p; //将S的后继指向P
                p->pre->next=s; //将P的本来前趋结点的后继指向S
        }
}
```



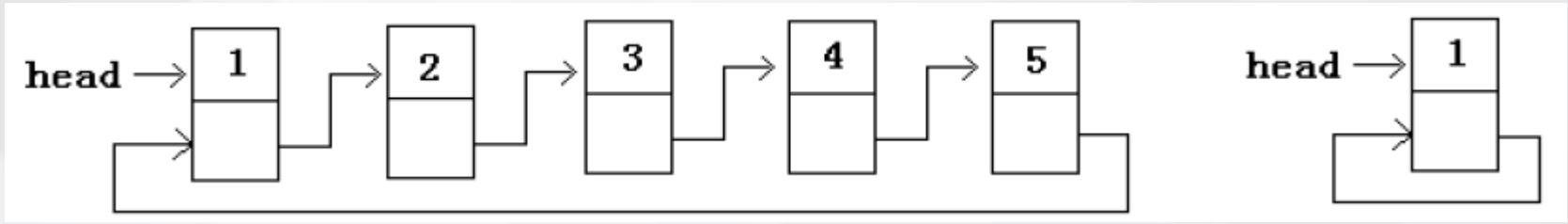
删除双向链表的第i个结点

```
void delete(node *head,int i) {//删除双向链表的第i个结点
    int j;
    node *p;
    p=head;
    j=0;
    while((p->next!=NULL)&&(j<i)) {
        p=p->next;
        j=j+1;
    } //p指向第i个结点
    if(p==NULL) cout<<"no this position!";
    else { //将结点P删除
        p->pre->next=p->next; //P的前趋结点的后继赋值为P的后继
        p->next->pre=p->pre; //P的后继结点的前趋赋值为P的前趋
    }
}
```

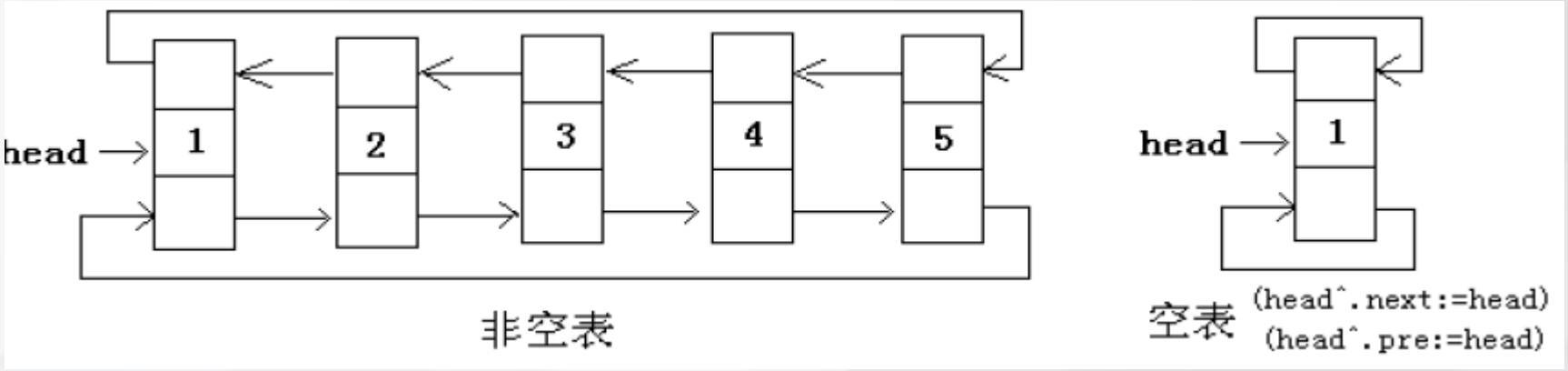


（三）循环链表

- 单向循环链表：最后一个结点的指针指向头结点。如下图：



- 双向循环链表：最后一个结点的指针指向头结点，且头结点的前趋指向最后一个结点。如下图：



练习题 约瑟夫环问题

【问题描述】有M个人，其编号分别为1 - M。这M个人按顺序排成一个圈。现在给定一个数N，从第一个人开始依次报数，数到N的人出列，然后又从下一个人开始又从1开始依次报数，数到N的人又出列．．．如此循环，直到最后一个人出列为止。

【输入格式】输入只有一行，包括2个整数M，N。之间用一个空格分开(0 < n <= m <= 100)。

【输出格式】

输出只有一行，包括M个整数

【样列输入】

8 5

【样列输出】

5 2 8 7 1 4 6 3

C++ STL list

list

- **STL的 list ： stl实现的双向链表。它的内存空间不必连续，通过指针来进行数据的访问，高效率地在任意地方删除和插入，插入和删除操作是常数时间。**
- **list和vector的优缺点正好相反，它们的应用场景不同：**

  (1) vector：插入和删除操作少，随机访问元素频繁；

  (2) list：插入和删除频繁，随机访问较少。

List定义和初始化

- 头文件和命名空间

```
#include <list>
using namespace std;
```

- 定义和初始化

```
list<int>lst1;          //创建名位lst1的空list
list<int> lst2(5);      //创建含有5个默认值是0的元素的list
list<int>lst3(3,2);  //创建含有3个元素的list，值都是2
list<int>lst4(lst2);    //复制lst2的lst4
list<int>lst5(lst2.begin(),lst2.end());  //同上一句
list<int> lst6{1,2,3,4,5};
```

遍历、访问List

- .begin()　返回指向链表第一个元素的迭代器。
- .end()　　返回指向链表最后一个元素之后的迭代器。

```
1    list<int> a1{1,2,3,4,5};
2    list<int>::iterator it;
3    for(it = a1.begin();it!=a1.end();it++){
4        cout << *it << "\t";
5    }
6    cout << endl;
```

- .rbegin()　返回逆向链表的第一个元素（即链表的最后一个数据）的迭代器。
- .rend()　　返回逆向链表的最后一个元素的下一个位置,即链表的第一个数据再往前的位置。

```
1    list<int> a1{1,2,3,4,5};
2    list<int>::reverse_iterator it;
3    for(it = a1.rbegin();it!=a1.rend();it++){
4        cout << *it << "\t";
5    }
6    cout << endl;
```

- lst.front(); //访问lst第一个元素
- lst.back(); //访问lst最后一个元素

添加插入元素

- .push_front(const T& x)　头部添加元素：
- .push_back(const T& x)　末尾添加元素：

```
lst.push_front(4); // 头部增加元素
lst.push_back(5); // 末尾添加元素
```

- .insert(iterator it, const T& x)　it对应位置插入一个元素x
- .insert(iterator it, int n, const T& x)　it对应位置插入 n 个相同元素x：
- .insert(iterator it, iterator first, iterator last) 插入另一个链表的 [first,last)　间的数据：

```
list<int> lst;
list<int>::iterator it = lst.begin();
lst.insert(it, 2); // 开始位置插入一个元素2
lst.insert(lst.begin(), 3, 9); // 开始位置插入3个相同元素9
list<int> lst2(5, 8);
lst.insert(lst.begin(), lst2.begin(), ++lst2.begin());
// 插入另一个向量的[first,last)间的数据
```

容量函数

- 容器大小：lst.size();
- 容器最大容量：lst.max_size();
- 更改容器大小：lst.resize();
- 容器判空：lst.empty();

```
list<int> lst;
for (int i = 0; i<6; i++) {
    lst.push_back(i);
}

cout << lst.size() << endl; // 输出：6
cout << lst.max_size() << endl; // 输出：357913941
lst.resize(0); // 更改元素大小
cout << lst.size() << endl; // 输出：0
if (lst.empty())
    cout << "元素为空" << endl; // 输出：元素为空
```

删除元素

- 头部删除元素：lst.pop_front();
- 末尾删除元素：lst.pop_back();
- 任意位置删除一个元素：lst.erase(iterator it);
- 删除 [first,last] 之间的元素：lst.erase(iterator first, iterator last);
- 清空所有元素：lst.clear();

```
list<int> lst;
for (int i = 0; i < 8; i++)
    lst.push_back(i);
lst.pop_front(); // 头部删除元素
lst.pop_back(); // 末尾删除元素
list<int>::iterator it = lst.begin();
lst.erase(it); // 任意位置删除一个元素
lst.erase(lst.begin(), ++lst.begin()); // 删除[first,last]之间的元素
for (it = lst.begin(); it != lst.end(); it++) // 遍历显示
    cout << *it << " "; // 输出：3 4 5 6
cout << endl;
lst.clear(); // 清空所有元素
if (lst.empty()) // 判断list是否为空
    cout << "元素为空" << endl; // 输出：元素为空
```

其他函数

- 多个元素赋值：lst.assign(int n, const T& x); // 类似于初始化时用数组进行赋值
- 交换两个同类型容器的元素：swap(list&, list&); 或 lst.swap(list&);
- 合并两个列表的元素（默认升序排列）：lst.merge();
- 在任意位置拼接入另一个list：lst.splice(iterator it, list&);
- 删除容器中相邻的重复元素：lst.unique();

算法sort、reverse

- #include <algorithm>

- sort(lst.begin(), lst.end()); // 对lst链表采用的是从小到大的排序
- // 如果想从大到小排序，可以采用先排序后反转的方式
- reverse(lst.begin(), lst.end());//元素翻转

- //也可以采用下面方法，自定义从大到小的比较器，用来改变排序方式
- bool Comp(const int& a, const int& b)
- {
- 	return a > b;
- }
- sort(lst.begin(), lst.end(), Comp);

练习题 士兵队列训练问题

- 士兵进行队列训练，按顺序依次编号并排成一列，士兵报数规则：从头开始1至2报数，凡报到2的出列，剩下的向小序号方向靠拢，再从头开始进行1至3报数，凡报到3的出列，剩下的向小序号方向靠拢，...，以后从头开始轮流进行1至2报数、1至3报数直到剩下的人数不超过3人为止。
- 输入：本题有多个测试数据组，第一行为组数N，接着为N行士兵人数，人数不超过5000。
- 输出：N行剩下的士兵最初的编号，编号之间有一个空格。
- Sample Input
- 2
- 20
- 40
- Sample Output
- 1 7 19
- 1 19 37

参考代码

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int t,n;
    cin>>t;
    while(t--) {
        cin>>n;
        int k=2;
        list<int> lst;              //定义
        list<int>::iterator it;
        for(int i=1;i<=n;i++)
            lst.push_back(i);     //赋值
        while(lst.size() > 3) {
            int num = 1;
            for(it=lst.begin(); it!=lst.end();){
                if(num++ % k == 0)
                    it = lst.erase(it);
                else   it++;
            }
            k==2 ? k=3:k=2;
             //1至2报数与1至3报数切换
        }
        for(it=lst.begin();it!=lst.end(); it++){
            if (it != lst.begin())
                    cout << " ";
            cout<<*it;
        }
        cout<<endl;
    }
    return 0;
}
```